aws

Guia do Desenvolvedor

Amazon DynamoDB

Versão da API 2012-08-10

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DynamoDB: Guia do Desenvolvedor

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

As marcas comerciais e imagens comerciais da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é o Amazon DynamoDB? 1
Características 2
Sem servidor 2
NoSQL 2
Totalmente gerenciado 2
Desempenho de latência inferior a dez milissegundos em qualquer escala 3
Casos de uso 3
Capacidades 4
Replicação multiativa com tabelas globais 4
Transações ACID 5
Captura de dados de alteração 5
Índices secundários 5
Integrações de serviços 5
Integrações sem servidor 5
Importar e exportar dados para o Amazon S3 6
Integração ETL zero 6
Armazenamento em cache 6
Segurança 6
Resiliência 7
Tabelas globais 8
Backups contínuos e recuperação para um ponto no tempo 8
Backup e restauração sob demanda 8
Acessar o DynamoDB 9
Preços 9
Conceitos básicos 9
Conceitos básicos do DynamoDB 11
Recursos para usuários iniciantes 12
Práticas recomendadas adicionais do Amazon DynamoDB para usuários iniciantes 13
Recursos do AWS CLI 13
Recursos de programação 14
Acessar o DynamoDB 14
Utilizar o console 15
Uso do AWS CLI 15
Uso da API 18

Usar o NoSQL workbench 18
Intervalos de endereços IP 19
Pré-requisitos 20
Configurar o DynamoDB 20
Configurar o DynamoDB (serviço da Web) 21
Configurar o DynamoDB local (versão para download) 24
Etapa 1: criar uma tabela 49
Etapa 2: gravar dados 103
Etapa 3: ler dados 135
Etapa 4: atualizar dados 162
Etapa 5: consultar dados 196
Etapa 6: (Opcional) limpar 234
Próximas etapas 253
Como funciona 254
Folha de dicas 254
Configuração inicial 254
SDK ou CLI 255
Ações básicas 255
Regras de nomenclatura 256
Noções básicas de cota de serviço 257
Mais informações 259
Componentes principais 259
Tabelas, itens e atributos 260
Chave primária 264
Índices secundários 265
DynamoDB Streams 268
API do DynamoDB 270
Ambiente de gerenciamento 270
Plano de dados 271
DynamoDB Streams 273
Transações 273
Regras de nomenclatura e tipos de dados 274
Regras de nomenclatura 274
Tipos de dados 275
Descritores de tipo de dados 280
Classes de tabelas do DynamoDB 281

Partições e distribuição de dados no DynamoDB ... 282
Distribuição de dados: chave de partição ... 282
Distribuição de dados: chave de partição e chave de classificação ... 284
Saiba como passar do SQL para o NoSQL ... 286
Relacional ou NoSQL? ... 287
Acesso a autenticação ... 290
Criar uma tabela ... 293
Obter informações sobre uma tabela ... 296
Gravar dados em uma tabela ... 298
Ler dados de uma tabela ... 302
Gerenciamento de Índices ... 311
Modificar dados em uma tabela ... 316
Excluir dados de uma tabela ... 320
Remover uma tabela ... 322
Recursos e ferramentas de aprendizado do Amazon DynamoDB ... 323
Ferramentas para codificação e visualização ... 323
Recomendações ... 324
Centro de Conhecimentos ... 325
Postagens em blogs, repositórios e guias ... 326
Modelagem de dados e padrões de design ... 327
Cursos de treinamento ... 327
Leituras e gravações ... 328
Consistência de leitura do DynamoDB ... 328
Leituras finais consistentes ... 328
Leituras altamente consistentes ... 329
Consistência de leitura de tabelas globais ... 329
Operações de leitura e de gravação ... 329
Consumo de operações de leitura ... 329
Consumo de operações de gravação ... 331
Capacidade de throughput do DynamoDB ... 333
Modo sob demanda ... 333
Modo provisionado ... 334
Modo de capacidade sob demanda do DynamoDB ... 334
Unidades de solicitação de leitura e unidades de solicitação de gravação ... 335
Throughput inicial e propriedades de escalabilidade ... 336
Throughput máximo do DynamoDB para tabelas sob demanda ... 337

Modo de capacidade provisionada ... 339
Unidades de capacidade de leitura e de gravação ... 341
Escolher as configurações iniciais de throughput ... 341
Ajuste de escala automático do DynamoDB ... 342
Gerenciar a capacidade de throughput com o ajuste de escala automático ... 343
Capacidade reservada ... 375
Throughput a quente ... 376
Conferir o throughput a quente da sua tabela ... 377
Aumentar o throughput a quente de sua tabela ... 380
Criar uma tabela com um throughput a quente mais alto ... 387
Cenários de throughput a quente ... 398
Capacidade de expansão e capacidade adaptável ... 400
Capacidade de expansão ... 400
Capacidade adaptável ... 401
Alternar modos de capacidade ... 402
Modo provisionado para modo sob demanda ... 403
Modo sob demanda para modo provisionado ... 405
Programação com o DynamoDB ... 407
Visão geral do suporte a AWS SDKs para o DynamoDB ... 407
Suporte SDK para endpoints baseados em contas da AWS ... 409
Interfaces programáticas que funcionam com o DynamoDB ... 410
Interfaces de programação de nível superior ... 417
Executar os exemplos de código ... 480
API de baixo nível ... 487
Programar com Python ... 493
Sobre o Boto ... 494
Documentação do Boto ... 494
Camadas do cliente e de recursos ... 495
Usar o batch_writer ... 498
Exemplos de código adicionais ... 499
Segurança de sessões e de threads ... 499
Config ... 500
Tratamento de erros ... 505
Registro em log ... 507
Hooks de eventos ... 509
Paginação e o paginador ... 510

Waiters ... 512
Programar com JavaScript ... 512
Sobre o AWS SDK para JavaScript ... 513
AWS SDK para JavaScript V3 ... 513
Documentação do JavaScript ... 514
Camadas de abstração ... 514
Função de utilitário de ordenação ... 516
Ler itens ... 517
Gravações condicionais ... 519
Paginação ... 519
Config ... 522
Waiters ... 525
Gerenciamento de erros ... 526
Registro em log ... 528
Considerações ... 529
Programar com o AWS SDK for Java 2.x ... 530
Sobre o AWS SDK for Java 2.x ... 530
Conceitos básicos ... 531
Documentação do SDK para Java 2.x ... 541
Interfaces compatíveis ... 541
Exemplos de código adicionais ... 556
Programação sincronizada e assíncrona ... 556
Clientes HTTP ... 557
Config ... 559
Gerenciamento de erros ... 566
ID da solicitação da AWS ... 567
Registro em log ... 567
Paginação ... 570
Anotações de classes de dados ... 571
Tratamento de erros ... 572
Componentes de erros ... 572
Erros transacionais ... 573
Mensagens e códigos de erro ... 573
Tratamento de erros na aplicação ... 579
Repetições de erro e recuo exponencial ... 579
Operações em lote e tratamento de erros ... 580

Como trabalhar com AWS SDKs ... 581
Como trabalhar com o DynamoDB ... 583
Trabalhar com tabelas ... 583
Operações básicas nas tabelas ... 584
Considerações ao escolher uma classe de tabela no DynamoDB ... 593
Etiquetas e rótulos ... 594
Trabalhar com tabelas globais ... 600
Como funciona ... 601
Tutoriais: Criação de tabelas globais ... 610
Segurança de tabelas globais ... 648
Faturamento para tabelas globais ... 653
Versões de tabelas globais ... 656
Melhores práticas de tabelas globais ... 667
Trabalhar com itens ... 669
Tamanhos e formatos de item ... 670
Ler um item ... 671
Gravar um item ... 672
Retornar valores ... 675
Operações em lote ... 676
Contadores atômicos ... 678
Gravações condicionais ... 679
Usar expressões ... 686
Vida útil (TTL) ... 727
Consultar tabelas ... 758
Verificar tabelas ... 767
Linguagem de consultas PartiQL ... 776
Trabalhar com itens: Java ... 824
Trabalhar com itens: .NET ... 856
Trabalhar com índices ... 889
Índices secundários globais ... 894
Índices secundários locais no DynamoDB ... 957
Trabalhar com transações ... 1013
Como funciona ... 1014
Usar o IAM com transações ... 1023
Código de exemplo ... 1026
Trabalhar com fluxos ... 1030

Opções ... 1031
Como trabalhar com o Kinesis Data Streams ... 1033
Como trabalhar com DynamoDB Streams ... 1051
Aceleração em memória com o DAX ... 1134
Casos de uso para o DAX ... 1135
Observações sobre o uso do DAX ... 1136
Como funciona ... 1137
Como o DAX processa solicitações ... 1139
Cache de itens ... 1141
Cache de consultas ... 1142
Componentes de cluster do DAX ... 1143
Nós ... 1143
Clusters ... 1144
Regiões e zonas de disponibilidade ... 1145
Grupos de parâmetros ... 1146
Grupos de segurança ... 1146
ARN do cluster ... 1146
Endpoint do cluster ... 1147
Endpoints de nó ... 1147
Grupos de sub-redes ... 1147
Eventos ... 1148
Janela de manutenção ... 1148
Criar um cluster do DAX ... 1149
Criar um perfil de serviço do IAM para o DAX acessar o DynamoDB ... 1150
Como usar o AWS CLI ... 1152
Usar o console ... 1158
Modelos de consistência ... 1163
Consistência entre nós de cluster do DAX ... 1164
Comportamento do cache de itens do DAX ... 1165
Comportamento do cache de consultas DAX ... 1168
Leituras altamente consistentes e transacionais ... 1169
Armazenamento em cache negativo ... 1169
Estratégias para gravações ... 1170
Desenvolver com o cliente do DAX ... 1173
Tutorial: executar um aplicativo de exemplo ... 1174
Como modificar uma aplicação existente para usar o DAX ... 1239

Gerenciar clusters do DAX 1241
Permissões do IAM para gerenciar um cluster do DAX 1241
Escalar um cluster do DAX 1244
Personalizar as configurações do cluster do DAX 1246
Configurar definições de TTL 1247
Compatibilidade com marcação para o DAX 1248
Integração do AWS CloudTrail 1249
Excluir um cluster do DAX 1250
Monitorar o DynamoDB Accelerator 1250
Ferramentas de monitoramento do DAX 1251
Monitoramento com CloudWatch 1252
Registrar operações do DAX usando o AWS CloudTrail 1279
Instâncias expansíveis T3/T2 do DAX 1280
Família de instâncias T2 do DAX 1280
Família de instâncias T3 do DAX 1280
Controle de acesso do DAX 1281
Perfil de serviço do IAM para o DAX 1282
Política do IAM para permitir acesso a cluster do DAX 1284
Estudo de caso: acesso ao DynamoDB e ao DAX 1285
Acesso ao DynamoDB, mas sem acesso com o DAX 1287
Acesso ao DynamoDB e ao DAX 1289
Acesso ao DynamoDB via DAX, mas sem acesso direto ao DynamoDB 1295
Criptografia em repouso do DAX 1297
Habilitar a criptografia em repouso usando o AWS Management Console 1300
Criptografia em trânsito do DAX 1301
Usar funções vinculadas ao serviço para o DAX 1301
Permissões de função vinculada ao serviço para o DAX 1302
Criar uma função vinculada ao serviço do DAX 1304
Editar uma função vinculada ao serviço do DAX 1304
Excluir uma função vinculada ao serviço do DAX 1304
Acessando o DAX em contas da AWS 1306
Configurar o IAM 1306
Configurar uma VPC 1309
Modificar o cliente do DAX para permitir acesso entre contas 1312
Guia de dimensionamento de clusters do DAX 1316
Visão geral 1317

Estimar tráfego ... 1317
Testes de carga ... 1318
Modelagem de dados ... 1321
Trabalhar com coleções de itens ... 1322
Acelerar as consultas organizando os dados com coleções de itens ... 1324
Fundamentos da modelagem de dados ... 1324
Design de tabela única ... 1325
Design de várias mesas ... 1327
Componentes básicos da modelagem de dados ... 1329
Chave de classificação composta ... 1329
Multilocação ... 1331
Índice esparso ... 1332
Vida útil ... 1334
Vida útil para arquivamento ... 1335
Particionamento vertical ... 1336
Fragmentação de gravação ... 1339
Pacotes de design de esquema de modelagem de dados ... 1340
Pré-requisitos ... 1341
Rede social ... 1342
Perfil de jogo ... 1351
Sistema de gerenciamento de reclamações ... 1360
Pagamentos recorrentes ... 1379
Atualizações de status de dispositivos ... 1384
Loja on-line ... 1398
Modelagem relacional ... 1422
Modelos tradicionais de banco de dados relacional ... 1423
Como o DynamoDB elimina a necessidade de operações JOIN ... 1425
Como as transações do DynamoDB eliminam a sobrecarga no processo de gravação ... 1426
Primeiras etapas ... 1427
Exemplo ... 1429
Migração para o DynamoDB ... 1434
Motivos para migrar ... 1434
Considerações ao migrar ... 1436
Como funciona ... 1438
Ferramentas de migração ... 1439
Selecionar uma estratégia de migração ... 1439

Migração offline ... 1442
Migração híbrida ... 1444
On-line: migrar cada tabela 1:1 ... 1446
On-line: migrar com uma tabela de preparação personalizada. ... 1447
NoSQL Workbench ... 1450
Baixar ... 1451
Instalar ... 1452
Modelador de dados ... 1460
Criar um novo modelo ... 1460
Importar um modelo existente ... 1467
Exportar um modelo ... 1470
Editar um modelo existente ... 1472
Visualizador de dados ... 1476
Adicionar dados de exemplo ... 1476
Importação do CSV ... 1479
Facetas ... 1480
Visualização agregada ... 1483
Confirmar um modelo de dados ... 1484
Criador de operações ... 1487
Conectar-se a conjuntos de dados ... 1488
Criar operações ... 1489
Clonar tabelas ... 1501
Exportar para CSV ... 1503
Modelos de dados de amostra ... 1504
Modelo de dados de funcionários ... 1504
Modelo de dados de fórum de discussão ... 1505
Modelo de dados de biblioteca de música ... 1505
Modelo de dados de estação de esqui ... 1506
Modelo de dados de ofertas de cartão de crédito ... 1506
Modelo de dados de favoritos ... 1507
Histórico de versões ... 1507
Backup e restauração ... 1515
Backups para um ponto no tempo ... 1517
Antes de começar ... 1517
Habilitar a recuperação para um ponto no tempo ... 1518
Backups sob demanda ... 1522

Como funciona ... 1523
Fazer backup de uma tabela ... 1526
Restaurar uma tabela ... 1529
Excluir um backup de tabela ... 1535
Usar o IAM ... 1536
Faturamento para backups ... 1544
Como funciona ... 1544
Exemplo de faturamento de backup do DynamoDB ... 1545
Restaurações ... 1550
Restaurar uma tabela usando a recuperação pontual ... 1550
Restaurar uma tabela do DynamoDB para um ponto no tempo ... 1552
Usar o Backup do AWS ... 1558
Como funciona ... 1560
Criar backups ... 1563
Copiar backups ... 1564
Restaurar uma tabela ... 1566
Excluir backups ... 1567
Backups sob demanda gerenciados pelo AWS Backup versus DynamoDB ... 1567
Exemplos de código ... 1569
Conceitos básicos ... 1582
Olá, DynamoDB ... 1583
Conheça os conceitos básicos ... 1593
Ações ... 1750
Cenários ... 2159
Acelerar leituras com o DAX ... 2162
Cenários avançados do índice secundário global ... 2171
Criar uma aplicação para enviar dados para uma tabela do DynamoDB ... 2174
Comparar vários valores com um único atributo ... 2175
Atualizar condicionalmente a TTL de um item ... 2198
Conecte-se a uma instância local ... 2205
Contar operadores de expressão ... 2206
Criar uma API REST para monitorar dados da COVID-19 ... 2239
Criar uma aplicação de mensageiro ... 2240
Criar uma aplicação com tecnologia sem servidor para gerenciar fotos ... 2241
Criar uma tabela com um índice secundário global ... 2245
Criar uma tabela com o throughput a quente habilitado ... 2250

Criar uma aplicação Web para monitorar dados do DynamoDB 2260
Criar uma aplicação de chat websocket 2262
Criar um item com TTL 2262
Criar e gerenciar tabelas globais com MRSC 2269
Crie e gerencie tabelas globais demonstrando o MREC 2289
Excluir dados usando DELETE do PartiQL 2299
Detectar EPI em imagens 2305
Inserir dados usando INSERT do PartiQL 2306
Invocar uma função do Lambda em um navegador 2310
Gerenciar índices secundários globais 2311
Gerenciar políticas baseadas em recursos 2313
Monitoramento do desempenho do DynamoDB 2316
Executar operações de consulta avançada 2317
Executar operações de lista 2352
Executar operações de mapa 2385
Executar operações de conjunto 2409
Consultar uma tabela usando lotes de instruções PartiQL 2439
Consultar uma tabela usando o PartiQL 2502
Consultar uma tabela usando um índice secundário global 2556
Consultar uma tabela usando uma condição begins_with 2565
Consulte uma tabela usando um intervalo de datas 2571
Consultar uma tabela com uma expressão de filtro complexa 2580
Consultar uma tabela com uma expressão de filtro dinâmica 2588
Consultar uma tabela com uma expressão de filtro e limite 2597
Consultar uma tabela com atributos aninhados 2602
Consultar uma tabela com paginação 2611
Consultar uma tabela com leituras altamente consistentes 2622
Consultar dados usando SELECT do PartiQL 2626
Consultar itens com TTL 2632
Consultar tabelas usando padrões de data e hora 2636
Salvar o EXIF e outras informações de imagem 2651
Configuração do controle de acesso por atributo 2652
Como funciona a ordem da expressão de atualização 2655
Atualizar a configuração de throughput a quente de uma tabela 2686
Atualiza a TTL de um item 2692
Atualizar dados usando UPDATE do PartiQL 2696

Usar o API Gateway para invocar uma função do Lambda ... 2703
Usar Step Functions para invocar funções do Lambda ... 2705
Usar um modelo de documento ... 2705
Usar um modelo de persistência de objetos de alto nível ... 2722
Usar operações de contador atômico ... 2731
Usar operações condicionais ... 2754
Usar nomes de atributo de expressão ... 2784
Usar eventos programados para invocar uma função do Lambda ... 2812
Trabalhar com índices secundários locais ... 2814
Trabalho com os fluxos e com a vida útil ... 2816
Trabalhar com tabelas globais e a consistência final de replicação mutirregional (MREC) . 2820
Trabalhar com a marcação de recursos ... 2824
Trabalhar com a criptografia de tabelas ... 2827
Exemplos sem servidor ... 2829
Invocar uma função do Lambda em um gatilho do DynamoDB ... 2829
Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB ... 2838
Contribuições da comunidade da AWS ... 2849
Compilar e testar uma aplicação com tecnologia sem servidor ... 2850
Segurança ... 2852
Políticas gerenciadas AWS ... 2853
Políticas gerenciadas AWS ... 2853
Política gerenciada pela AWS: AmazonDynamoDBFullAccess_v2 ... 2854
AmazonDynamoDBReadOnlyAccess ... 2854
Atualizações do DynamoDB para políticas gerenciadas pela AWS ... 2856
Políticas baseadas em recursos ... 2858
Create table ... 2859
Associar uma política baseada em recursos ... 2865
Associar uma política a um fluxo ... 2870
política uma política baseada em recursos ... 2873
Acesso entre contas ... 2873
Bloquear o acesso público ... 2875
Operações de API ... 2878
Autorização do IAM ... 2886
Exemplos ... 2887
Considerações ... 2894
Práticas recomendadas ... 2895

Controle de acesso por atributo 2897
Por que devo usar o ABAC? 2899
Chaves de condição 2899
Considerações 2900
Habilitar o ABAC no DynamoDB 2900
Usar o ABAC 2903
Exemplo de casos de uso 2905
Solução de problemas 2919
Proteção de dados 2920
Criptografia em repouso 2921
Proteger as conexões do DynamoDB 2948
IAM 2950
Gerenciamento de Identidade e Acesso 2950
Usar condições 2987
Validação de conformidade 3013
Resiliência 3014
Segurança da infraestrutura 3015
Usar endpoints da VPC 3015
AWS PrivateLink para DynamoDB 3025
Tipos de endpoint da Amazon VPC 3026
Considerações ao usar o AWS PrivateLink para Amazon DynamoDB 3027
Criar um Amazon VPC endpoint 3028
Acessar os endpoints de interface do Amazon DynamoDB 3028
Acessar tabelas do DynamoDB e operações de API de controle por meio dos endpoints da interface do DynamoDB 3028
Atualizar uma configuração de DNS on-premises 3030
Criar uma política de endpoint da Amazon VPC 3033
AWS PrivateLink para DynamoDB Streams 3034
Análise de configuração e vulnerabilidade 3039
Práticas recomendadas de segurança 3040
Práticas recomendadas de segurança preventiva 3040
Práticas recomendadas de segurança de detecção 3043
Monitorar e registrar em log 3047
Plano de monitoramento 3047
Linha de base de performance 3047
Serviços integrados 3048

Ferramentas de monitoramento automatizadas ... 3048
Monitoramento de métricas ... 3049
Como usar as métricas do DynamoDB? ... 3050
Visualizar métricas no console do CloudWatch ... 3051
Visualizar métricas na AWS CLI ... 3051
Métricas e dimensões ... 3052
Criar alarmes do CloudWatch no DynamoDB ... 3084
Operações de registro em log ... 3087
Informações do DynamoDB no CloudTrail ... 3088
Noções básicas das entradas dos arquivos de log do DynamoDB ... 3092
Contributor Insights ... 3111
Como funciona ... 3111
Começar ... 3123
Usar o IAM ... 3132
Práticas recomendadas ... 3139
Design em NoSQL ... 3139
NoSQL vs. RDBMS ... 3140
Dois conceitos-chave ... 3140
Abordagem geral ... 3141
NoSQL Workbench ... 3142
Lentes do Well-Architected para DynamoDB ... 3143
Otimização de custo ... 3143
Realizar a análise do Lentes do Well-Architected para Amazon DynamoDB ... 3194
Pilares do Lentes do Well-Architected para Amazon DynamoDB ... 3194
Design de chave de partição ... 3197
Distribuir workloads ... 3198
Fragmentação de gravação ... 3199
Carregar dados eficientemente ... 3201
Design de chaves de classificação ... 3202
Controle de versões ... 3203
Índices secundários ... 3204
Diretrizes gerais ... 3205
Índices esparsos ... 3208
Agregação ... 3210
Sobrecarga do GSI ... 3212
Fragmentação do GSI ... 3213

Criar uma réplica ... 3218
Itens grandes ... 3219
Compactação ... 3220
Particionamento vertical ... 3220
Uso do Amazon S3 ... 3221
Dados de séries temporais ... 3221
Padrão de design para dados de séries temporais ... 3222
Exemplos de tabelas de séries temporais ... 3222
Relações de muitos para muitos ... 3223
Listas de adjacências ... 3224
Gráficos materializados ... 3225
Consultar e verificar ... 3230
Desempenho de verificação ... 3230
Evitar picos ... 3231
Verificações paralelas ... 3234
Design de tabelas ... 3235
Design de tabelas globais ... 3235
Design de tabelas globais ... 3236
Fatos importantes ... 3236
Casos de uso ... 3238
Modos de gravação ... 3239
Roteamento de solicitações ... 3247
Evacuar uma região ... 3256
Capacidade de throughput com tabelas globais ... 3259
Lista de verificação e perguntas frequentes para tabelas globais ... 3261
Ambiente de gerenciamento ... 3268
Operações de dados em massa ... 3269
Atualização condicional em lote ... 3269
Operações em massa eficientes ... 3275
Implementar controle de versão ... 3276
Quando usar esse padrão ... 3277
Padrão de design ... 3277
Uso do padrão ... 3278
Relatórios de uso e faturamento ... 3281
Capacidade de throughput ... 3284
Fluxos ... 3288

Armazenamento ... 3289
Backup e restauração ... 3290
Transferência de dados ... 3294
CloudWatch ... 3295
DAX ... 3296
Migrar uma tabela do DynamoDB de uma conta para outra ... 3298
Migrar uma tabela usando o AWS Backup para backup e restauração entre contas ... 3299
Migrar uma tabela usando a exportação para o S3 e a importação do S3 ... 3301
Recomendações do DAX ... 3303
Avaliar a adequação do DAX ... 3304
Configurar clientes do DAX ... 3307
Configurar um cluster do DAX ... 3308
Dimensionar um cluster do DAX ... 3314
Implantar um cluster ... 3321
Operações de cluster ... 3323
Monitoramento do DAX ... 3326
Usar o DynamoDB com outros serviços da AWS ... 3329
Integração com o Amazon Cognito ... 3329
Integração com o Amazon Redshift ... 3332
Considerações sobre a integração entre contas com a CMK ... 3332
Integração ETL zero com o Amazon Redshift ... 3336
Enviar dados do DynamoDB ao Amazon Redshift com COPY ... 3344
Integração com o Amazon EMR ... 3346
Visão geral ... 3346
Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive ... 3347
Criar uma tabela externa no Hive ... 3356
Processar instruções em HiveQL ... 3360
Consultar dados no DynamoDB ... 3361
Copiar dados de e para o Amazon DynamoDB ... 3364
Ajuste de performance ... 3378
Integração com o S3 ... 3384
Importar do Amazon S3 ... 3384
Exportar para o Amazon S3. ... 3405
Integração com o Amazon SageMaker Lakehouse ... 3428
Integração ETL zero com o Amazon SageMaker Lakehouse ... 3428
Integrar ao Amazon OpenSearch Service ... 3430

Como funciona ... 3430
Criar uma integração ... 3431
Próximas etapas ... 3432
Lidar com alterações significativas ... 3432
Integração ETL zero com o OpenSearch Service ... 3436
Integração ao Amazon EventBridge ... 3439
Como funciona ... 3439
Criar uma integração por meio do console ... 3440
Próximas etapas ... 3443
Integração com o Amazon MSK ... 3443
Como funciona ... 3444
Exemplo de integração ... 3445
Próximas etapas ... 3451
Práticas recomendadas de integração ... 3452
Criar um snapshot ... 3452
Captura de dados de alteração ... 3452
Usar IA generativa com o DynamoDB ... 3454
Casos de uso de IA generativa para o DynamoDB ... 3454
Blogs de IA generativa para o DynamoDB ... 3455
Como aproveitar a integração ETL zero do DynamoDB com o OpenSearch Service ... 3455
Cotas e restrições ... 3457
Realizar tarefas de gerenciamento de cotas ... 3457
Acessar as cotas do DynamoDB ... 3457
Visualizar as cotas atuais no console ... 3458
Visualizar as cotas atuais usando a AWS CLI ... 3458
Solicitar um aumento de cota ... 3460
Cotas ... 3461
Throughput de leitura e gravação ... 3462
Capacidade reservada ... 375
Tabelas ... 3465
Tabelas globais ... 3465
Índices secundários ... 3467
Atributos projetados do índice secundário ... 3467
DynamoDB Streams ... 3467
Importar do Amazon S3 ... 3468
Exportação de tabela para o Amazon S3 ... 3468

Backup e restauração ... 3468
Contributor Insights ... 3468
Restrições ... 3468
Modo de capacidade de leitura/gravação ... 3469
Índices secundários ... 3467
Chaves de partição e chaves de classificação ... 3470
Regras de nomenclatura ... 3471
Tipos de dados ... 3472
Itens ... 3473
Atributos ... 3473
Parâmetros de expressão ... 3474
Transações do DynamoDB ... 3475
DynamoDB Streams ... 3475
DynamoDB Accelerator (DAX) ... 3475
Restrições específicas da API ... 3476
Criptografia em repouso do DynamoDB ... 3479
Referência da API ... 3480
Solução de problemas ... 3481
Erros internos do servidor ... 3481
Investigar erros internos do servidor ... 3482
Minimizar o impacto dos erros internos do servidor ... 3483
Melhorar o reconhecimento operacional ... 3483
Latência ... 3487
Controle de utilização ... 3491
Diagnosticar o controle de utilização ... 3493
Guia de resolução ... 3500
Controle de utilização de gravação e contrapressão do GSI ... 3528
Métricas de controle de utilização do CloudWatch ... 3530
Apêndice ... 3533
Solução de problemas de estabelecimento de conexão SSL/TLS com o DynamoDB ... 3533
Como testar seu aplicativo ou serviço ... 3533
Como testar o navegador cliente ... 3534
Como atualizar o aplicativo de software cliente ... 3534
Como atualizar o navegador cliente ... 3535
Como atualizar manualmente seu pacote de certificado ... 3535
Exemplos de tabelas e dados para uso no DynamoDB ... 3536

Arquivos de dados de amostra 3537
Criar exemplos de tabelas e carregar dados 3550
Criar exemplos de tabelas e carregar dados - Java 3550
Criar exemplos de tabelas e carregar dados - .NET 3560
Aplicação de exemplo em AWS SDK para Python (Boto3) 3572
Etapa 1: implantar e testar localmente 3573
Etapa 2: examinar o modelo de dados e os detalhes da implantação 3578
Etapa 3: implantar em produção 3588
Etapa 4: Limpar os recursos 3598
Palavras reservadas no DynamoDB 3598
Exemplos do AWS SDK for Java 1.x 3611
DAX e Java SDK v1 3612
Modificar uma aplicação do SDK for Java 1.x existente para usar o DAX 3624
Consultar índices secundários globais com o SDK for Java 1.x 3629
Exemplos do SDK da AWS para Go 1.x 3633
Go e DAX 3633
Exemplos do SDK da AWS para Node.js 2.x 3636
Node.js e DAX 3636
Histórico do documento 3647
Atualizações anteriores 3685
Recursos herdados 3721
Global Tables versão 2017.11.29 (herdada) 3721
Como funciona 3722
Melhores práticas e requisitos 3727
Criar uma tabela global 3731
Monitorar tabelas globais 3736
Usar o IAM com tabelas globais 3738
Versão anterior da API de baixo nível do DynamoDB (5/12/2011) 3741
BatchGetItem 3742
BatchWriteItem 3750
CreateTable 3757
DeleteItem 3766
DeleteTable 3773
DescribeTables 3777
GetItem 3781
ListTables 3785

PutItem 3788
Consulta 3796
Verificar 3813
UpdateItem 3832
UpdateTable 3842
Parâmetros condicionais legados do DynamoDB 3847
AttributesToGet 3849
AttributeUpdates 3850
ConditionalOperator 3853
Esperados 3853
KeyConditions 3859
QueryFilter 3862
ScanFilter 3865
Criar condições com parâmetros herdados 3867

O que é o Amazon DynamoDB?

O Amazon DynamoDB é um banco de dados sem servidor, NoSQL e totalmente gerenciado com desempenho de latência inferior a dez milissegundos em qualquer escala.

O DynamoDB atende às suas necessidades para superar as complexidades operacionais e de escalabilidade dos bancos de dados relacionais. O DynamoDB tem propósito específico e é otimizado para workloads operacionais que exigem desempenho consistente em qualquer escala. Por exemplo, o DynamoDB oferece desempenho consistente com latência inferior a dez milissegundos para um caso de uso de carrinho de compras, independentemente de você ter dez ou cem milhões de usuários. Lançado em 2012, o DynamoDB continua ajudando você a abandonar os bancos de dados relacionais e, ao mesmo tempo, reduzir os custos e melhorar o desempenho em grande escala.

Clientes de todos os portes, setores e regiões usam o DynamoDB para criar aplicações modernas sem servidor que podem começar pequenas e escalar globalmente. O DynamoDB é escalável para oferecer suporte a tabelas de praticamente qualquer tamanho, ao mesmo tempo em que fornece desempenho consistente com latência inferior a dez milissegundos e alta disponibilidade.

Para eventos como o Amazon Prime Day, o DynamoDB alimenta várias propriedades e sistemas da Amazon de alto tráfego, incluindo a Alexa, os sites da Amazon.com e todos os centros de distribuição da Amazon. Para eventos desse tipo, as APIs do DynamoDB já processaram trilhões de chamadas de propriedades e sistemas da Amazon. O DynamoDB atende continuamente centenas de clientes com tabelas que têm pico de tráfego de mais de meio milhão de solicitações por segundo. Ele também atende centenas de clientes com tabelas que excedem 200 TB, processando mais de um bilhão de solicitações por hora.

Tópicos

- Características do DynamoDB
- Casos de uso do DynamoDB
- Recursos do DynamoDB
- Integrações de serviços
- Segurança
- Resiliência
- Acessar o DynamoDB

- Preços do DynamoDB
- Conceitos básicos do DynamoDB

Características do DynamoDB

Sem servidor

Com o DynamoDB, você não precisa provisionar nenhum servidor nem corrigir, gerenciar, instalar, manter ou operar nenhum software. O DynamoDB oferece manutenção sem nenhum tempo de inatividade. Ele não tem versões (principal, secundária ou patch) e não há janelas de manutenção.

O modo de capacidade sob demanda do DynamoDB oferece preço conforme o uso para solicitações de leitura e gravação, para que você pague apenas pelo que usar. Com o modo sob demanda, o DynamoDB aumenta ou reduz instantaneamente a escala vertical das tabelas para ajustar a capacidade e manter o desempenho sem nenhuma administração. Ele também reduz a escala vertical para zero, para que você não pague por throughput quando sua tabela não tem tráfego e não há partidas a frio.

NoSQL

Como banco de dados NoSQL, o DynamoDB foi desenvolvido especificamente para oferecer melhor desempenho, escalabilidade, capacidade de gerenciamento e flexibilidade em comparação aos bancos de dados relacionais tradicionais. Para oferecer suporte a uma ampla variedade de casos de uso, o DynamoDB é compatível com modelos de dados de chave e valor e de documentos.

Ao contrário dos bancos de dados relacionais, o DynamoDB não oferece suporte a um operador JOIN. Recomendamos que você desnormalize seu modelo de dados para reduzir as viagens de ida e volta do banco de dados e a capacidade de processamento necessária para responder às consultas. Como banco de dados NoSQL, o DynamoDB oferece uma consistência de leitura robusta e transações ACID para criar aplicações de nível corporativo.

Totalmente gerenciado

Como um serviço de banco de dados totalmente gerenciado, o DynamoDB lida com o trabalho pesado indiferenciado de gerenciar um banco de dados, para que você possa se concentrar na criação de valor para seus clientes. Ele lida com instalação, configurações, manutenção, alta disponibilidade, provisionamento de hardware, segurança, backups, monitoramento e muito mais. Isso garante que, ao criar uma tabela do DynamoDB, ela esteja instantaneamente pronta para

workloads de produção. O DynamoDB melhora constantemente sua disponibilidade, confiabilidade, desempenho, segurança e funcionalidade sem exigir nenhuma atualização ou tempo de inatividade.

Desempenho de latência inferior a dez milissegundos em qualquer escala

O DynamoDB foi desenvolvido especificamente para oferecer melhor desempenho e escalabilidade do que os bancos de dados relacionais, proporcionando desempenho de latência inferior a dez milissegundos em qualquer escala. Para alcançar essa escala e desempenho, o DynamoDB é otimizado para workloads de alto desempenho e fornece APIs que incentivam o uso eficiente do banco de dados. Ele omite recursos que são ineficientes e não apresentam boa performance em grande escala, como as operações JOIN. O DynamoDB oferece desempenho consistente com latência inferior a dez milissegundos para sua aplicação, independentemente de você ter cem ou cem milhões de usuários.

Casos de uso do DynamoDB

Clientes de todos os portes, setores e regiões usam o DynamoDB para criar aplicações modernas sem servidor que podem começar pequenas e escalar globalmente. O DynamoDB é ideal para casos de uso que exigem desempenho consistente em qualquer escala com pouca ou nenhuma sobrecarga operacional. A seguinte lista apresenta alguns casos de uso em que você pode usar o DynamoDB:

- Aplicações de serviços financeiros: suponha que você seja uma empresa de serviços financeiros que desenvolve aplicações, como negociação e roteamento em tempo real, gerenciamento de empréstimos, geração de tokens e registros de transações. Com as tabelas globais do DynamoDB, suas aplicações podem responder a eventos e fornecer tráfego das Regiões da AWS escolhidas com desempenho rápido e local de leitura e gravação.

  O DynamoDB é adequado para aplicações com os requisitos de disponibilidade mais rigorosos. Ele elimina a carga operacional de escalar manualmente as instâncias para aumentar o armazenamento ou o throughput, versionamento e licenciamento.

  Você pode usar as transações do DynamoDB para alcançar atomicidade, consistência, isolamento e durabilidade (ACID) em uma ou mais tabelas com uma única solicitação. As transações (ACID) são adequadas para workloads que incluem processamento de transações financeiras ou atendimento de pedidos. O DynamoDB acomoda instantaneamente a expansão e a redução das workloads, permitindo que você escale o banco de dados de maneira eficiente de acordo com as condições do mercado, como o horário de negociação.

- Aplicações de jogos: como empresa de jogos, você pode usar o DynamoDB para todas as partes das plataformas de jogos, como estado do jogo, dados de jogadores, histórico de sessões e tabelas de classificação. Escolha o DynamoDB por sua escala, desempenho consistente e facilidade de operação proporcionada por sua arquitetura sem servidor. O DynamoDB é adequado para arquiteturas que podem aumentar a escala horizontalmente, necessárias para desenvolver jogos bem-sucedidos. Ele consegue aumentar e reduzir (escalar para zero sem partida a frio) rapidamente o throughput do seu jogo. Essa escalabilidade otimiza a eficiência da sua arquitetura, esteja você aumentando a escala horizontalmente para picos de tráfego ou reduzindo a escala em cenários de baixa jogabilidade.
- Aplicações de streaming: empresas de mídia e entretenimento usam o DynamoDB como um índice de metadados para conteúdo, serviço de gerenciamento de conteúdo ou para fornecer estatísticas esportivas quase em tempo real. Elas também usam o DynamoDB para executar serviços de listas de títulos assistidos e favoritos de usuários e para processar bilhões de eventos diários de clientes para gerar recomendações. Esses clientes se beneficiam da escalabilidade, do desempenho e da resiliência do DynamoDB. O DynamoDB ajusta a escala à medida que a workload aumenta ou diminui, permitindo casos de uso de streaming de mídia que podem oferecer suporte a qualquer nível de demanda.

Para saber mais sobre como clientes de diferentes setores usam o DynamoDB, consulte Amazon DynamoDB Customers e This is My Architecture.

Recursos do DynamoDB

Replicação multiativa com tabelas globais

As tabelas globais fornecem replicação multiativa dos dados entre as Regiões da AWS escolhidas com disponibilidade de 99,999%. As tabelas globais fornecem uma solução totalmente gerenciada para a implantação de um banco de dados multirregional e multiativo, sem precisar criar e manter uma solução de replicação própria. Com as tabelas globais, é possível especificar as Regiões da AWS em que você deseja que as tabelas estejam disponíveis. O DynamoDB replica as alterações de dados contínuas para todas essas tabelas.

Aplicações distribuídas globalmente podem acessar dados localmente nas regiões selecionadas para alcançar desempenho de leitura e gravação com latência inferior a dez milissegundos. Como as tabelas globais são multiativas, você não precisa de uma tabela primária. Isso significa que não há nenhum failover complicado ou atrasado nem tempo de inatividade do banco de dados ao realizar failover de uma aplicação entre regiões.

Transações ACID

O DynamoDB foi criado para workloads críticas. Ele inclui suporte a transações (ACID) para aplicações que exigem lógica comercial complexa. O DynamoDB oferece suporte nativo do lado do servidor a transações, simplificando a experiência do desenvolvedor ao fazer alterações coordenadas de tudo ou nada em vários itens dentro e entre tabelas.

Captura de dados de alteração para arquiteturas orientadas por eventos

O DynamoDB oferece suporte a streaming de registros de captura de dados de alteração (CDC) em nível de item quase em tempo real. Ele oferece dois modelos de streaming para CDC: DynamoDB Streams e Kinesis Data Streams para DynamoDB. Sempre que uma aplicação cria, atualiza ou exclui itens em uma tabela, o Streams registra uma sequência em ordem temporal de cada alteração em nível de item quase em tempo real. Isso torna o DynamoDB Streams ideal para que aplicações com arquitetura orientada a eventos consumam e ajam de acordo com as mudanças.

Índices secundários

O DynamoDB oferece a opção de criar índices secundários globais e locais, que permitem consultar os dados da tabela usando uma chave alternativa. Com esses índices secundários, você pode acessar dados com atributos diferentes da chave primária, oferecendo flexibilidade máxima para acessar seus dados.

Integrações de serviços

O DynamoDB se integra amplamente a vários Serviços da AWS para ajudar você a obter mais valor de seus dados, eliminar o trabalho pesado indiferenciado e operar workloads em grande escala. Alguns exemplos são: AWS CloudFormation, Amazon CloudWatch, Amazon S3, AWS Identity and Access Management (IAM) e AWS Auto Scaling. As seções a seguir descrevem algumas das integrações de serviços que você pode realizar usando o DynamoDB:

Integrações sem servidor

Para desenvolver aplicações sem servidor de ponta a ponta, o DynamoDB se integra nativamente a vários Serviços da AWS sem servidor. Por exemplo, você pode integrar o DynamoDB ao AWS Lambda para criar acionadores, que são trechos de código que respondem automaticamente a eventos no DynamoDB Streams. Com os acionadores, você pode criar aplicações orientadas por eventos que reagem às modificações de dados em tabelas do DynamoDB. Para otimizar os custos, você pode filtrar os eventos que o Lambda processa de um fluxo do DynamoDB.

A seguinte lista apresenta alguns exemplos de integrações sem servidor com o DynamoDB:

- AWS AppSync para criar APIs do GraphQL
- Amazon API Gateway para criar APIs REST
- Lambda para computação sem servidor
- Amazon Kinesis Data Streams para captura de dados de alteração (CDC)

Importar e exportar dados para o Amazon S3

A integração do DynamoDB ao Amazon S3 permite que você exporte dados com facilidade para um bucket do Amazon S3 para fins de analytics e machine learning. O DynamoDB oferece suporte a exportações de tabelas inteiras e exportações incrementais para exportar dados alterados, atualizados ou excluídos em um período especificado. Também é possível importar dados do Amazon S3 para uma nova tabela do DynamoDB.

Integração ETL zero

O DynamoDB oferece suporte à integração ETL zero com o Amazon Redshift e usando um pipeline de ingestão do OpenSearch com o Amazon DynamoDB. Essas integrações permitem que você execute tarefas complexas de analytics e use recursos avançados de pesquisa nos dados da tabela do DynamoDB. Por exemplo, você pode realizar pesquisa vetorial e de texto completo e pesquisa semântica nos dados do DynamoDB. As integrações ETL zero não afetam as workloads de produção executadas no DynamoDB.

Armazenamento em cache

O DynamoDB Accelerator (DAX) é um serviço de armazenamento em cache totalmente gerenciado e altamente disponível criado para o DynamoDB. O DAX proporciona melhorias de até dez vezes na performance, de milissegundos para microssegundos, mesmo com milhões de solicitações por segundo. O DAX faz todo o trabalho pesado necessário para adicionar aceleração na memória às suas tabelas do DynamoDB, sem exigir que você gerencie a invalidação de cache, o preenchimento de dados ou o gerenciamento de clusters.

Segurança

O DynamoDB utiliza o IAM para ajudar você a controlar com segurança o acesso a recursos do DynamoDB. Com o IAM, é possível gerenciar de maneira centralizada as permissões que controlam

quais usuários do DynamoDB podem acessar os recursos. Você usa o IAM para controlar quem é autenticado (fez login) e autorizado (tem permissões) a usar os recursos. Como o DynamoDB utiliza o IAM, não há nenhum nome de usuário ou senha para acessar o DynamoDB. Como você não tem nenhuma política complicada de rotação de senhas para gerenciar, sua postura de segurança é simplificada. Com o IAM, também é possível habilitar um controle de acesso refinado para fornecer autorização no nível do atributo. Você também pode definir políticas baseadas em recurso compatíveis com o IAM Access Analyzer e o Bloqueio de Acesso Público (BPA) para simplificar o gerenciamento de políticas.

Por padrão, o DynamoDB criptografa todos os dados de clientes em repouso. A criptografia em repouso aumenta a segurança dos dados, usando chaves de criptografia armazenadas no AWS Key Management Service(AWS KMS). Com a criptografia de dados em repouso, você pode criar aplicativos confidenciais que atendem a requisitos rigorosos de conformidade e regulamentação de criptografia. Quando você acessa uma tabela criptografada, o DynamoDB descriptografa os dados da tabela de forma transparente. Você não precisa alterar seu código nem suas aplicações para usar ou gerenciar tabelas criptografadas. O DynamoDB continua proporcionando a mesma latência inferior a dez milissegundos que você espera, e todas as consultas do DynamoDB funcionam perfeitamente em dados criptografados.

Você pode especificar se o DynamoDB deve usar uma Chave pertencente à AWS (tipo de criptografia padrão), uma Chave gerenciada pela AWS ou uma chave gerenciada pelo cliente para criptografar os dados do usuário. A criptografia padrão usando chaves do KMS de propriedade da AWS está disponível sem nenhum custo adicional. Para criptografia do lado do cliente, você pode usar o SDK de criptografia de banco de dados da AWS.

O DynamoDB também segue vários padrões de conformidade, incluindo HIPAA, PCI DSS e RGPD, possibilitando que você atenda aos requisitos regulatórios.

Resiliência

Por padrão, o DynamoDB replica automaticamente seus dados em três zonas de disponibilidade para oferecer alta durabilidade e um SLA de 99,99% de disponibilidade. O DynamoDB também fornece recursos adicionais para ajudar você a alcançar seus objetivos de continuidade dos negócios e de recuperação de desastres.

O DynamoDB inclui os seguintes recursos para ajudar a oferecer suporte às suas necessidades de resiliência e backup de dados:

Atributos

- Tabelas globais
- Backups contínuos e recuperação para um ponto no tempo
- Backup e restauração sob demanda

Tabelas globais

As tabelas globais do DynamoDB possibilitam um SLA com disponibilidade de 99,999% e resiliência multirregional. Isso ajuda você a desenvolver aplicações resilientes e otimizá-las para o objetivo de tempo de recuperação (RTO) e o objetivo de ponto de recuperação (RPO) mais baixos. As tabelas globais também se integram ao AWS Fault Injection Service (AWS FIS) para realizar experimentos de injeção de falhas em suas workloads de tabelas globais. Por exemplo, pausar a replicação de tabelas globais em qualquer tabela-réplica.

Backups contínuos e recuperação para um ponto no tempo

Os backups contínuos fornecem detalhamento por segundo e a capacidade de iniciar uma recuperação para um ponto no tempo. Com a recuperação para um ponto no tempo, você pode restaurar uma tabela para qualquer ponto no tempo (com precisão de segundos) durante os últimos 35 dias. É possível definir o período de recuperação para qualquer valor entre 1 e 35 dias.

Os backups contínuos e a inicialização de uma restauração para um ponto no tempo não usam capacidade provisionada. Eles também não têm nenhum impacto no desempenho ou na disponibilidade das aplicações.

Backup e restauração sob demanda

O backup e a restauração sob demanda permitem que você crie backups completos de uma tabela para retenção a longo prazo e arquivamento para atender às necessidades de conformidade regulamentar. Os backups não afetam o desempenho da tabela e você pode fazer backup de tabelas de qualquer tamanho. Com a integração do AWS Backup, você pode usar o AWS Backup para agendar, copiar, marcar e gerenciar o ciclo de vida de backups sob demanda do DynamoDB automaticamente. Usando o AWS Backup, você pode copiar backups sob demanda entre contas e regiões e fazer a transição de backups mais antigos para armazenamento frio a fim de reduzir os custos.

Acessar o DynamoDB

Você pode trabalhar com o DynamoDB usando o AWS Management Console, a AWS Command Line Interface, o NoSQL Workbench para DynamoDB ou as APIs do DynamoDB.

Para obter mais informações, consulte Acessar o DynamoDB.

Preços do DynamoDB

O DynamoDB cobra pela leitura, gravação e armazenamento de dados nas tabelas, bem como todo recurso opcional que você optar por habilitar. O DynamoDB tem dois modos de capacidade com suas respectivas opções de cobrança para processar leituras e gravações nas tabelas: sob demanda e provisionado.

O DynamoDB também está incluído no nível sempre gratuito, oferecendo 25 GB de armazenamento. O nível gratuito também inclui 25 unidades de capacidade de gravação (WCUs) provisionadas e 25 unidades de capacidade de leitura (RCUs) provisionadas, o que é suficiente para lidar com 200 milhões de solicitações por mês.

Para obter mais informações, consulte a Definição de preço do Amazon DynamoDB.

Conceitos básicos do DynamoDB

Se você está usando o DynamoDB pela primeira vez, convém começar lendo os seguintes tópicos:

- Conceitos básicos do DynamoDB: orienta você no processo de configuração do DynamoDB, criação de tabelas de amostra e upload de dados. Este tópico também fornece informações sobre como realizar algumas operações básicas de banco de dados usando o AWS Management Console, a AWS CLI, o NoSQL Workbench e as APIs do DynamoDB.
- Componentes principais do DynamoDB: descreve os conceitos básicos do DynamoDB.
- Práticas recomendadas para desenhar e arquitetar com o DynamoDB: fornece recomendações sobre design NoSQL, Lente do Well-Architected para DynamoDB, design de tabelas e vários outros recursos do DynamoDB. Essas práticas recomendadas ajudam você a maximizar o desempenho e minimizar os custos de throughput ao trabalhar com o DynamoDB.

Também recomendamos que confira os tutoriais a seguir, que apresentam procedimentos completos para você se familiarizar com o DynamoDB. Você pode concluir estes tutoriais usando o recurso de nível sempre gratuito.

- Criar e consultar uma tabela NoSQL com o Amazon DynamoDB
- Criar uma aplicação usando um armazenamento de dados de chave-valor NoSQL

Para obter informações sobre recursos, ferramentas e estratégias de migração para o DynamoDB, consulte Migrar para o DynamoDB. Para ler os blogs e whitepapers mais recentes, consulte Recursos do Amazon DynamoDB.

Conceitos básicos do DynamoDB

Você aprenderá a se conectar, criar e gerenciar tabelas do DynamoDB nas seções a seguir.

Antes de começar, é importante conhecer os conceitos básicos do Amazon DynamoDB. Você pode obter uma rápida visão geral em O que é o Amazon DynamoDB? e uma análise mais aprofundada em Componentes principais do Amazon DynamoDB. Em seguida, continue com os Pré-requisitos.

> **Note**
>
> Ao se cadastrar na AWS, você poderá começar a usar o DynamoDB pelo nível gratuito da AWS. Se você ainda não tiver utilizado todos os benefícios do nível gratuito do Amazon DynamoDB, não haverá nenhum custo para concluir os exemplos desta seção. Caso contrário, você incorrerá nas taxas de uso padrão do DynamoDB desde o momento em que criar as tabelas até quando excluí-las.
> Se você não quiser criar uma conta de nível gratuito, poderá configurar o DynamoDB local (versão disponível para download) no computador. A versão para download permite desenvolver e testar aplicações localmente sem que seja necessário cadastrar uma conta AWS ou acessar o serviço da web do DynamoDB.

Tópicos

- Recursos do Amazon DynamoDB para usuários iniciantes
- Acessar o DynamoDB
- Pré-requisitos
- Configurar o DynamoDB
- Etapa 1: crie uma tabela no DynamoDB
- Etapa 2: grave dados em uma tabela do DynamoDB
- Etapa 3: leia os dados de uma tabela do DynamoDB
- Etapa 4: atualize os dados em uma tabela do DynamoDB
- Etapa 5: consulte os dados de uma tabela do DynamoDB
- Etapa 6: (Opcional) exclua uma tabela do DynamoDB para limpar os recursos
- Aprender mais sobre o DynamoDB

Recursos do Amazon DynamoDB para usuários iniciantes

Recomendamos que os usuários iniciantes comecem lendo as seguintes seções e as consultem conforme necessário.

- Destaques do serviço e preços: a página de detalhes do produto apresenta uma visão geral do DynamoDB, destaques do serviço e preços.
- Recursos do DynamoDB: a seção Recursos do DynamoDB tem vídeos, tutoriais e orientações prescritivas que apresentam o serviço, os conceitos de modelagem de dados e os principais recursos e capacidades.
- Conceitos básicos: a seção Conceitos básicos do DynamoDB inclui informações sobre como configurar o DynamoDB, criar tabelas de exemplo e fazer upload de dados. Também são fornecidas informações sobre como realizar operações básicas de banco de dados usando o AWS Management Console, a AWS CLI, o NoSQL Workbench e as APIs do DynamoDB.
- Curso básico do DynamoDB: um curso digital gratuito que ensina os fundamentos do DynamoDB, como design de tabelas, tipos de dados e operações básicas. Esse curso é ideal para desenvolvedores iniciantes no DynamoDB e em bancos de dados NoSQL.
- DynamoDB Nuggets: uma coleção de tutoriais em vídeo de curta duração e específicos que explicam os principais conceitos e recursos do DynamoDB. Esses vídeos de curta duração abordam vários tópicos em um formato fácil de entender, como modelagem de dados, estratégias de particionamento, padrões de acesso e práticas recomendadas.
- Repositório de exemplos de código do DynamoDB: oferece exemplos práticos de código do DynamoDB em várias linguagens de programação.
- Treinamento gratuito do DynamoDB: a AWS oferece cursos digitais gratuitos que abrangem conceitos, recursos e práticas recomendadas do DynamoDB. A série de cursos aprofundados do DynamoDB ajuda você a aprender a projetar modelos de dados eficientes, implementar chaves de partição eficazes e usar índices secundários com eficiência.
- NoSQL Workbench para DynamoDB: o NoSQL Workbench é uma ferramenta visual unificada que fornece recursos de modelagem de dados, visualização de dados e desenvolvimento de consultas. Ele ajuda você a projetar, criar, consultar e gerenciar tabelas do DynamoDB, o que o torna especialmente valioso para aprender os conceitos de modelagem de dados do DynamoDB e testar padrões de acesso.
- Padrões de design do DynamoDB: o tópico Práticas recomendadas e exemplos de modelagem de dados que demonstram práticas recomendadas para diferentes casos de uso. Inclui exemplos práticos e amostras de código para implementar soluções eficazes.

- Tutoriais práticos: tutoriais passo a passo no AWS Management Console que orientam você em tarefas comuns do DynamoDB, como criar tabelas e implementar consultas complexas.
- Migração para o DynamoDB: o tópico Migração para o DynamoDB apresenta uma visão geral do processo, das ferramentas e das estratégias para migrar um banco de dados para o DynamoDB.
- Lentes do AWS Well-Architected para DynamoDB: o tópico the section called “Lentes do Well-Architected para DynamoDB” apresenta práticas recomendadas de arquitetura para projetar e operar aplicações confiáveis, seguras, eficientes e econômicas usando o DynamoDB. Ele inclui princípios de design, padrões de arquitetura e orientação operacional alinhados aos seis pilares do Well-Architected Framework.

Práticas recomendadas adicionais do Amazon DynamoDB para usuários iniciantes

Depois de completar as seções anteriores, leia estas seções:

- Capacidade de throughput do DynamoDB

  Apresenta uma visão geral dos dois modos de throughput disponíveis para o DynamoDB e considerações sobre a seleção do modo de capacidade adequado para a aplicação. O modo sob demanda é a opção de throughput padrão e recomendada para a maioria das workloads do DynamoDB.

- Práticas recomendadas

  Identifique e resolva problemas para maximizar o desempenho e minimizar os custos ao trabalhar com o DynamoDB.

Recursos do AWS CLI

Se quiser usar a AWS Command Line Interface (AWS CLI), poderá usar estes documentos para ajudar você a começar:

- Documentação da AWS CLI

  Esta seção fornece informações sobre como baixar a AWS CLI, fazer a AWS CLI funcionar no seu sistema e fornecer suas credenciais da AWS.

- Documentação da AWS CLI para o DynamoDB

Esse documento separado abrange todos os aspectos da AWS CLI para comandos do DynamoDB, incluindo sintaxe e exemplos.

Recursos de programação

Você pode escrever programas de aplicação para usar a API do DynamoDB com uma variedade de linguagens de programação conhecidas. Aqui estão alguns recursos:

- Ferramentas para a Amazon Web Services

  A AWS fornece uma série de kits de desenvolvimento de software (SDKs) compatíveis com o DynamoDB. Você pode escrever código para o DynamoDB usando Java, .NET, PHP, Ruby e outras linguagens. Esses SDKs podem simplificar consideravelmente o desenvolvimento da aplicação, formatando solicitações para o DynamoDB, analisando respostas e fornecendo a lógica de repetição e o tratamento de erros.
- Referência de API do DynamoDB

  Se não quiser usar os SDKs da AWS, você poderá interagir com o DynamoDB diretamente usando a API do DynamoDB. Esse documento abrange todas as operações da API do DynamoDB, incluindo a sintaxe e exemplos. É possível encontrar dicas de solução de problemas e informações sobre como criar e autenticar solicitações e lidar com respostas nesta seção.

Acessar o DynamoDB

Você pode acessar o Amazon DynamoDB usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API do DynamoDB.

Tópicos

- Utilizar o console
- Uso do AWS CLI
- Uso da API
- Usar o NoSQL Workbench para DynamoDB
- Intervalos de endereços IP

Utilizar o console

Você pode acessar o AWS Management Console para Amazon DynamoDB em https://console.aws.amazon.com/dynamodb/home.

Aqui estão algumas das ações que você pode realizar no console do DynamoDB:

- Gerenciar tabelas: crie, atualize e exclua tabelas. A calculadora de capacidade pode ajudar a estimar os requisitos de capacidade.
- Interagir com dados: visualize, adicione, atualize e exclua itens das tabelas. Gerencie as configurações de vida útil (TTL).
- Monitorar e analisar: visualize painéis, monitore e configure alarmes e analise métricas e alertas das tabelas do DynamoDB.
- Otimizar e ampliar: gerencie índices secundários, fluxos, acionadores, capacidade reservada e outros recursos avançados para aprimorar o uso do DynamoDB.

O console do DynamoDB fornece uma interface abrangente para gerenciar os recursos do DynamoDB. Recomendamos que você acesse o console e interaja com ele para saber mais.

Uso do AWS CLI

Você pode usar a AWS Command Line Interface (AWS CLI) para controlar vários serviços da AWS via linha de comando e automatizá-los usando scripts. Use o AWS CLI para operações ad hoc, como a criação de uma tabela. Ela também pode ser usada para incorporar operações do Amazon DynamoDB em scripts utilitários.

Antes de usar a AWS CLI com o DynamoDB, você deve obter um ID de chave de acesso e uma chave de acesso secreta. Para ter mais informações, consulte Conceder acesso programático .

Para obter uma lista completa de todos os comandos disponíveis para o DynamoDB na AWS CLI, consulte a Referência de comandos da AWS CLI.

Download e configuração da AWS CLI

O AWS CLI está disponível em http://aws.amazon.com/cli. Ela é executada no Windows, macOS ou Linux. Depois de fazer download da AWS CLI, siga estas etapas para instalá-la e configurá-la:

1. Acesse o Guia do usuário do AWS Command Line Interface.
2. Siga as instruções de Instalação da AWS CLI e Configuração da AWS CLI.

Como usar a AWS CLI com o DynamoDB

O formato de linha de comando consiste em um nome de operação do DynamoDB seguido pelos parâmetros dessa operação. O AWS CLI suporta uma sintaxe abreviada para os valores de parâmetro, assim como JSON.

Por exemplo, o comando a seguir cria uma tabela chamada Music. A chave de partição é Artist, e a chave de classificação é SongTitle. (Para facilitar a leitura, comandos longos nesta seção são divididos em linhas separadas.)

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Os comandos a seguir adicionam novos itens à tabela. Esses exemplos usam uma combinação de sintaxe abreviada e JSON.

```
aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}}' \
    --return-consumed-capacity TOTAL

aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Acme Band"},
        "SongTitle": {"S": "Happy Day"},
        "AlbumTitle": {"S": "Songs About Life"} }' \
    --return-consumed-capacity TOTAL
```

Na linha de comando, pode ser difícil compor um JSON válido. No entanto, a AWS CLI pode ler arquivos JSON. Por exemplo, considere o seguinte trecho de código JSON, que é armazenado em um arquivo chamado key-conditions.json.

```
{
    "Artist": {
        "AttributeValueList": [
            {
                "S": "No One You Know"
            }
        ],
        "ComparisonOperator": "EQ"
    },
    "SongTitle": {
        "AttributeValueList": [
            {
                "S": "Call Me Today"
            }
        ],
        "ComparisonOperator": "EQ"
    }
}
```

Agora você pode emitir uma solicitação `Query` usando a AWS CLI. Neste exemplo, o conteúdo do arquivo key-conditions.json é usado para o parâmetro `--key-conditions`.

```
aws dynamodb query --table-name Music --key-conditions file://key-conditions.json
```

Usar a AWS CLI com o DynamoDB local

A AWS CLI também pode interagir com o DynamoDB local (versão para download) executado no computador. Para ativar isso, adicione o parâmetro a seguir em cada comando:

`--endpoint-url http://localhost:8000`

O exemplo a seguir usa a AWS CLI para listar as tabelas em um banco de dados local.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Se o DynamoDB estiver usando um número de porta diferente do padrão (8000), modifique o valor `--endpoint-url` de acordo com a necessidade.

> **Note**
>
> A AWS CLI não pode usar o DynamoDB local (versão para download) como um endpoint padrão. Portanto, você deve especificar `--endpoint-url` com cada comando.

Uso da API

Você pode usar o AWS Management Console e a AWS Command Line Interface para trabalhar de forma interativa com o Amazon DynamoDB. No entanto, para aproveitar ao máximo o DynamoDB, você pode escrever o código da aplicação usando os AWS SDKs.

Os SDKs da AWS oferecem amplo suporte para o DynamoDB em Java, JavaScript no navegador, .NET, Node.js, PHP, Python, Ruby, C++, Go, Android e iOS.

Antes de usar os AWS SDKs com o DynamoDB, você deve obter um ID de chave de acesso da AWS e uma chave de acesso secreta. Para ter mais informações, consulte Configurar o DynamoDB (serviço da Web) .

Para obter uma visão geral de alto nível da programação de aplicações do DynamoDB com AWS SDKs, consulte Programação com o DynamoDB e os AWS SDKs.

Usar o NoSQL Workbench para DynamoDB

Também é possível acessar o DynamoDB baixando e usando o NoSQL Workbench para DynamoDB.

O NoSQL Workbench para Amazon DynamoDB é uma aplicação GUI multiplataforma do lado do cliente para operações e desenvolvimento de bancos de dados modernos. Ele está disponível para Windows, macOS e Linux. O NoSQL Workbench é uma ferramenta de desenvolvimento visual que fornece recursos de modelagem de dados, visualização de dados e desenvolvimento de consultas para ajudar você a projetar, criar, consultar e gerenciar tabelas do DynamoDB. O NoSQL Workbench agora inclui o DynamoDB local como parte opcional do processo de instalação, o que facilita a modelagem de dados no DynamoDB local. Para saber mais sobre o DynamoDB local e seus requisitos, consulte Configurar o DynamoDB local (versão para download) .

> **Note**
>
> No momento, o NoSQL Workbench para DynamoDB não comporta logins da AWS configurados com a autenticação de dois fatores (2FA).

Modelagem de dados

Com o NoSQL Workbench para DynamoDB, você pode criar novos modelos de dados a partir de ou projetar modelos com base em modelos de dados existentes que satisfaçam os padrões de acesso a dados das suas aplicações. Pode também importar e exportar o modelo de dados designado no final do processo. Para ter mais informações, consulte Criar modelos de dados com o NoSQL Workbench.

Visualização de dados

O visualizador de modelo de dados oferece uma tela na qual você pode mapear consultas e visualizar os padrões de acesso (facetas) do aplicativo sem precisar escrever código. Cada faceta corresponde a um padrão de acesso diferente no DynamoDB. Você pode gerar dados de amostra automaticamente para uso no modelo de dados. Para ter mais informações, consulte Visualizar padrões de acesso a dados.

Criação de operação

O NoSQL Workbench oferece uma avançada interface gráfica do usuário para você desenvolver e testar consultas. Você pode usar o criador de operações para visualizar, explorar e consultar conjuntos de dados dinâmicos. Você também pode usar o criador de operações estruturadas para criar e executar operações de plano de dados. Ele oferece suporte para expressões de projeção e condição e permite que você gere código de exemplo em várias linguagens. Para ter mais informações, consulte Explorar conjuntos de dados e criar operações com o NoSQL Workbench.

Intervalos de endereços IP

A Amazon Web Services (AWS) publica seus intervalos de endereços IP atuais em formato JSON. Para visualizar os intervalos atuais, faça download do arquivo ip-ranges.json. Para obter mais informações, consulte Intervalos de endereços IP da AWS no Referência geral da AWS.

Para encontrar os intervalos de endereços IP que podem ser usados para acessar os índices e as tabelas do DynamoDB, procure no arquivo ip-ranges.json a seguinte string: `"service": "DYNAMODB"`.

> **Note**
>
> Os intervalos de endereços IP não se aplicam ao DynamoDB Streams ou ao DynamoDB Accelerator (DAX).

Pré-requisitos

Antes de iniciar o tutorial do Amazon DynamoDB, saiba como acessar o DynamoDB em Acessar o DynamoDB. Em seguida, configure o DynamoDB por meio do serviço da web ou da versão baixada localmente em Configurar o DynamoDB . Depois disso, continue em Etapa 1: crie uma tabela no DynamoDB.

> **Note**
>
> - Caso planeje interagir com o DynamoDB somente por meio do AWS Management Console, você não precisará de uma chave de acesso da AWS. Conclua as etapas em Cadastrar-se na AWS e prossiga em Etapa 1: crie uma tabela no DynamoDB.
> - Se você não quiser criar uma conta de nível gratuito, poderá configurar o DynamoDB local (versão disponível para download). Em seguida, vá para Etapa 1: crie uma tabela no DynamoDB.
> - Há diferenças ao trabalhar com comandos da CLI em terminais no Linux e no Windows. O guia a seguir apresenta comandos formatados para terminais do Linux (isso inclui macOS) e comandos formatados para CMD do Windows. Escolha o comando que melhor se adapta à aplicação de terminal que você está usando.

Configurar o DynamoDB

Além do serviço da web Amazon DynamoDB, a AWS fornece uma versão disponível para download do DynamoDB que pode ser executada localmente em um computador. A versão para download é útil para desenvolver e testar código. Ela permite gravar e testar aplicações localmente sem que seja necessário acessar o serviço da web DynamoDB.

Os tópicos desta seção descrevem como configurar o DynamoDB (versão para download) e o serviço da Web DynamoDB.

Tópicos

- Configurar o DynamoDB (serviço da Web)
- Configurar o DynamoDB local (versão para download)

Configurar o DynamoDB (serviço da Web)

Para usar o serviço da Web Amazon DynamoDB:

1. Cadastre-se no AWS.
2. Obtenha uma chave de acesso da AWS (usada para acessar o DynamoDB por programação).

> **Note**
>
> Caso pretenda interagir com o DynamoDB somente por meio do AWS Management Console, você não precisará de uma chave de acesso da AWS e poderá avançar para Utilizar o console.

3. Configure suas credenciais (usadas para acessar o DynamoDB por programação).

Como se cadastrar na AWS

Para usar o serviço DynamoDB, é necessário ter uma conta da AWS. Se você ainda não tiver uma conta, será solicitado a criar uma ao se cadastrar. Você não será cobrado por nenhum dos serviços da AWS nos quais se cadastrar, a menos que os utilize.

Para cadastrar-se na AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica ou uma mensagem de texto e inserir um código de verificação pelo teclado do telefone.

   Quando você se inscreve para uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática

recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

Conceder acesso programático

Para poder acessar o DynamoDB de maneira programática ou por meio da AWS Command Line Interface (AWS CLI), você deve ter acesso programático. Você não precisa de acesso programático se planeja usar apenas o console do DynamoDB.

Os usuários precisam de acesso programático se quiserem interagir com a AWS de fora do AWS Management Console. A forma de conceder acesso programático depende do tipo de usuário que está acessando a AWS.

Para conceder acesso programático aos usuários, selecione uma das seguintes opções:

Qual usuário precisa de acesso programático?	Para	Por
Identidade da força de trabalho (Usuários gerenciados no Centro de Identidade do IAM)	Use credenciais temporárias para assinar solicitações programáticas para a AWS CLI, os SDKs da AWS ou as APIs da AWS.	Siga as instruções da interface que deseja utilizar. • Para a AWS CLI, consulte Configuração da AWS CLI para usar o AWS IAM Identity Center no Guia do usuário da AWS Command Line Interface. • Para os SDKs da AWS, ferramentas e APIs da AWS, consulte Autenticação do Centro de Identidade do IAM no Guia de referência de ferramentas e SDKs da AWS.
IAM	Use credenciais temporárias para assinar solicitações programáticas para a AWS	Siga as instruções em Usar credenciais temporárias com

Qual usuário precisa de acesso programático?	Para	Por
	CLI, os SDKs da AWS ou as APIs da AWS.	recursos da AWS no Guia do usuário do IAM.
IAM	(Não recomendado) Use credenciais de longo prazo para assinar solicitaç ões programáticas para a AWS CLI, os SDKs da AWS ou as APIs da AWS.	Siga as instruções da interface que deseja utilizar. • Para a AWS CLI, consulte Autenticação usando as credenciais de usuário do IAM no Guia do usuário da AWS Command Line Interface. • Para as ferramentas e SDKs da AWS, consulte Autenticação usando as credenciais de longo prazo no Guia de referência de ferramentas e SDKs da AWS. • Para as APIs da AWS, consulte Gerenciamento de chaves de acesso de usuários do IAM no Guia do usuário do IAM.

Configurar credenciais

Para poder acessar o DynamoDB por programação ou via AWS CLI, você deve configurar suas credenciais para habilitar a autorização para suas aplicações.

Há várias maneiras de fazer isso. Por exemplo, é possível criar manualmente o arquivo de credenciais para armazenar o ID da chave de acesso e a chave de acesso secreta. Também é possível usar o comando `aws configure` da AWS CLI para criar o arquivo automaticamente. Como alternativa, você pode usar variáveis de ambiente. Para obter mais informações sobre

como configurar suas credenciais, consulte o guia do desenvolvedor do AWS SDK específico de programação.

Para instalar e configurar a AWS CLI, consulte Uso do AWS CLI.

Integração com outros serviços do DynamoDB

Você pode integrar o DynamoDB com muitos outros serviços da AWS. Para obter mais informações, consulte:

- Usar o DynamoDB com outros serviços da AWS
- AWS CloudFormation para DynamoDB
- Como usar a AWS Backup com o DynamoDB
- AWS Identity and Access Management (IAM) e DynamoDB
- Usar o AWS Lambda com o Amazon DynamoDB

Configurar o DynamoDB local (versão para download)

Com a versão para download do Amazon DynamoDB, você pode desenvolver e testar aplicações sem acessar o serviço da Web DynamoDB. Em vez disso, o banco de dados é autossuficiente em seu computador. Quando estiver pronto para implantar a aplicação no ambiente de produção, remova o endpoint local no código e ele apontará para o serviço da Web DynamoDB.

Essa versão local ajuda você a economizar em throughput, armazenamento de dados e taxas de transferência de dados. Além disso, você não precisa ter uma conexão com a Internet ao desenvolver a aplicação.

O DynamoDB local está disponível como download (requer JRE), como dependência do Apache Maven ou como imagem do Docker.

Se você preferir usar o serviço da Web Amazon DynamoDB, consulte Configurar o DynamoDB (serviço da Web) .

Tópicos

- Implantar o DynamoDB localmente no computador
- Observações sobre o uso do DynamoDB local
- Histórico de versões do DynamoDB local
- Telemetria no DynamoDB local

Implantar o DynamoDB localmente no computador

> **Note**
>
> - O DynamoDB local está disponível em três versões: v3.x (atual), v2.x (legada) e v1.x (obsoleta).
> - O DynamoDB v3.x é recomendado para seu uso local em testes e desenvolvimento.
> - A migração do DynamoDB local V2.x para a V3.x requer a atualização das instruções de importação de `com.amazonaws.services.dynamodbv2` para `software.amazon.dynamodb` e a atualização das dependências do Maven para usuários do Maven.
> - Se você estiver migrando um aplicativo que usa o SDK para Java v1.x para o SDK para Java 2.x, siga as etapas para AWS SDK para Java 2.x.

Baixar o DynamoDB local

Siga estas etapas para configurar e executar o DynamoDB em seu computador.

Para configurar o DynamoDB em seu computador

1. Baixe o DynamoDB local gratuitamente de um dos locais a seguir.

Links para fazer download	Somas de verificação
.tar.gz | .zip	.tar.gz.sha256 | .zip.sha256

> **Important**
>
> Para executar o DynamoDB v2.6.0 ou posterior no computador, é necessário ter o Ambiente de Execução Java (JRE) versão 17.x ou mais recente instalado. A aplicação não é executada em versões mais antigas do JRE.

2. Depois de fazer download do arquivo, extraia o conteúdo e copie o diretório extraído para um local de sua escolha.

3. Para iniciar o DynamoDB em seu computador, abra uma janela de prompt de comando, vá para o diretório onde você extraiu o arquivo `DynamoDBLocal.jar` e insira o comando seguir.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

> **Note**
>
> Se estiver usando o Windows PowerShell, inclua o nome do parâmetro ou todo o nome e valor da seguinte forma:
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
> O DynamoDB processa as solicitações de entrada até que você o interrompa. Para interromper o DynamoDB, pressione Ctrl+C no prompt de comando.
> O DynamoDB usa a porta 8000 por padrão. Se a porta 8000 estiver indisponível, este comando lançará uma exceção. Para obter uma lista completa das opções de tempo de execução do DynamoDB, incluindo `-port`, execute este comando.
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

4. Para poder acessar o DynamoDB por programação ou via AWS Command Line Interface (AWS CLI), você deve configurar suas credenciais para habilitar a autorização para suas aplicações. O DynamoDB para download requer que todas as credenciais funcionem, conforme mostrado no exemplo a seguir.

```
AWS Access Key ID: "fakeMyKeyId"
AWS Secret Access Key: "fakeSecretAccessKey"
Default Region Name: "fakeRegion"
```

   Você pode usar o comando `aws configure` da AWS CLI para configurar credenciais. Para obter mais informações, consulte Uso do AWS CLI.

5. Comece a escrever aplicações. Para acessar o DynamoDB em execução local com a AWS CLI, use o parâmetro `--endpoint-url` . Por exemplo, use o comando a seguir para listar as tabelas do DynamoDB.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Executar o DynamoDB local como imagem do Docker

A versão para download do Amazon DynamoDB está disponível como uma imagem do Docker. Para obter mais informações, consulte dynamodb-local. Para ver a versão do DynamoDB local atual, execute o seguinte comando:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -version
```

Para obter um exemplo de uso do DynamoDB local como parte de um aplicativo REST criado com a AWS Serverless Application Model (AWS SAM), consulte Aplicação SAM do DynamoDB para gerenciamento de pedidos. Esta aplicação de exemplo demonstra como usar o DynamoDB local para testes.

Se você quiser executar uma aplicação de vários contêineres que também use o contêiner local do DynamoDB, use o Docker Compose para definir e executar todos os serviços em sua aplicação, incluindo o DynamoDB local.

Para instalar e executar o DynamoDB local com o Docker Compose:

1. Baixe e instale a área de trabalho do Docker.
2. Copie o código a seguir em um arquivo e salve-o como `docker-compose.yml`.

   ```
   services:
    dynamodb-local:
      command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
      image: "amazon/dynamodb-local:latest"
      container_name: dynamodb-local
      ports:
        - "8000:8000"
      volumes:
        - "./docker/dynamodb:/home/dynamodblocal/data"
      working_dir: /home/dynamodblocal
   ```

   Se desejar que sua aplicação e o DynamoDB local estejam em contêineres separados, use o arquivo yaml a seguir.

   ```
   version: '3.8'
   services:
    dynamodb-local:
   ```

```
    command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
    image: "amazon/dynamodb-local:latest"
    container_name: dynamodb-local
    ports:
      - "8000:8000"
    volumes:
      - "./docker/dynamodb:/home/dynamodblocal/data"
    working_dir: /home/dynamodblocal
  app-node:
    depends_on:
      - dynamodb-local
    image: amazon/aws-cli
    container_name: app-node
    ports:
     - "8080:8080"
    environment:
      AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
      AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
    command:
      dynamodb describe-limits --endpoint-url http://dynamodb-local:8000 --region
  us-west-2
```

Este script docker-compose.yml cria um contêiner `app-node` e um contêiner `dynamodb-local`. O script executa um comando no contêiner `app-node` que usa a AWS CLI para se conectar ao contêiner `dynamodb-local` e descreve os limites da conta e da tabela.

Para usar com sua própria imagem de aplicação, substitua o valor `image` no exemplo abaixo pelo valor da sua aplicação.

```
version: '3.8'
services:
  dynamodb-local:
    command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
    image: "amazon/dynamodb-local:latest"
    container_name: dynamodb-local
    ports:
      - "8000:8000"
    volumes:
      - "./docker/dynamodb:/home/dynamodblocal/data"
    working_dir: /home/dynamodblocal
  app-node:
    image: location-of-your-dynamodb-demo-app:latest
```

```
  container_name: app-node
  ports:
    - "8080:8080"
  depends_on:
    - "dynamodb-local"
  links:
    - "dynamodb-local"
  environment:
    AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
    AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
    REGION: 'eu-west-1'
```

> **Note**
>
> Os scripts YAML exigem que você especifique uma chave de acesso da AWS e uma chave secreta da AWS, mas elas não precisam ser chaves válidas da AWS para acessar o DynamoDB local.

3. Execute o seguinte comando a linha de comando:

```
docker-compose up
```

Executar o DynamoDB local como dependência do Apache Maven

> **Note**
>
> Se você estiver migrando um aplicativo que usa o SDK para Java v1.x para o SDK para Java 2.x, siga as etapas para AWS SDK para Java 2.x.

Siga estas etapas para usar o Amazon DynamoDB em sua aplicação como uma dependência.

Para implantar o DynamoDB como um repositório do Apache Maven

1. Faça download do Apache Maven e instale-o. Para obter mais informações, consulte Download do Apache Maven e Instalação do Apache Maven.
2. Adicione o repositório Maven do DynamoDB ao arquivo Project Object Model (POM) da sua aplicação.

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>software.amazon.dynamodb</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>3.0.0</version>
   </dependency>
</dependencies>
```

Exemplo de modelo para uso com o Spring Boot 3 e/ou Spring Framework 6:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>SpringMavenDynamoDB</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
   <spring-boot.version>3.0.1</spring-boot.version>
   <maven.compiler.source>17</maven.compiler.source>
   <maven.compiler.target>17</maven.compiler.target>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

   <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>3.0.1</version>
   </parent>

<dependencies>
   <dependency>
       <groupId>software.amazon.dynamodb</groupId>
       <artifactId>DynamoDBLocal</artifactId>
       <version>3.0.0</version>
   </dependency>
   <!-- Spring Boot -->
```

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Other Spring dependencies -->
    <!-- Replace the version numbers with the desired version -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>6.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>6.0.0</version>
    </dependency>
    <!-- Add other Spring dependencies as needed -->
    <!-- Add any other dependencies your project requires -->
</dependencies>
</project>
```

> **Note**
>
> Também é possível usar o URL do repositório central do Maven.

Executar o DynamoDB localmente no AWS CloudShell

O AWS CloudShell é um shell pré-autenticado baseado em navegador que você pode iniciar diretamente do AWS Management Console. Você pode navegar até o AWS CloudShell por meio do AWS Management Console de maneiras diferentes. Para obter mais informações, consulte Conceitos básicos do AWS CloudShell.

Siga estas etapas para executar o DynamoDB localmente no AWS CloudShell em qualquer lugar do AWS Management Console.

Como executar o DynamoDB localmente no AWS CloudShell por meio do AWS Management Console

1. Inicie o AWS CloudShell por meio da interface do console, escolhe uma Região da AWS disponível e alterne para o shell de sua preferência; por exemplo, Bash, PowerShell ou shell Z.
2. Para escolher uma Região da AWS para trabalhar, acesse o menu Selecionar uma região e escolha uma Região da AWS compatível. (As regiões disponíveis estão em destaque.)
3. Por meio do AWS Management Console, é possível iniciar o AWS CloudShell escolhendo uma das opções a seguir:
   a. Na barra de navegação, escolha o ícone AWS CloudShell.
   b. Na caixa Pesquisar, insira a palavra CloudShell e escolha CloudShell.
   c. No widget Visitado recentemente, escolha CloudShell.
   d. Na barra de ferramentas do console, escolha CloudShell.
4. Para executar o DynamoDB localmente no AWS CloudShell, você pode usar o alias `dynamodb-local`. É possível especificar outras opções de linha de comando para alterar as configurações locais do DynamoDB. Consulte the section called “Observações de uso” para obter as opções disponíveis.

   > **Note**
   >
   > Para executar o DynamoDB localmente em segundo plano, execute o DynamoDB localmente no AWS CloudShell usando `dynamodb-local &`.

5. Para acessar o DynamoDB em execução local no AWS CloudShell, use o parâmetro `--endpoint-url`. Por exemplo, use o seguinte comando para listar tabelas do DynamoDB:

   ```
   aws dynamodb list-tables --endpoint-url http://localhost:8000
   ```

Para ver um exemplo de projeto de amostra que mostra várias abordagens para configurar e usar o DynamoDB local, inclusive baixar arquivos JAR, executá-lo como uma imagem do Docker e usá-lo como uma dependência do Maven, consulte DynamoDB Local Sample Java Project.

Observações sobre o uso do DynamoDB local

Com exceção do endpoint, as aplicações que são executadas com a versão para download do Amazon DynamoDB também devem funcionar com o serviço da Web do DynamoDB. No entanto, ao usar o DynamoDB localmente, você deve estar ciente do seguinte:

- Se você usar a opção `-sharedDb`, o DynamoDB criará um único arquivo de banco de dados chamado shared-local-instance.db. Todos os programas que se conectam ao DynamoDB acessam este arquivo. Se você excluir o arquivo, perderá todos os dados armazenados nele.
- Se você omitir `-sharedDb`, o arquivo do banco de dados se chamará myaccesskeyid_region.db, com o ID de chave de acesso AWS e região da AWS da forma como são exibidos na configuração da aplicação. Se você excluir o arquivo, perderá todos os dados armazenados nele.
- Se você usar a opção `-inMemory`, o DynamoDB não gravará nenhum arquivo de banco de dados. Em vez disso, todos os dados são gravados na memória, e eles não são salvos quando o DynamoDB é terminado.
- Se você usar `-inMemory`, a opção `-sharedDb` também será necessária.
- Se você usar a opção `-optimizeDbBeforeStartup`, também deverá especificar o parâmetro `-dbPath` para que o DynamoDB possa encontrar seu arquivo de banco de dados.
- Os AWS SDKs para DynamoDB exigem que a configuração da sua aplicação especifique um valor de chave de acesso e um valor de região da AWS. A menos que você esteja usando a opção `-sharedDb` ou `-inMemory`, o DynamoDB usará esses valores para nomear o arquivo de banco de dados local. Esses não precisam ser valores válidos da AWS para executar localmente. No entanto, pode ser conveniente usar valores válidos, de modo que você possa executar o código na nuvem mais tarde alterando o endpoint que você está usando.
- O DynamoDB local sempre retorna nulo para `billingModeSummary.`
- O DynamoDB local `AWS_ACCESS_KEY_ID` pode conter somente letras (A-Z, a-z) e números (0-9).
- O DynamoDB local não comporta recuperação para um ponto no tempo (PITR).

Tópicos

- Opções de linha de comando
- Definir o endpoint local

- Diferenças entre o DynamoDB para download e o serviço da Web DynamoDB

Opções de linha de comando

Você pode usar uma das seguintes opções de linha de comando com a versão para download do DynamoDB:

- `-cors value`: habilita o suporte ao CORS (compartilhamento de recursos entre origens) para JavaScript. Você deve fornecer uma lista de "permissões" de domínios específicos separados por vírgulas. A configuração padrão de `-cors` é um asterisco (*), que permite acesso público.
- `-dbPath value`: o diretório em que o DynamoDB grava seu arquivo de banco de dados. Se você não especificar essa opção, o arquivo será gravado no diretório atual. Não é possível especificar `-dbPath` e `-inMemory` ao mesmo tempo.
- `-delayTransientStatuses`: faz com que o DynamoDB introduza atrasos em determinadas operações. O DynamoDB (versão para download) pode executar algumas tarefas quase instantaneamente, como criar/atualizar/excluir operações em tabelas e índices. No entanto, o serviço DynamoDB requer mais tempo para essas tarefas. A definição desse parâmetro ajuda o DynamoDB em execução em seu computador a simular o comportamento do serviço da Web DynamoDB de maneira mais precisa. (No momento, esse parâmetro apresenta atrasos apenas para índices secundários globais que estão no status CREATING ou DELETING.)
- `-help`: imprime um resumo de uso e opções.
- `-inMemory`: o DynamoDB é executado na memória, em vez de usar um arquivo de banco de dados. Quando você interrompe o DynamoDB, nenhum dos dados é salvo. Não é possível especificar `-dbPath` e `-inMemory` ao mesmo tempo.
- `-optimizeDbBeforeStartup`: otimiza as tabelas de banco de dados subjacentes antes de iniciar o DynamoDB no seu computador. Você também deve especificar `-dbPath` ao usar este parâmetro.
- `-port value`: o número da porta que o DynamoDB usa para se comunicar com sua aplicação. Se você não especificar essa opção, a porta padrão será `8000`.

> **Note**
>
> O DynamoDB usa a porta 8000 por padrão. Se a porta 8000 estiver indisponível, este comando lançará uma exceção. Você pode usar a opção `-port` para especificar um número de porta diferente. Para obter uma lista completa das opções de tempo de execução do DynamoDB, incluindo `-port`, digite este comando:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar
-help
```

- `-sharedDb`: se você especificar `-sharedDb`, o DynamoDB usará um único arquivo de banco de dados, em vez de usar arquivos separados para cada credencial e região.
- `-disableTelemetry`: quando especificada, o DynamoDB local não enviará nenhuma telemetria.
- `-version`: imprime a versão do DynamoDB local.

Definir o endpoint local

Por padrão, os AWS SDKs e as ferramentas usam endpoints para o serviço da Web Amazon DynamoDB. Para usar os SDKs e as ferramentas com as versões para download do DynamoDB, especifique o endpoint local:

`http://localhost:8000`

AWS Command Line Interface

Você pode usar a AWS Command Line Interface (AWS CLI) para interagir com o DynamoDB para download.

Para acessar o DynamoDB em execução localmente, use o parâmetro `--endpoint-url`. Veja a seguir um exemplo de uso da AWS CLI para listar as tabelas do DynamoDB em seu computador.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

> **Note**
>
> A AWS CLI não pode usar a versão para download do DynamoDB como um endpoint padrão. Portanto, você deve especificar `--endpoint-url` com cada comando na AWS CLI.

SDKs da AWS

A maneira como você especifica um endpoint depende da linguagem de programação e do AWS SDK que você está usando. As seções a seguir descrevem como fazer isso:

- Java: configurar a região e o endpoint da AWS (DynamoDB local oferece suporte ao AWS SDK para Java V1 e V2)

- CodeSamples.Java.RegionAndEndpoint .NET: configurar a região e o endpoint da AWS

Diferenças entre o DynamoDB para download e o serviço da Web DynamoDB

A versão para download do DynamoDB deve ser usada somente para testes e desenvolvimento. Em comparação, o serviço da Web DynamoDB é um serviço gerenciado com recursos de escalabilidade, disponibilidade e durabilidade que o tornam ideal para uso no ambiente de produção.

A versão para download do DynamoDB difere do serviço da Web das seguintes formas:

- Regiões da AWS e Contas da AWS distintas não são aceitas em nível de cliente.
- As configurações de throughput provisionado são ignoradas no DynamoDB para download, embora a operação `CreateTable` as exija. Em `CreateTable`, você pode especificar quaisquer números desejados para o throughput de leitura e gravação provisionado, mesmo que esses números não sejam usados. Você pode chamar `UpdateTable` quantas vezes quiser por dia. No entanto, todas as alterações nos valores do throughput provisionado são ignoradas.
- As operações `Scan` são executadas sequencialmente. Verificações paralelas não são aceitas. Os parâmetros `Segment` e `TotalSegments` da operação `Scan` são ignorados.
- A velocidade das operações de leitura e gravação nos dados da tabela é limitada apenas pela velocidade do computador. As operações `CreateTable`, `UpdateTable` e `DeleteTable` ocorrem imediatamente, e o estado da tabela é sempre ACTIVE. As operações `UpdateTable` que alteram apenas as configurações de throughput provisionado em tabelas ou índices secundários globais ocorrem imediatamente. Se uma operação `UpdateTable` criar ou excluir qualquer índice secundário global, esses índices farão uma transição pelos estados normais (como CREATING e DELETING, respectivamente) antes de passarem para um estado ACTIVE. A tabela permanece no estado ACTIVE durante esse tempo.
- As operações de leitura são eventualmente consistentes. No entanto, devido à velocidade do DynamoDB em execução no local no computador, a maioria das leituras parecerão ser fortemente consistentes.
- As métricas de coleção de itens e os tamanhos de coleção de itens não são controlados. Nas respostas de operação, nulos são retornados em vez de métricas de coleção de itens.
- No DynamoDB, há um limite de 1 MB em dados retornados por conjunto de resultados. Tanto o serviço da Web DynamoDB quanto a versão para download impõem esse limite. No entanto, ao consultar um índice, o serviço DynamoDB calcula apenas o tamanho da chave e dos atributos projetados. Por outro lado, a versão para download do DynamoDB calcula o tamanho do item inteiro.

- Se você estiver usando o DynamoDB Streams, a taxa na qual fragmentos são criados poderá ser diferente. No serviço da Web DynamoDB, o comportamento de criação de fragmentos é parcialmente influenciado pela atividade da partição da tabela. Quando o DynamoDB é executado localmente, não há particionamento de tabelas. Em qualquer um dos casos, os fragmentos são efêmeros, portanto, seu aplicativo não deve ser dependente do comportamento do fragmento.
- `TransactionConflictExceptions` não são lançadas pelo DynamoDB para download para APIs transacionais. Recomendamos o uso de um framework de imitação Java para simular `TransactionConflictExceptions` no manipulador do DynamoDB para testar como o aplicativo responde a transações em conflito.
- No serviço web do DynamoDB, quer seja acessado pelo console ou pela AWS CLI, os nomes das tabelas diferenciam maiúsculas de minúsculas. Uma tabela chamada `Authors` e outra chamada `authors` podem existir como tabelas separadas. Na versão que pode ser obtida por download, os nomes de tabelas não diferenciam maiúsculas de minúsculas, e a criação dessas duas tabelas resulta em erro.
- A marcação não é aceita na versão para download do DynamoDB.
- A versão para download do DynamoDB ignora o parâmetro Limit em ExecuteStatement.

Histórico de versões do DynamoDB local

A tabela a seguir descreve as alterações importantes em cada versão do DynamoDB local.

Versão	Alteração	Descrição	Data
3.0.0	Migrar de SDK da AWS Java V1 para V2	• Migrar de SDK da AWS Java V1 para V2 • Estrutura de pacotes atualizada de `com.amazonaws.services.dynamodbv2` para `software.amazon.dy`	17 de julho de 2025

Versão	Alteração	Descrição	Data
		`namodb.se rvices` • Dependências removidas do SDK da AWS Java V1	
2.6.0	Suporte ao ARN da tabela como nome de tabela nas APIs do DynamoDB Correção de performance e atualizações de segurança	• Adição de suporte para usar o ARN da tabela como nome da tabela em várias APIs do DynamoDB • Correção do bug `CreateStr eamTable` em máquinas de alta performance, como o Mac M3 • Atualização de dependências para corrigir problemas de vulnerabilidade (CVE-2022-49043, CVE-2024-56732, CVE-2020-29582, CVE-2025-21502, CVE-2024-50602, CVE-2025-24970, CVE-2025-25193)	13 de março de 2025

Versão	Alteração	Descrição	Data
2.5.4	Atualização para as dependências do Jetty	• Atualização do Jetty 12.0.8 para o Jetty 12.0.14 (resolve CVE-2024-6763, CVE-2024-8184, CVE-2024-47535) Mitigação para (CVE-2024-21634)	12 de dezembro de 2024
2.5.3	Atualizar as dependências do Jackson para 2.17.x no Log4j Core (Resolve CVE-2022-1471)	• Atualizar as dependências do Jackson para 2.17.x no Log4j Core (Resolve CVE-2022-1471) para resolver uma vulnerabilidade crítica de segurança na biblioteca SnakeYAML, que é uma dependência transitiva	6 de novembro de 2024
2.5.2	Correção de bug para o fluxo de trabalho de atualização de tabela	• Correção de bug no fluxo de trabalho quando a atualização de tabela tenta atualizar a tabela com o modo de faturamento sob demanda para provisionado com GSI	20 de junho de 2024

Versão	Alteração	Descrição	Data
2.5.1	Patch para bugs introduzidos no recurso `OndemandThroughPut`	• Corrigidos alguns bugs relacionados a `OndemandThroughPut`	5 de junho de 2024
2.5.0	Suporte a throughput máximo configurável para tabelas sob demanda, `ReturnValuesOnConditionCheckFailure`, `BatchExecuteStatement` e `ExecuteTransaction Request`	• Adição de telemetria ao modo incorporado • Correção da tradução de SDKv2 para `ConditionalCheckException`	28 de maio de 2024
2.4.0	Suporte para `ReturnValuesOnConditionCheckFailure` : modo incorporado	• Correção de modo incorporado para `TrimmedDataAccessException` para operação em vários fluxos • Corrigir a tradução de exceções para o SDKv2 no modo incorporado	17 de abril de 2024

Versão	Alteração	Descrição	Data
2.3.0	Atualização do Jetty e do JDK	• Atualizar para o Jetty 12.0.2 • Atualizar para o JDK 17 • Atualizar o ANTLR4 para 4.10.1	14 de março de 2024
2.2.0	Foi adicionada compatibilidade com a proteção contra exclusão de tabelas e o parâmetro `ReturnValuesOnConditionCheckFailure`	• Foi adicionada compatibilidade com a proteção contra exclusão de tabelas • Foi adicionada compatibilidade com ReturnValuesOnConditionCheckFailure • Foi adicionada compatibilidade com o sinalizador -version.	14 de dezembro de 2023

Versão	Alteração	Descrição	Data
2.1.0	Suporte para bibliotecas nativas SQLite para projetos Maven e adição de telemetria	• Adição de telemetria ao DynamoDB local • Cópia dinâmica de bibliotecas nativas SQLite para projetos Maven • Remoção da biblioteca io.github.ganadist.sqlite4java da dependência do Maven • Atualização do GoogleGuava para 32.1.1-jre	23 de outubro de 2023
2.0.0	Migração do namespace javax para jakarta e suporte ao JDK11	• Migração do namespace javax para jakarta e suporte ao JDK11 • Correção para lidar com acesso inválido e chave secreta durante a inicialização do servidor • Correção de vulnerabilidades identificadas no Maven por meio da atualização de dependências	5 de julho de 2023

Versão	Alteração	Descrição	Data
1.25.1	Atualizar as dependências do Jackson para 2.17.x no Log4j Core (Resolve CVE-2022-1471)	Atualizar as dependências do Jackson para 2.17.x no Log4j Core (Resolve CVE-2022-1471) para resolver uma vulnerabilidade crítica de segurança na biblioteca SnakeYAML, que é uma dependência transitiva	6 de novembro de 2024
1.25.0	Foi adicionada compatibilidade com a proteção contra exclusão de tabelas e o parâmetro `ReturnValuesOnConditionCheckFailure`	• Foi adicionada compatibilidade com a proteção contra exclusão de tabelas • Foi adicionada compatibilidade com ReturnValuesOnConditionCheckFailure • Foi adicionada compatibilidade com o sinalizador -version.	18 de dezembro de 2023

Versão	Alteração	Descrição	Data
1.24.0	Suporte para bibliotecas nativas SQLite para projetos Maven e adição de telemetria	• Adição de telemetria ao DynamoDB local • Cópia dinâmica de bibliotecas nativas SQLite para projetos Maven • Remoção da biblioteca io.github.ganadist.sqlite4java da dependência do Maven • Atualização do GoogleGuava para 32.1.1-jre	23 de outubro de 2023
1.23.0	Tratamento do acesso inválido e da chave secreta durante a inicialização do servidor	• Correção para lidar com acesso inválido e chave secreta durante a inicialização do servidor • Correção de vulnerabilidades identificadas no Maven por meio da atualização de dependências	28 de junho de 2023

Versão	Alteração	Descrição	Data
1.22.0	Compatibilidade com a operação de limite para PartiQL	• Otimização da cláusula IN para PartiQL • Compatibilidade com a operação de limite • Suporte a M1 para projetos do Maven	8 de junho de 2023
1.21.0	Suporte para 100 ações por transação	• Aumento das ações por transação de 25 para 100 • Atualização da imagem do docker Open JDK para 11 • Correção da paridade da exceção lançada quando há itens duplicados em BatchExecuteStatement	26 de janeiro de 2023
1.20.0	Adição de suporte para Mac M1	• Adição de suporte para Mac M1 • Atualização da dependência do Jetty para 9.4.48.v20220622	12 de setembro de 2022
1.19.0	Atualização do PartiQL Parser	Atualização do PartiQL Parser e de outras bibliotecas relacionadas	27 de julho de 2022

Versão	Alteração	Descrição	Data
1.18.0	Atualização do Log4j-core e Jackson-core	Atualização do Log4j-core para 2.17.1 e do Jackson-core 2.10.x para 2.12.0	10 de janeiro de 2022
1.17.2	Atualização do log4j-core	Atualização da dependência do log4j-core para a versão 2.16	16 de janeiro de 2021
1.17.1	Atualização do log4j-core	Atualização da dependência do log4j-core para corrigir a exploração de dia zero e evitar a execução remota de código (Log4Shel)	10 de janeiro de 2023
1.17.0	Descontinuação do Javascript Web Shell	• Atualização da dependência do AWS SDK para o AWS SDK para Java 1.12.x • Descontinuação do Javascript Web Shell	8 de janeiro de 2021

Telemetria no DynamoDB local

Na AWS, para desenvolver e lançar serviços usamos o que aprendemos nas interações com os clientes e usamos o feedback deles para iterar nossos produtos. Telemetria são informações adicionais que nos ajudam a entender melhor as necessidades dos clientes, diagnosticar problemas e fornecer recursos que aprimoram a experiência do cliente.

O DynamoDB local coleta telemetria, como métricas genéricas de uso, informações sobre sistemas e ambientes e erros. Para obter detalhes sobre os tipos de telemetria coletados, consulte Tipos de informação coletados.

O DynamoDB local não coleta informações pessoais, como nomes de usuário ou endereços de e-mail. Ele também não extrai informações sigilosas em nível de projeto.

Como cliente, você controla se a telemetria está ativada e pode alterar as configurações a qualquer momento. Se a telemetria permanecer ativada, o DynamoDB local enviará dados de telemetria em segundo plano sem exigir nenhuma interação adicional com o cliente.

Desative a telemetria usando as opções da linha de comando

Você pode desativar a telemetria por meio das opções de linha de comando ao iniciar o DynamoDB local usando a opção `-disableTelemetry`. Para ter mais informações, consulte Opções de linha de comando .

Desative a telemetria de uma única sessão

Nos sistemas operacionais macOS e Linux, você pode desativar a telemetria de uma única sessão. Para desativar a telemetria da sessão atual, execute o comando a seguir a fim de definir a variável de ambiente `DDB_LOCAL_TELEMETRY` como `false`. Repita o comando para cada novo terminal ou sessão.

```
export DDB_LOCAL_TELEMETRY=0
```

Desative a telemetria do seu perfil em todas as sessões

Execute os comandos a seguir para desativar a telemetria de todas as sessões quando você estiver executando o DynamoDB local no sistema operacional.

Para desabilitar a telemetria no Linux

1. Execute:

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. Execute:

   ```
   source ~/.profile
   ```

Para desabilitar a telemetria no macOS

1. Execute:

```
echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
```

2. Execute:

```
source ~/.profile
```

Para desabilitar a telemetria no Windows

1. Execute:

```
setx DDB_LOCAL_TELEMETRY 0
```

2. Execute:

```
refreshenv
```

Desativar a telemetria usando o DynamoDB local incorporado em projetos Maven

Você pode desativar a telemetria usando o DynamoDB local incorporado em projetos Maven.

```
boolean disableTelemetry = true;
// AWS SDK v1
 AmazonDynamoDB amazonDynamoDB =
 DynamoDBEmbedded.create(disableTelemetry).amazonDynamoDB();

// AWS SDK v2
DynamoDbClient ddbClientSDKv2Local =
 DynamoDBEmbedded.create(disableTelemetry).dynamoDbClient();
```

Tipos de informação coletados

- Informações de uso: telemetria genérica, como início/parada do servidor e a API ou operação chamada.

- Informações do sistema e do ambiente: a versão Java, o sistema operacional (Windows, Linux ou macOS), o ambiente no qual o DynamoDB local é executado (por exemplo, JAR independente, contêiner do Docker ou como dependência do Maven) e valores de hash dos atributos de uso.

Saiba mais

Os dados de telemetria que o DynamoDB local coleta seguem as políticas de privacidade de dados da AWS. Para obter mais informações, consulte as informações a seguir.

- Termos de serviço da AWS
- Perguntas frequentes sobre privacidade de dados

Etapa 1: crie uma tabela no DynamoDB

Nesta etapa, você criará uma tabela `Music` no Amazon DynamoDB. Essa tabela tem os seguintes detalhes:

- Chave de partição: `Artist`
- Chave de classificação: `SongTitle`

Para obter mais informações sobre operações em tabelas, consulte Trabalhar com tabelas e dados no DynamoDB.

> **Note**
>
> Antes de começar, siga as etapas em Pré-requisitos.

AWS Management Console

Para criar uma tabela `Music` usando o console do DynamoDB:

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).
3. Escolha Create table.
4. Insira os Detalhes da tabela da seguinte forma:

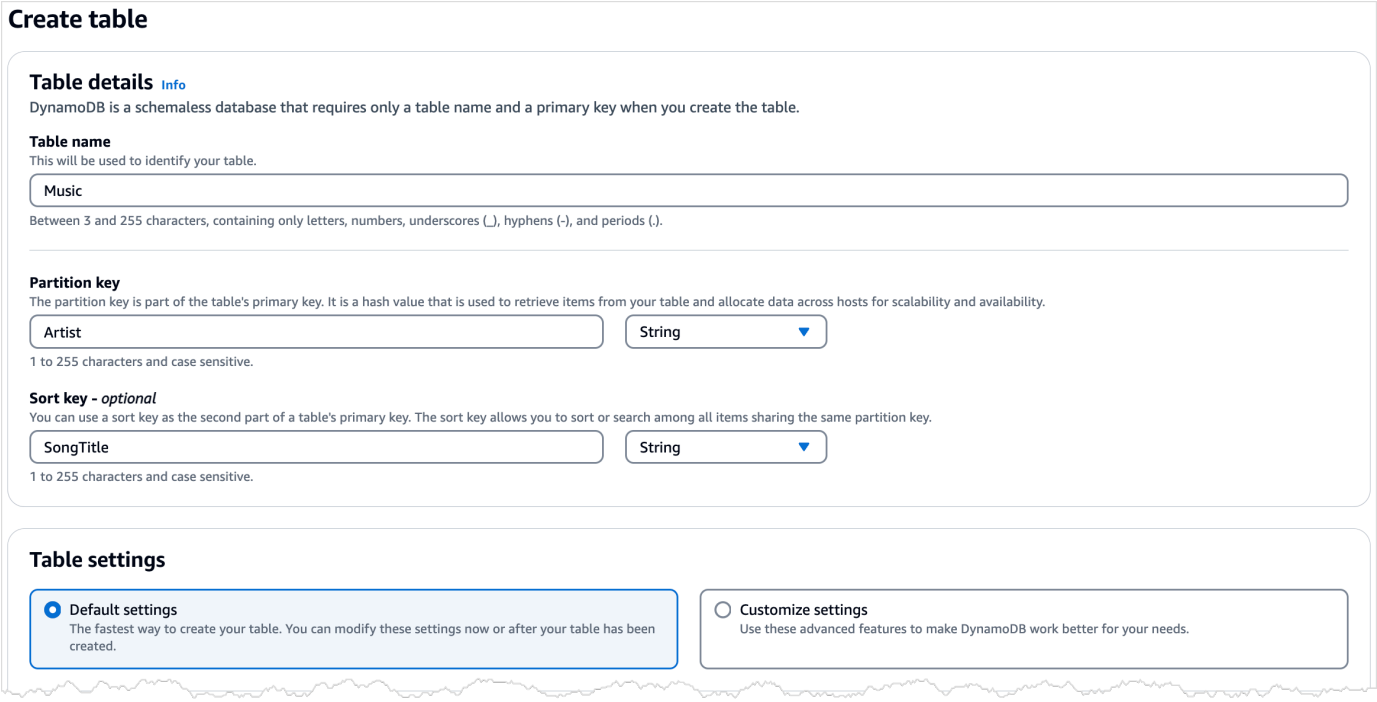
a. Em Table name (Nome da tabela), insira **`Music`**.

b. Em Partition key, (Chave de partição), insira **`Artist`**.

c. Em Chave de classificação, insira **`SongTitle`**.

5. Em Configurações de tabela, mantenha a seleção padrão de Configurações padrão.

6. Selecione Criar tabela para criar a tabela.



7. Quando a tabela está no status `ACTIVE`, é prática recomendada habilitar [Backups para um ponto no tempo do DynamoDB] na tabela realizando as seguintes etapas:

a. Selecione no nome da tabela para abri-la.

b. Escolha Backups.

c. Selecione Editar na seção Recuperação em um ponto anterior no tempo (PITR).

d. Na seção Editar configurações de recuperação a um ponto no tempo, selecione Ativar a recuperação para um ponto no tempo.

e. Escolha Salvar alterações.

AWS CLI

O exemplo da AWS CLI a seguir cria uma nova tabela `Music` usando `create-table`.

Linux

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

CMD do Windows

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

O uso de `create-table` retorna o seguinte resultado de exemplo.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
```

```
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-03-29T12:11:43.379000-04:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:111122223333:table/Music",
        "TableId": "60abf404-1839-4917-a89b-a8b0ab2a1b87",
        "TableClassSummary": {
            "TableClass": "STANDARD"
        }
    }
}
}
```

Observe que o valor do campo `TableStatus` está definido como `CREATING`.

Para verificar se o DynamoDB terminou de criar a tabela `Music`, use o comando `describe-table`.

Linux

```
aws dynamodb describe-table --table-name Music | grep TableStatus
```

CMD do Windows

```
aws dynamodb describe-table --table-name Music | findstr TableStatus
```

Esse comando retorna o seguinte resultado. Quando o DynamoDB conclui a criação da tabela, o valor do campo `TableStatus` é definido como `ACTIVE`.

```
"TableStatus": "ACTIVE",
```

Quando a tabela está no status `ACTIVE`, é uma prática recomendada habilitar Backups para um ponto no tempo do DynamoDB na tabela executando o comando a seguir.

Linux

```
aws dynamodb update-continuous-backups \
    --table-name Music \
    --point-in-time-recovery-specification \
        PointInTimeRecoveryEnabled=true
```

CMD do Windows

```
aws dynamodb update-continuous-backups --table-name Music --point-in-time-recovery-
specification PointInTimeRecoveryEnabled=true
```

Esse comando retorna o seguinte resultado.

```
{
    "ContinuousBackupsDescription": {
        "ContinuousBackupsStatus": "ENABLED",
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": "2023-03-29T12:18:19-04:00",
            "LatestRestorableDateTime": "2023-03-29T12:18:19-04:00"
        }
    }
}
```

> **Note**
>
> Há implicações de custo para a habilitação de backups contínuos com PITR. Para obter informações sobre preços, consulte Definição de preço do Amazon DynamoDB.

AWS SDK

Os exemplos de código a seguir mostram como criar uma tabela do DynamoDB usando um AWS SDK.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
 -a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
#############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
```

```
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte CreateTable na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
```

```
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como criar uma tabela com tags

O exemplo de `create-table` a seguir usa os atributos especificados e o esquema de chaves para criar uma tabela chamada `MusicCollection`. Essa tabela usa um throughput provisionado e é criptografada em repouso usando a CMK de propriedade padrão da AWS. O comando também aplica uma tag à tabela, com uma chave `Owner` e valor de `blueTeam`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
```

```
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como criar uma tabela no modo sob demanda

O exemplo a seguir cria uma tabela chamada `MusicCollection` usando o modo sob demanda, em vez do modo de throughput provisionado. Esse método é útil para tabelas com workloads imprevisíveis.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```


Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 3: como criar uma tabela e criptografá-la com uma CMK gerenciada pelo cliente

O exemplo a seguir cria uma tabela chamada `MusicCollection` e a criptografa usando uma CMK gerenciada pelo cliente.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
```

```
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 4: como criar uma tabela com um índice secundário local

O exemplo a seguir usa os atributos especificados e o esquema de chaves para criar uma tabela chamada `MusicCollection` com um índice secundário local chamado `AlbumTitleIndex`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
```

```
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
```

```
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 5: como criar uma tabela com um índice secundário global

O exemplo a seguir cria uma tabela chamada `GameScores` com um índice secundário global chamado `GameTitleIndex`. A tabela-base tem uma chave de partição `UserId` e uma chave de classificação `GameTitle`, permitindo que você encontre a melhor pontuação de

um usuário individual para um jogo específico de forma eficiente, enquanto o GSI tem uma chave de partição `GameTitle` e uma chave de classificação `TopScore`, permitindo que você encontre rapidamente a pontuação mais alta geral para um jogo específico.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
```

```
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
```

```
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 6: como criar uma tabela com vários índices secundários globais ao mesmo tempo

O exemplo a seguir cria uma tabela chamada `GameScores` com dos índices secundários globais. Os esquemas do GSI são passados por meio de um arquivo, e não pela linha de comando.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Conteúdo de `gsi.json`:

```
[
    {
        "IndexName": "GameTitleIndex",
```

```
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Saída:

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
```

```
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
```

```
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 7: como criar uma tabela com o Streams habilitado

O exemplo a seguir cria uma tabela chamada `GameScores` com o DynamoDB Streams habilitado. Imagens novas e antigas de cada item serão gravadas no fluxo.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
```

```
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 8: como criar uma tabela com o fluxo somente de chaves habilitado

O exemplo a seguir cria uma tabela chamada `GameScores` com o DynamoDB Streams habilitado. Somente os atributos-chave dos itens modificados são gravados no fluxo.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Para obter mais informações, consulte Captura de dados de alterações com o Amazon DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 9: como criar uma tabela com a classe Standard-Infrequent Access

O exemplo a seguir cria uma tabela chamada `GameScores` e atribui a classe de tabela Standard-Infrequent Access (DynamoDB Standard-IA). Essa classe de tabela é otimizada para que o armazenamento seja o custo dominante.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
```

```
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

Para obter mais informações, consulte Classes de tabela no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 10: como criar uma tabela com a proteção contra exclusão habilitada

O exemplo a seguir cria uma tabela chamada `GameScores` e habilita a proteção contra exclusão.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Saída:


```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Para obter mais informações, consulte Usar a proteção contra exclusão no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte CreateTable na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
```

```
   public static void main(String[] args) {
       final String usage = """

           Usage:
               <tableName> <key>

           Where:
               tableName - The Amazon DynamoDB table to create (for example,
Music3).
               key - The key for the Amazon DynamoDB table (for example,
Artist).
           """;

       if (args.length != 2) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       System.out.println("Creating an Amazon DynamoDB table " + tableName + "
with a simple primary key: " + key);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
           .region(region)
           .build();

       String result = createTable(ddb, tableName, key);
       System.out.println("New table is " + result);
       ddb.close();
   }

   public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
       DynamoDbWaiter dbWaiter = ddb.waiter();
       CreateTableRequest request = CreateTableRequest.builder()
           .attributeDefinitions(AttributeDefinition.builder()
               .attributeName(key)
               .attributeType(ScalarAttributeType.S)
               .build())
           .keySchema(KeySchemaElement.builder()
               .attributeName(key)
               .keyType(KeyType.HASH)
               .build())
```

```
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
```

```
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
```

```
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma tabela.

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: este exemplo cria uma tabela chamada Thread que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave).

O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemplo 2: este exemplo cria uma tabela chamada “Thread” que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave). Um índice secundário local também é definido. A chave do índice secundário local será definida automaticamente por meio da chave de hash primária na tabela (ForumName). O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
```

```
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemplo 3: este exemplo mostra como usar um único pipeline para criar uma tabela chamada “Thread” que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave) e um índice secundário local. O Add-DDBKeySchema e o Add-DDBIndexSchema criam um objeto TableSchema para você se um não for fornecido pelo pipeline ou pelo parâmetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para ver detalhes da API, consulte CreateTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: este exemplo cria uma tabela chamada Thread que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave). O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemplo 2: este exemplo cria uma tabela chamada “Thread” que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave). Um índice secundário local também é definido. A chave do índice secundário local será definida automaticamente por meio da chave de hash primária na tabela (ForumName). O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
```

```
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemplo 3: este exemplo mostra como usar um único pipeline para criar uma tabela chamada "Thread" que tem uma chave primária que consiste em "ForumName" (hash do tipo de chave) e "Subject" (intervalo de tipos de chave) e um índice secundário local. O Add-DDBKeySchema e o Add-DDBIndexSchema criam um objeto TableSchema para você se um não for fornecido pelo pipeline ou pelo parâmetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para ver detalhes da API, consulte CreateTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma tabela para armazenar dados de filmes.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
```

```
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
```

```
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
```

```
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
      " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
```

```
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência de API do AWS SDK para Swift.

Para obter mais exemplos do DynamoDB, consulte Exemplos de código do DynamoDB usando AWS SDKs.

Após criar uma nova tabela, acesse Etapa 2: grave dados em uma tabela do DynamoDB.

Etapa 2: grave dados em uma tabela do DynamoDB

Nesta etapa, insira muitos itens na tabela `Music` que você criou em Etapa 1: crie uma tabela no DynamoDB.

Para obter mais informações sobre operações de gravação, consulte Gravar um item.

AWS Management Console

Siga estas etapas para gravar dados na tabela `Music` usando o console do DynamoDB.

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).
3. Na página Tabelas, selecione a tabela Música.
4. Escolha Explore table items (Explorar itens da tabela).
5. Na seção Itens retornados, selecione Criar item.
6. Na página Criar item, faça o seguinte para adicionar itens à tabela:
    a. Selecione Add new attribute (Adicionar novo atributo) e depois escolha Number (Número).
    b. Em Nome do atributo, insira **`Awards`**.
    c. Repita o processo para criar um **`AlbumTitle`** do tipo String.
    d. Escolha os seguintes valores para o item:
        i. Em Artist (Artista), insira **`No One You Know`**.
        ii. Em SongTitle (Nome da música), insira **`Call Me Today`**.
        iii. Em AlbumTitle (Nome do álbum), insira **`Somewhat Famous`**.
        iv. Em Awards (Prêmios), insira **`1`**.
7. Selecione Create Item (Criar item).
8. Repita o processo para criar outro item com os seguintes valores:
    a. Em Artist (Artista), insira **`Acme Band`**.
    b. Em SongTitle (Nome da música) insira **`Happy Day`**.
    c. Em AlbumTitle (Nome do álbum), insira **`Songs About Life`**.
    d. Em Awards (Prêmios), insira **`10`**.
9. Faça isso mais uma vez para criar outro item com o mesmo Artist (Artista) como na etapa anterior, mas com valores diferentes para os outros atributos:

a. Em Artist (Artista), insira **`Acme Band`**.
b. Em SongTitle (Nome da música) insira **`PartiQL Rocks`**.
c. Em AlbumTitle (Nome do álbum), insira **`Another Album Title`**.
d. Em Awards (Prêmios), insira **`8`**.

AWS CLI

O exemplo da AWS CLI a seguir cria muitos itens novos na tabela `Music`. É possível fazer isso com a API do DynamoDB ou PartiQL, uma linguagem de consulta compatível com SQL para o DynamoDB.

DynamoDB API

Linux

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Howdy"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "2"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "PartiQL Rocks"},
 "AlbumTitle": {"S": "Another Album Title"}, "Awards": {"N": "8"}}'
```

CMD do Windows

```
aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Call
 Me Today\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\":
 \"1\"}}"

aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Howdy
\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\": \"2\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day\"},
 \"AlbumTitle\": {\"S\": \"Songs About Life\"}, \"Awards\": {\"N\": \"10\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"PartiQL Rocks
\"}, \"AlbumTitle\": {\"S\": \"Another Album Title\"}, \"Awards\": {\"N\": \"8\"}}"
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Call Me Today',
 'AlbumTitle':'Somewhat Famous', 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Howdy',
 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs
 About Life', 'Awards':'10'}"
```

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
               VALUE  \
               {'Artist':'Acme Band','SongTitle':'PartiQL Rocks',
 'AlbumTitle':'Another Album Title', 'Awards':'8'}"
```

CMD do Windows

```
aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Call Me Today', 'AlbumTitle':'Somewhat Famous',
 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Howdy', 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'PartiQL Rocks', 'AlbumTitle':'Another Album Title',
 'Awards':'8'}"
```

Para obter mais informações sobre gravação de dados com PartiQL, consulte Instruções Insert em PartiQL.

Para obter mais informações sobre tipos de dados compatíveis com o DynamoDB, consulte Tipos de dados.

Para obter mais informações sobre como representar os tipos de dados do DynamoDB em JSON, consulte Valores de atributos.

AWS SDK

Os exemplos de código a seguir mostram como gravar um item em uma tabela do DynamoDB usando um AWS SDK.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
```

```
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi
```

```
  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte PutItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);
```

```
    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
```

```
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como adicionar um item a uma tabela

O exemplo de `put-item` a seguir adiciona um novo item à tabela MusicCollection.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
```

```
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Conteúdo de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Saída:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como substituir condicionalmente um item em uma tabela

O exemplo de `put-item` a seguir substitui um item existente na tabela `MusicCollection` somente se o item existente tiver um atributo `AlbumTitle` com o valor `Greatest Hits`. O comando retorna o valor anterior do item.

```
aws dynamodb put-item \
```

```
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Conteúdo de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Conteúdo de `names.json`:

```
{
    "#A": "AlbumTitle"
}
```

Conteúdo de `values.json`:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Saída:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
```

```
}
```

Se a chave já existir, você verá a seguinte saída:

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte PutItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
```

```
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
```

```
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Coloca um item em uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is
 placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
```

```
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {
```

```
        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Coloque um item em uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Coloque um item em uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
```

```
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
```

```
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: cria um item ou substitui um item por um novo item.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
```

```
Set-DDBItem -TableName 'Music' -Item $item
```

- Para ver detalhes da API, consulte PutItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: cria um item ou substitui um item por um novo item.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Para ver detalhes da API, consulte PutItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
```

```
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
```

```
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
```

```
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();
```

```
    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
```

```
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
```

```
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Para obter detalhes da API, consulte PutItem na Referência de API do AWS SDK para Swift.

Para obter mais exemplos do DynamoDB, consulte Exemplos de código do DynamoDB usando AWS SDKs.

Após gravar os dados na sua tabela, acesse Etapa 3: leia os dados de uma tabela do DynamoDB.

Etapa 3: leia os dados de uma tabela do DynamoDB

Nessa etapa, você lerá um dos itens criados em Etapa 2: grave dados em uma tabela do DynamoDB. Você pode usar o console do DynamoDB ou a AWS CLI para ler um item da tabela `Music` especificando `Artist` e `SongTitle`.

Para ver mais informações sobre as operações de leitura no DynamoDB, consulte Ler um item.

AWS Management Console

Siga estas etapas para ler os dados da tabela `Music` usando o console do DynamoDB.

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).
3. Na página Tabelas, selecione a tabela Música.
4. Escolha Explore table items (Explorar itens da tabela).
5. Na seção Itens retornados, veja a lista de itens armazenados na tabela, classificados por `Artist` e `SongTitle`. O primeiro item da lista é aquele com o Artista chamado Acme Band e o TítulodaMúsica PartiQL Rock.

AWS CLI

O exemplo da AWS CLI a seguir lê um item da tabela `Music`. É possível fazer isso com a API do DynamoDB ou PartiQL, uma linguagem de consulta compatível com SQL para o DynamoDB.

DynamoDB API

> **Note**
>
> O comportamento padrão do DynamoDB é o de leituras finais consistentes. O parâmetro `consistent-read` é usado abaixo para demonstrar leituras altamente consistentes.

Linux

```
aws dynamodb get-item --consistent-read \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'
```

CMD do Windows

```
aws dynamodb get-item --consistent-read ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}"
```

O uso de `get-item` retorna o seguinte resultado de exemplo.

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
WHERE Artist='Acme Band' AND SongTitle='Happy Day'"
```

CMD do Windows

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band' AND SongTitle='Happy Day'"
```

Usar a instrução `Select` PartiQL retorna o resultado de exemplo a seguir.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Songs About Life"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Para obter mais informações sobre leitura de dados com PartiQL, consulte Instruções Select em PartiQL.

AWS SDK

Os exemplos de código a seguir mostram como ler um item em uma tabela do DynamoDB usando um AWS SDK.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
```

```
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- Para obter detalhes da API, consulte GetItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
```

```
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como ler um item em uma tabela

O exemplo de `get-item` a seguir recupera um item da tabela `MusicCollection`. A tabela tem uma chave primária de hash e intervalo (`Artist` e `SongTitle`), portanto, você deve especificar esses dois atributos. O comando também solicita informações sobre a capacidade de leitura consumida pela operação.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Saída:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obter mais informações, consulte Ler um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como ler um item usando uma leitura consistente

O exemplo a seguir recupera um item da tabela `MusicCollection` usando leituras altamente consistentes.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Saída:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obter mais informações, consulte Ler um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 3: como recuperar atributos específicos de um item

O exemplo a seguir usa uma expressão de projeção para recuperar apenas três atributos do item desejado.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Conteúdo de `names.json`:

```
{
```

```
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Saída:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Para obter mais informações, consulte Ler um item no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte GetItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
```

```
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
```

```
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Obtenha um item de uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
```

```
   public static void main(String[] args) {
       final String usage = """

               Usage:
                   <tableName> <key> <keyVal>

               Where:
                   tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                   key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                   keyval - The key value that represents the item to get (for
example, Famous Band).
               """;

       if (args.length != 3) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       String keyVal = args[2];
       System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
               .region(region)
               .build();

       getDynamoDBItem(ddb, tableName, key, keyVal);
       ddb.close();
   }

   public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
       HashMap<String, AttributeValue> keyToGet = new HashMap<>();
       keyToGet.put(key, AttributeValue.builder()
               .s(keyVal)
               .build());

       GetItemRequest request = GetItemRequest.builder()
               .key(keyToGet)
               .tableName(tableName)
```

```
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Obtenha um item de uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
```

```
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtenha um item de uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: exibe o item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Para ver detalhes da API, consulte GetItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: exibe o item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Para ver detalhes da API, consulte GetItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table
```

```
def initialize(table_name)
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamo_resource.table(table_name)
end

# Gets movie data from the table for a specific movie.
#
# @param title [String] The title of the movie.
# @param year [Integer] The release year of the movie.
# @return [Hash] The data about the requested movie.
def get_item(title, year)
  @table.get_item(key: { 'year' => year, 'title' => title })
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename               = iv_table_name
      it_key                     = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
    MESSAGE 'Movie name is: ' && lo_title->get_s( )
      && 'Movie year is: ' && lo_year->get_n( )
```

```
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
```

```
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte GetItem na Referência de API do AWS SDK para Swift.

Para obter mais exemplos do DynamoDB, consulte Exemplos de código do DynamoDB usando AWS SDKs.

Para atualizar os dados na tabela, acesse Etapa 4: atualize os dados em uma tabela do DynamoDB.

Etapa 4: atualize os dados em uma tabela do DynamoDB

Nesta etapa, você atualiza um item que criou em Etapa 2: grave dados em uma tabela do DynamoDB. Você pode usar o console do DynamoDB ou a AWS CLI para atualizar o `AlbumTitle` de um item na tabela `Music` especificando `Artist`, `SongTitle` e o `AlbumTitle` atualizado.

Para obter mais informações sobre operações de gravação, consulte Gravar um item.

AWS Management Console

Você pode usar o console do DynamoDB para atualizar os dados na tabela `Music`.

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).

3. Escolha a tabela Music (Música) na lista de tabelas.
4. Escolha Explore table items (Explorar itens da tabela).
5. Em Itens retornados, para a linha de itens com Acme Band em Artista e Happy Day em TítulodaMúsica, faça o seguinte:

   a. Coloque o cursor no TítulodoÁlbum chamado Songs about Life.

   b. Selecione o ícone Editar.

   c. Na janela pop-up Editar string, digite **`Songs of Twilight`**.

   d. Escolha Salvar.

> **Tip**
>
> Opcionalmente, para atualizar um item, faça o seguinte na seção Itens retornados:
>
> 1. Selecione a linha do item com o Artista chamado Acme Band e o TítulodaMúsica Happy Day.
> 2. Na lista suspensa Ações, selecione Editar item.
> 3. Em TítulodoÁlbum, insira **`Songs of Twilight`**.
> 4. Escolha Save and close.

AWS CLI

O exemplo da AWS CLI a seguir atualiza um item da tabela `Music`. É possível fazer isso com a API do DynamoDB ou PartiQL, uma linguagem de consulta compatível com SQL para o DynamoDB.

DynamoDB API

Linux

```
aws dynamodb update-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' \
    --update-expression "SET AlbumTitle = :newval" \
    --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' \
    --return-values ALL_NEW
```

CMD do Windows

```
aws dynamodb update-item ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}" ^
    --update-expression "SET AlbumTitle = :newval" ^
    --expression-attribute-values "{\":newval\":{\"S\":\"Updated Album Title\"}}" ^
    --return-values ALL_NEW
```

Usar `update-item` retornará o seguinte resultado de exemplo porque `return-values` `ALL_NEW` foi especificado.

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Updated Album Title"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "UPDATE Music  \
    SET AlbumTitle='Updated Album Title'  \
    WHERE Artist='Acme Band' AND SongTitle='Happy Day' \
    RETURNING ALL NEW *"
```

CMD do Windows

```
aws dynamodb execute-statement --statement "UPDATE Music SET AlbumTitle='Updated
 Album Title' WHERE Artist='Acme Band' AND SongTitle='Happy Day' RETURNING ALL NEW
 *"
```

Usar a instrução `Update` retornará o seguinte resultado de exemplo porque `RETURNING ALL NEW *` foi especificado.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Para obter mais informações sobre a atualização de dados com PartiQL, consulte Instruções Update em PartiQL .

AWS SDK

Os exemplos de código a seguir mostram como atualizar um item em uma tabela do DynamoDB usando um AWS SDK.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },
```

```
                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
```

```
    echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
 }

 while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
 fi
 if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
 fi

 if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}
```

```
#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
```

```
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API do AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como atualizar um item em uma tabela

O exemplo da `update-item` a seguir atualiza um item da tabela `MusicCollection`. Ele adiciona um novo atributo (`Year`) e modifica o atributo `AlbumTitle`. Todos os atributos no item, conforme aparecem após a atualização, são retornados na resposta.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Conteúdo de `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Conteúdo de `expression-attribute-values.json`:

```
{
```

```
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Saída:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como atualizar um item de forma condicional

O exemplo a seguir atualiza um item na tabela `MusicCollection`, mas somente se o item existente ainda não tiver um atributo `Year`.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Conteúdo de `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Conteúdo de `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Se o item já tiver um atributo `Year`, o DynamoDB retornará saída a seguir.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte UpdateItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
```

```
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
```

```
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Atualiza um item em uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
```

```
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
example, Awards).
                    updateVal - The value used to update an item (for example,
14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());
```

```
        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
```

```
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: define o atributo de gênero como “Rap” no item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para ver detalhes da API, consulte UpdateItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: define o atributo de gênero como “Rap” no item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
```

```
Update-DDBItem @updateDdbItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para ver detalhes da API, consulte UpdateItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Atualize um item usando uma expressão de atualização.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
```

```
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Atualize um item usando uma expressão de atualização que inclui uma operação aritmética.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Atualize um item somente quando ele atender a determinadas condições.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
```

```
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
```

```
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obter os detalhes da API, consulte UpdateItem na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
 nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
 [:]

            if rating != nil {
```

```
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência de API do AWS SDK para Swift.

Para obter mais exemplos do DynamoDB, consulte Exemplos de código do DynamoDB usando AWS SDKs.

Para consultar os dados na tabela `Music`, acesse Etapa 5: consulte os dados de uma tabela do DynamoDB.

Etapa 5: consulte os dados de uma tabela do DynamoDB

Nesta etapa, você consulta dados que gravou na tabela `Music` em the section called “Etapa 2: gravar dados” especificando `Artist`. Isso exibirá todas as músicas associadas à chave de partição: `Artist`.

Para obter mais informações sobre as operações de consulta, consulte Consultar tabelas no DynamoDB.

AWS Management Console

Siga estas etapas para usar o console do DynamoDB para consultar dados em uma tabela `Music`.

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).
3. Escolha a tabela Music (Música) na lista de tabelas.
4. Escolha Explore table items (Explorar itens da tabela).
5. Em Verificar ou consultar itens, verifique se a opção Consultar está selecionada.
6. Em Partition key (Chave de partição), insira **`Acme Band`** e escolha Run (Executar).

AWS CLI

O exemplo da AWS CLI a seguir consulta um item da tabela `Music`. É possível fazer isso com a API do DynamoDB ou PartiQL, uma linguagem de consulta compatível com SQL para o DynamoDB.

DynamoDB API

Consulte um item por meio da API do DynamoDB usando `query` e informando a chave de partição.

Linux

```
aws dynamodb query \
```

```
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

CMD do Windows

```
aws dynamodb query ^
    --table-name Music ^
    --key-condition-expression "Artist = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Acme Band\"}}"
```

Usar `query` retornará todas as músicas associadas a este `Artist` específico.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "N": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "N": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
```

```
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

PartiQL for DynamoDB

Consulte um item com PartiQL usando a instrução `Select` e informando a chave de partição.

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band'"
```

CMD do Windows

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band'"
```

Usar a instrução `Select` desse modo retornará todas as músicas associadas a este `Artist` específico.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
```

```
            "Awards": {
                "S": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Para obter mais informações sobre consulta de dados com PartiQL, consulte Instruções Select em PartiQL.

AWS SDK

Os exemplos de código a seguir mostram como consultar uma tabela do DynamoDB usando um AWS SDK.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
```

```
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
```

```
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
```

```
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
```

```
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
```

```
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
```

```
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte Query na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
```

```
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::QueryRequest request;

    request.SetTableName(tableName);

    if (!projectionExpression.empty()) {
        request.SetProjectionExpression(projectionExpression);
    }

    // Set query key condition expression.
    request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

    // Set Expression AttributeValues.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
    attributeValues.emplace(":valueToMatch", partitionValue);

    request.SetExpressionAttributeValues(attributeValues);

    bool result = true;

    // "exclusiveStartKey" is used for pagination.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 exclusiveStartKey;
    do {
        if (!exclusiveStartKey.empty()) {
            request.SetExclusiveStartKey(exclusiveStartKey);
```

```
            exclusiveStartKey.clear();
        }
        // Perform Query operation.
        const Aws::DynamoDB::Model::QueryOutcome &outcome =
 dynamoClient.Query(request);
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "Number of items retrieved from Query: " <<
 items.size()
                          << std::endl;
                // Iterate each item and print.
                for (const auto &item: items) {
                    std::cout
                            <<
 "*************************************************"
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como consultar uma tabela

O exemplo da `query` a seguir consulta itens da tabela `MusicCollection`. A tabela tem uma chave primária de hash e intervalo (`Artist` e `SongTitle`), mas essa consulta especifica apenas o valor da chave de hash. Ela retorna nomes de músicas do artista “No One You Know”.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Conteúdo de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
```

```
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como consultar uma tabela usando leituras altamente consistentes e percorrer o índice em ordem decrescente

O exemplo a seguir executa a mesma consulta do primeiro exemplo, mas retorna os resultados na ordem inversa e usa leituras altamente consistentes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Conteúdo de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
```

```
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 3: como filtrar resultados específicos

O exemplo a seguir consulta o `MusicCollection`, mas exclui os resultados com valores específicos no atributo `AlbumTitle`. Observe que isso não afeta `ScannedCount` ou `ConsumedCapacity` já que o filtro é aplicado após a leitura dos itens.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Conteúdo de `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Conteúdo de `names.json`:

```
{
    "#n1": "Artist",
```

```
    "#n2": "AlbumTitle"
}
```

Saída:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 4: como recuperar somente uma contagem de itens

O exemplo a seguir recupera uma contagem de itens que correspondem à consulta, mas não recupera os itens em si.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Conteúdo de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 5: como consultar um índice

O exemplo a seguir consulta o índice secundário local `AlbumTitleIndex`. A consulta retorna todos os atributos da tabela base projetados no índice secundário local. Ao consultar um índice secundário local ou global, você deve fornecer o nome da tabela base usando o parâmetro `table-name`.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Conteúdo de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
```

```
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte Query na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
```

```
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
```

```
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte Query na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Consulta uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
```

```
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
```

```
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Consulta uma tabela usando o `DynamoDbClient` e um índice secundário.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
```

```
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
```

```
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
```

```
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
```

```
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: invoca uma consulta que exibe itens do DynamoDB com SongTitle e Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
```

```
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
```

- Para ver detalhes da API, consulte Query na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: invoca uma consulta que exibe itens do DynamoDB com SongTitle e Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Saída:

```
Name                            Value
----                            -----
Genre                           Country
Artist                          No One You Know
Price                           1.94
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
```

- Para ver detalhes da API, consulte Query na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Consulte itens usando uma expressão de condição de chave.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

Consulte itens e projete-os para retornar um subconjunto de dados.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
```

```
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Encontre os filmes feitos no ano especificado.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
```

```
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obter os detalhes da API, consulte Query na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
```

```
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
```

```
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência de API do AWS SDK para Swift.

Para obter mais exemplos do DynamoDB, consulte Exemplos de código do DynamoDB usando AWS SDKs.

Para criar um índice secundário global para sua tabela, acesse Etapa 6: (Opcional) exclua uma tabela do DynamoDB para limpar os recursos.

Etapa 6: (Opcional) exclua uma tabela do DynamoDB para limpar os recursos

Se a tabela do Amazon DynamoDB criada durante o tutorial não for mais necessária, você poderá excluí-la. Esta etapa ajuda a garantir que você não será cobrado pelos recursos que não está utilizando. Use o console do DynamoDB ou a AWS CLI para excluir a tabela `Music` que você criou em Etapa 1: crie uma tabela no DynamoDB.

Para ver mais informações sobre as operações de tabelas no DynamoDB, consulte Trabalhar com tabelas e dados no DynamoDB.

AWS Management Console

Para excluir uma tabela `Music` usando o console:

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).
3. Marque a caixa de seleção ao lado da tabela Música na lista de tabelas.
4. Escolha Excluir.

AWS CLI

O exemplo da AWS CLI a seguir exclui a tabela `Music` usando `delete-table`.

```
aws dynamodb delete-table --table-name Music
```

AWS SDK

Os exemplos de código a seguir mostram como excluir uma tabela do DynamoDB usando um SDK da AWS.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
```

```
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
```

```
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
```

```
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
########################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para C++.

CLI

AWS CLI

Como excluir uma tabela

O exemplo de `delete-table` a seguir exclui a tabela `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Saída:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Para obter mais informações, consulte Excluir uma tabela no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte DeleteTable na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
```

```
 return err
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).
```

```
                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
```

```
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: exclui a tabela especificada. A confirmação será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable"
```

Exemplo 2: exclui a tabela especificada. A confirmação não será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para ver detalhes da API, consulte DeleteTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: exclui a tabela especificada. A confirmação será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable"
```

Exemplo 2: exclui a tabela especificada. A confirmação não será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para ver detalhes da API, consulte DeleteTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
```

```
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        " Wait till the table is actually deleted.
```

```
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
```

```
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Swift.

Para obter mais exemplos do DynamoDB, consulte Exemplos de código do DynamoDB usando AWS SDKs.

Aprender mais sobre o DynamoDB

Para obter mais informações sobre o uso do Amazon DynamoDB consulte os seguintes tópicos:

- Trabalhar com tabelas e dados no DynamoDB
- Trabalhar com itens e atributos no DynamoDB
- Consultar tabelas no DynamoDB
- Como usar índices secundários globais no DynamoDB
- Trabalhar com transações
- Aceleração em memória com o DynamoDB Accelerator (DAX)
- Programação com o DynamoDB e os AWS SDKs

Amazon DynamoDB: como funciona

As seções a seguir fornecem uma visão geral dos componentes do serviço Amazon DynamoDB e como eles interagem.

Tópicos

- Folha de dicas para o DynamoDB
- Componentes principais do Amazon DynamoDB
- API do DynamoDB
- Tipos de dados compatíveis e regras de nomenclatura no Amazon DynamoDB
- Classes de tabelas do DynamoDB
- Partições e distribuição de dados no DynamoDB
- Saiba como passar do SQL para o NoSQL
- Recursos e ferramentas de aprendizado do Amazon DynamoDB

Folha de dicas para o DynamoDB

Esta folha de dicas fornece uma referência rápida para trabalhar com o Amazon DynamoDB e seus vários AWS SDKs.

Configuração inicial

1. Cadastre-se no AWS.
2. Obtenha uma chave de acesso da AWS (usada para acessar o DynamoDB de forma programática).
3. Configure suas credenciais do DynamoDB.

Consulte também:

- Configurar o DynamoDB (serviço da Web)
- Conceitos básicos do DynamoDB
- Visão geral básica dos componentes principais

SDK ou CLI

Escolha seu SDK preferido ou configure a AWS CLI.

> **Note**
>
> Quando você usa a AWS CLI no Windows, uma barra invertida (\) que não esteja dentro de uma cotação será tratada como um retorno de carro. Além disso, você deve escapar de todas as aspas e chaves dentro de outras aspas. Como exemplo, consulte a guia Windows em “Criar uma tabela” na próxima seção.

Consulte também:

- AWS CLI com o DynamoDB
- Conceitos básicos do DynamoDB: etapa 2

Ações básicas

Esta seção fornece código para tarefas básicas do DynamoDB. Para obter mais informações sobre essas tarefas, consulte Conceitos básicos do DynamoDB e dos AWS SDKs.

Criar uma tabela

Default

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows

```
aws dynamodb create-table ^
```

```
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

Gravar um item em uma tabela

```
aws dynamodb put-item \ --table-name Music \ --item file://item.json
```

Ler um item de uma tabela

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

Excluir um item de uma tabela

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

Consultar uma tabela

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

Excluir uma tabela

```
aws dynamodb delete-table --table-name Music
```

Listar nomes de tabela

```
aws dynamodb list-tables
```

Regras de nomenclatura

- Todos os nomes devem ser codificados usando UTF-8 e diferenciam maiúsculas de minúsculas.

- Nomes de tabelas e nomes de índices devem ter entre 3 e 255 caracteres e podem conter somente os seguintes caracteres:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (sublinhado)
  - `-` (hífen)
  - `.` (ponto)
- Os nomes de atributos devem ter pelo menos um caractere e no mínimo 64 KB.

Para obter mais informações, consulte Regras de nomenclatura.

Noções básicas de cota de serviço

Unidades de leitura e gravação

- Unidade de capacidade de leitura (RCU): uma leitura altamente consistente por segundo, ou duas leituras finais consistentes por segundo, para itens até 4 KB de tamanho.
- Unidade de capacidade de gravação (WCU): uma gravação por segundo para itens de até 1 KB de tamanho.

Limites da tabela

- Tamanho da tabela: não há limite prático para o tamanho de uma tabela. As tabelas não são limitadas em termos de número de itens ou de bytes.
- Número de tabelas: para qualquer conta da AWS, há uma cota inicial de 2,5 mil tabelas por região da AWS.
- Limite de tamanho de página para consulta e verificação: há um limite de 1 MB por página, por consulta ou verificação. Se os seus parâmetros de consulta ou operação de verificação em uma tabela resultarem em mais de 1 MB de dados, o DynamoDB retornará os itens correspondentes iniciais. Ele também retorna uma propriedade `LastEvaluatedKey` que pode ser usada em uma nova solicitação para ler a próxima página.

Índices

- Índices secundários locais (LSIs): é possível definir um máximo de cinco índices secundários locais. Os LSIs são úteis principalmente quando um índice deve ter uma consistência alta com a tabela-base.
- Índices secundários globais (GSIs): há uma cota padrão de 20 índices secundários globais por tabela.
- Atributos de índices secundários projetados por tabela: é possível projetar um total de até 100 atributos em todos os índices secundários locais e globais de uma tabela. Isso se aplica somente a atributos projetados especificados pelo usuário.

Chaves de partição

- O tamanho mínimo de um valor de chave de partição é 1 byte. O comprimento máximo é de 2048 bytes.
- Não há um limite prático para o número de valores de chave de partição distintos, para tabelas ou para índices secundários.
- O tamanho mínimo de um valor de chave de classificação é 1 byte. O comprimento máximo é de 1024 bytes.
- Em geral, não há limite prático para o número de valores de chave de classificação distintos por valor de chave de partição. Há exceção para tabelas com índices secundários.

Para obter mais informações sobre índices secundários, design de chave de partição e design de chave de classificação, consulte Práticas recomendadas.

Limites para tipos de dados comumente usados

- String: o tamanho de uma string é restrito pelo tamanho de item máximo de 400 KB. Strings são Unicode com codificação binária UTF-8.
- Número: um número pode ter até 38 dígitos de precisão, e pode ser positivo, negativo ou zero.
- Binário: o tamanho de um binário é restrito pelo tamanho de item máximo de 400 KB. As aplicações que funcionam com atributos binários devem codificar os dados em formato base64 antes de enviá-los para o DynamoDB.

Para obter uma lista dos tipos de dados compatíveis, consulte Tipos de dados. Para obter mais informações, consulte Service quotas.

Itens, atributos e parâmetros de expressão

O tamanho de item máximo no DynamoDB é 400 KB, o que inclui o tamanho binário do nome do atributo (tamanho UTF-8) e os tamanhos binários do valor dos atributo (tamanho UTF-8). O nome do atributo conta para o limite de tamanho.

Não há limite para o número de valores em uma lista, um mapa ou um conjunto, desde que o item que contenha os valores permaneça no limite de tamanho de item de 400 KB.

Para os parâmetros de expressão, o tamanho máximo de qualquer string de expressão é 4 KB.

Para obter mais informações sobre tamanho do item, atributos e parâmetros de expressão, consulte Service quotas.

Mais informações

- Segurança
- Monitorar e registrar
- Trabalhar com fluxos
- Backups e recuperação pontual
- Integrar-se a outros serviços da AWS
- Referência de API
- Centro de arquitetura: práticas recomendadas de banco de dados
- Vídeos de tutorial
- Fórum do DynamoDB

Componentes principais do Amazon DynamoDB

No DynamoDB, tabelas, itens e atributos são os componentes principais com que você trabalha. Uma tabela é uma coleção de itens, e cada item é uma coleção de atributos. O DynamoDB utiliza chaves primárias para identificar de modo exclusivo cada item em uma tabela. Você pode usar o DynamoDB Streams para capturar eventos de modificação de dados em tabelas do DynamoDB.

No entanto, há limites no DynamoDB. Para obter mais informações, consulte Cotas no Amazon DynamoDB.

O vídeo a seguir apresenta uma introdução sobre tabelas globais, itens e atributos.

Tabelas, itens e atributos

Tabelas, itens e atributos

People

Primary key Partition key: PersonID	Attributes			
101	LastName	FirstName	Phone	
	Smith	Fred	555-4321	
102	LastName	FirstName	Address	
	Jones	Mary	{"Street":"123 Main","City":"Anytown","State":"OH","ZIPCode":12345}	
103	LastName	FirstName	FavoriteColor	Address
	Stephens	Howard	Blue	{"Street":"123 Main","City":"London","PostalCode":"ER3 5K8"}

Estes são os componentes básicos do DynamoDB:

- Tabelas: semelhante a outros sistemas de banco de dados, o DynamoDB armazena dados em tabelas. Uma tabela é uma coleção de dados. Por exemplo, consulte a tabela de exemplo People que você pode usar para armazenar informações pessoais de contato de amigos, familiares ou qualquer outra pessoa de interesse. Você também poder ter uma tabela Cars para armazenar informações sobre os veículos que as pessoas dirigem.
- Items: cada tabela contém zero ou mais itens. Um item é um grupo de atributos identificável exclusivamente entre todos os outros itens. Na tabela People de exemplo, cada item representa uma pessoa. Na tabela Cars, cada item representa um veículo. Os itens no DynamoDB são semelhantes de muitas formas a linhas, registros ou tuplas em outros sistemas de banco de dados. No DynamoDB, não há limite para o número de itens que você pode armazenar em uma tabela.
- Atributos: cada item é composto por um ou mais atributos. Um atributo é um elemento de dados fundamental, algo que não precisa ser dividido ainda mais. Por exemplo, um item na tabela People contém os atributos PersonID, LastName, FirstName e assim por diante. Em uma tabela Department, um item pode ter atributos como DepartmentID, Name, Manager, etc. Os atributos no

DynamoDB se parecem de várias maneiras com campos ou colunas em outros sistemas de banco de dados.

O diagrama a seguir mostra uma tabela People com alguns itens e atributos de exemplo.

```
People

{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}

{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}

{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
                "Street": "123 Main",
                "City": "London",
                "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

Observe o seguinte sobre a tabela People:

- Cada item da tabela tem um identificador exclusivo, ou chave primária, que o distingue de todos os outros na tabela. Na tabela People, a chave primária consiste em um único atributo (PersonID).
- Além da chave primária, a tabela People não tem esquema, o que significa que não é necessário definir seus atributos e tipos de dados previamente. Cada item pode ter seus próprios atributos distintos.
- A maioria dos atributos é escalar, o que significa que podem ter apenas um valor. Strings e números são exemplos comuns de escalares.
- Alguns itens têm um atributo aninhado (Address). O DynamoDB oferece suporte a atributos aninhados com até 32 níveis de profundidade.

Veja a seguir outra tabela de exemplo chamada Music, que você pode usar para controlar sua coleção de músicas.

```
Music

{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}

{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
```

```
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Observe o seguinte sobre a tabela Music:

- A chave primária da tabela Music é composta por dois atributos (Artist e SongTitle). Cada item da tabela deve ter esses dois atributos. A combinação de Artist e SongTitle distingue cada item da tabela de todos os outros.
- Além da chave primária, a tabela Music não tem esquema, o que significa que nem os atributos nem seus tipos de dados precisam ser definidos previamente. Cada item pode ter seus próprios atributos distintos.
- Um dos itens tem um atributo aninhado (PromotionInfo) que contém outros atributos aninhados. O DynamoDB oferece suporte a atributos aninhados com até 32 níveis de profundidade.

Para obter mais informações, consulte Trabalhar com tabelas e dados no DynamoDB.

Chave primária

Ao criar uma tabela, além do nome dela, você deve especificar a chave primária da tabela. A chave primária identifica exclusivamente cada item na tabela, de modo que não possa haver dois itens com a mesma chave.

O DynamoDB é compatível com dois tipos diferentes de chaves primárias:

- Chave de partição: uma chave primária simples, formada por um atributo conhecido como a chave de partição.

  O DynamoDB usa o valor da chave de partição como entrada para uma função de hash interna. A saída da função de hash determina a partição (armazenamento físico interno do DynamoDB) em que o item será armazenado.

  Em uma tabela que possui somente uma chave de partição, dois itens não podem ter o mesmo valor de chave de partição.

  A tabela People descrita em Tabelas, itens e atributos é um exemplo de uma tabela com uma chave primária simples (PersonID). Você pode acessar diretamente qualquer item na tabela People fornecendo o valor PersonId desse item.
- Chave de partição e chave de classificação: conhecidas como chave primária composta, esse tipo de chave é formado por dois atributos. O primeiro atributo é a chave de partição, e o segundo atributo é a chave de classificação.

  O DynamoDB usa o valor da chave de partição como entrada para uma função de hash interna. A saída da função de hash determina a partição (armazenamento físico interno do DynamoDB) em que o item será armazenado. Todos os itens com o mesmo valor de chave de partição são armazenados juntos, na ordem classificada por valor de chave de classificação.

  Em uma tabela que tenha uma chave de partição e uma chave de classificação, vários itens podem ter o mesmo valor de chave de partição. No entanto, esses itens devem ter valores de chave de classificação diferentes.

  A tabela Music descrita em Tabelas, itens e atributos é um exemplo de uma tabela com chave primária composta (Artist e SongTitle). Você poderá acessar diretamente qualquer item da tabela Music, se fornecer os valores Artist e SongTitle desse item.

  Uma chave primária composta oferece flexibilidade adicional ao consultar dados. Por exemplo, se você fornecer somente o valor de Artist, o DynamoDB recuperará todas as músicas desse artista.

Para recuperar apenas um subconjunto de músicas de um determinado artista, você pode fornecer um valor para Artist com um intervalo de valores de SongTitle.

> **Note**
>
> A chave de partição de um item também é conhecida como seu atributo de hash. O termo atributo de hash é derivado do uso de uma função de hash interna no DynamoDB que distribui uniformemente os itens de dados entre partições com base em seus valores de chave de partição.
>
> A chave de classificação de um item também é conhecida como seu atributo de intervalo. O termo atributo de intervalo deriva da forma como o DynamoDB armazena itens fisicamente próximos com a mesma chave de partição, classificados em ordem de valor da chave de classificação.

Cada atributo de chave primária deve ser um escalar (o que significa que ele só pode conter um valor). Os únicos tipos de dados permitidos para atributos de chave primária são string, número ou binário. Essas restrições não existem para outros atributos não relacionados a chaves.

Índices secundários

É possível criar um ou mais índices secundários em uma tabela. Um índice secundário permite consultar os dados na tabela usando uma chave alternativa, além de consultas com base na chave primária. O DynamoDB não exige que você use índices, mas eles conferem às aplicações mais flexibilidade durante a consulta dos dados. Depois de criar um índice secundário em uma tabela, você pode ler os dados do índice da mesma maneira que faz na tabela.

O DynamoDB aceita dois tipos de índices:

- Índice secundário global: um índice com uma chave de partição e uma chave de classificação que podem ser diferentes daquelas contidas na tabela. Os valores da chave primária em um índice secundário global não precisam ser exclusivos.
- Índice secundário local: um índice que tem a mesma chave de partição que a tabela, mas uma chave de classificação diferente.

No DynamoDB, os índices secundários globais (GSIs) são índices que abrangem toda a tabela, permitindo que você faça consultas em todas as chaves de partição. Índices secundários locais

(LSIs) são índices com a mesma chave de partição que a tabela base, mas uma chave de classificação diferente.

Cada tabela no DynamoDB tem uma cota de 20 índices secundários globais (cota padrão) e 5 índices secundários locais.

Na tabela Music de exemplo mostrada anteriormente, é possível consultar itens de dados por Artist (chave de partição) ou por Artist e SongTitle (chave de partição e chave de classificação). E se você também quiser consultar os dados por Genre e AlbumTitle? Para fazer isso, você pode criar um índice em Genre e AlbumTitle e, em seguida, consultar esse índice da mesma forma como consulta a tabela Music.

O diagrama a seguir mostra a tabela Music de exemplo, com um novo índice chamado GenreAlbumTitle. No índice, Genre é a chave de partição e AlbumTitle é a chave de classificação.

Tabela de música	GenreAlbumTitle

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Hey Now",
	    "Artist": "No One You Know",
	    "SongTitle": "My Dog Spot"
	}</pre>

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The
 Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Somewhat Famous",
	    "Artist": "No One You Know",
	    "SongTitle": "Somewhere Down The
	 Road"
	}</pre>

Tabela de música	GenreAlbumTitle

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": {
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        },
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Still In Love"
	}</pre>

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 0.99,
    "Genre": "Rock"
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Look Out, World"
	}</pre>

Observe o seguinte sobre o índice GenreAlbumTitle:

- Cada índice pertence a uma tabela, que é chamada de tabela-base para o índice. No exemplo anterior, Music é a tabela-base do índice GenreAlbumTitle.
- O DynamoDB mantém os índices automaticamente. Quando você adiciona, atualiza ou exclui um item na tabela-base, o DynamoDB adiciona, atualiza ou exclui o item correspondente em quaisquer índices que pertençam a essa tabela.
- Ao criar um índice, você especifica quais atributos serão copiados, ou projetados, da tabela-base para o índice. No mínimo, o DynamoDB projeta os atributos de chave da tabela-base no índice. Este é o caso com `GenreAlbumTitle`, em que somente os atributos de chave da tabela `Music` são projetados no índice.

Você pode consultar o índice GenreAlbumTitle para localizar todos os álbuns de um gênero específico (por exemplo, todos os álbuns de Rock). Você também pode consultar o índice para localizar todos os álbuns de um gênero específico que tenham determinados títulos de álbum (por exemplo, todos os álbuns Country com títulos que começam com a letra H).

Para obter mais informações, consulte Melhorar o acesso aos dados com índices secundários no DynamoDB.

DynamoDB Streams

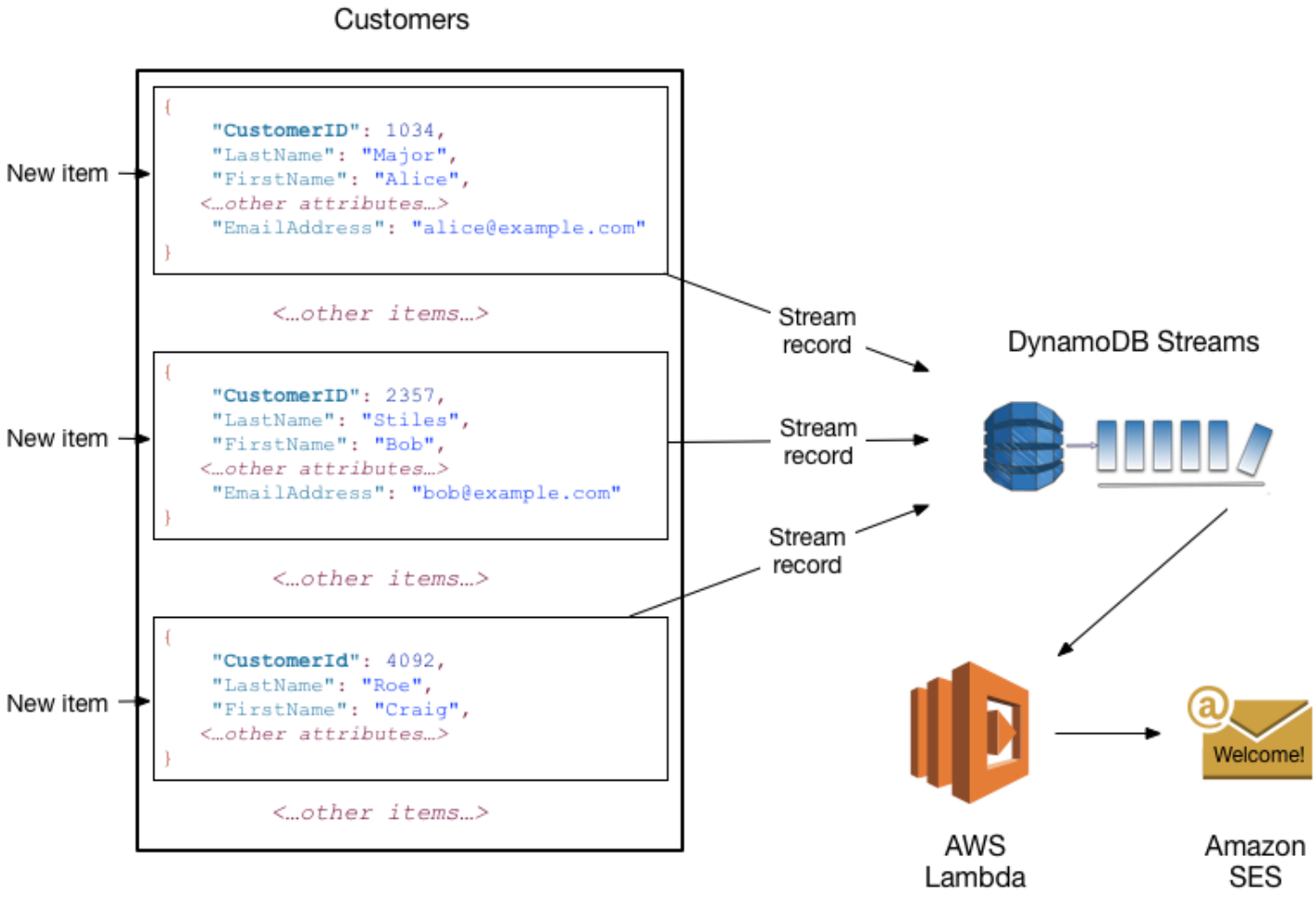
O DynamoDB Streams é um recurso opcional que captura eventos de modificação de dados em tabelas do DynamoDB. Os dados sobre esses eventos são exibidos no fluxo quase em tempo real e na ordem em que os eventos ocorreram.

Cada evento é representado por um registro de fluxo. Se você habilitar um fluxo em uma tabela, o DynamoDB Streams gravará um registro de fluxo sempre que um dos seguintes eventos ocorrer:

- Um novo item é adicionado à tabela: o fluxo captura uma imagem de todo o item, inclusive todos os atributos.
- Um item é atualizado: o fluxo captura as imagens “antes” e “depois” de todos os atributos que tenham sido modificados no item.
- Um item é excluído da tabela: o fluxo captura uma imagem de todo o item antes de ter sido excluído.

Cada registro de fluxo também contém o nome da tabela, o carimbo de data/hora do evento e outros metadados. Registros de fluxo tem um tempo de vida de 24 horas; depois disso, eles são automaticamente removidos do fluxo.

Você pode usar o DynamoDB Streams junto com o AWS Lambda para criar um acionador, um código que é executado automaticamente sempre que um evento de interesse aparece em um fluxo. Por exemplo, considere uma tabela Customers que contém informações sobre os clientes de uma empresa. Suponhamos que você queira enviar um e-mail de “boas-vindas” a cada novo cliente. Você pode habilitar um fluxo nessa tabela e depois associá-lo a uma função do Lambda. A função do Lambda seria executada sempre que um novo registro de fluxo aparecesse, mas processaria somente os novos itens adicionados à tabela Customers. Para qualquer item que tenha um atributo `EmailAddress`, a função do Lambda invocaria o Amazon Simple Email Service (Amazon SES) para enviar um e-mail a esse endereço.



> **Note**
>
> Neste exemplo, o último cliente, Craig Roe, não receberá um e-mail, pois não tem um `EmailAddress`.

Além de acionadores, o DynamoDB Streams possibilita implementar soluções poderosas, como a replicação de dados dentro e entre regiões da AWS, visualizações materializadas de dados em tabelas do DynamoDB, análise de dados usando visualizações materializadas do Kinesis e muito mais.

Para obter mais informações, consulte Capturar dados de alterações para o DynamoDB Streams.

API do DynamoDB

Para trabalhar com o Amazon DynamoDB, a aplicação deve usar algumas operações de API simples. Veja a seguir um resumo dessas operações, organizadas por categoria.

> **Note**
>
> Para obter uma lista completa de operações de API, consulte a Referência de API do Amazon DynamoDB.

Tópicos

- Ambiente de gerenciamento
- Plano de dados
- DynamoDB Streams
- Transações

Ambiente de gerenciamento

As operações do ambiente de gerenciamento permitem criar e gerenciar tabelas do DynamoDB. Elas também permitem que você trabalhe com índices, fluxos e outros objetos que são dependentes de tabelas.

- `CreateTable`: cria uma nova tabela. Opcionalmente, você pode criar um ou mais índices secundários e habilitar o DynamoDB Streams para a tabela.
- `DescribeTable`: retorna informações sobre uma tabela, como seu esquema de chaves primárias, configurações de throughput e informações de índice.
- `ListTables`: retorna os nomes de todas as suas tabelas em uma lista.

- `UpdateTable`: modifica as configurações de uma tabela ou de seus índices, cria ou remove novos índices em uma tabela ou modifica as configurações do DynamoDB Streams para uma tabela.
- `DeleteTable`: remove uma tabela e todos os seus objetos dependentes do DynamoDB.

Plano de dados

As operações do plano de dados permitem criar, ler, atualizar e excluir (também chamadas de CRUD) nos dados de uma tabela. Algumas das operações de plano de dados também permitem que você leia dados de um índice secundário.

Você pode usar o PartiQL: uma linguagem de consultas compatível com SQL para o Amazon DynamoDB para executar essas operações CRUD ou pode usar as APIs CRUD clássicas do DynamoDB que separam cada operação em uma chamada de API distinta.

PartiQL: uma linguagem de consultas compatível com SQL

- `ExecuteStatement`: lê vários itens de uma tabela. Você também pode gravar ou atualizar um único item de uma tabela. Ao gravar ou atualizar um único item, é necessário especificar os atributos de chave primária.
- `BatchExecuteStatement`: grava, atualiza ou lê vários itens de uma tabela. Isso é mais eficiente do que `ExecuteStatement`, pois sua aplicação precisa apenas de uma única viagem de ida e volta na rede para gravar ou ler os itens.

APIs clássicas

Criar dados

- `PutItem`: grava um único item em uma tabela. Você deve especificar os atributos de chave primária, mas não precisa especificar outros atributos.
- `BatchWriteItem`: grava até 25 itens em uma tabela. Isso é mais eficiente do que chamar `PutItem` várias vezes, pois seu aplicativo precisa apenas de uma única viagem de ida e volta na rede para gravar os itens.

Leitura de dados

- `GetItem`: recupera um único item de uma tabela. É necessário especificar a chave primária do item desejado. É possível recuperar o item inteiro ou apenas um subconjunto dos seus atributos.
- `BatchGetItem`: recupera até 100 itens de uma ou mais tabelas. Isso é mais eficiente do que chamar `GetItem` várias vezes, pois seu aplicativo precisa apenas de uma única viagem de ida e volta na rede para ler os itens.
- `Query`: recupera todos os itens que têm uma chave de partição específica. Você deve especificar o valor da chave de partição. É possível recuperar itens inteiros, ou apenas um subconjunto dos seus atributos. Se desejar, é possível aplicar uma condição aos valores de chaves de classificação, para recuperar somente um subconjunto dos dados que têm a mesma chave de partição. Essa operação pode ser usada em uma tabela, desde que essa tabela tenha uma chave de partição e uma chave de classificação. Ela também pode ser usada em um índice, desde que esse índice tenha uma chave de partição e uma chave de classificação.
- `Scan`: recupera todos os itens na tabela ou no índice especificado. É possível recuperar itens inteiros, ou apenas um subconjunto dos seus atributos. Opcionalmente, você pode aplicar uma condição de filtragem para retornar apenas os valores de interesse e descartar o restante.

Atualização de dados

- `UpdateItem`: modifica um ou mais atributos em um item. É necessário especificar a chave primária do item que você deseja modificar. É possível adicionar novos atributos e modificar ou remover atributos existentes. Também é possível realizar atualizações condicionais, para que a atualização apenas seja bem-sucedida quando o uma condição definida pelo usuário for atendida. Opcionalmente, você pode implementar um contador atômico, que incrementa ou diminui um atributo numérico sem interferir em outras solicitações de gravação.

Excluir dados

- `DeleteItem`: exclui um único item de uma tabela. É necessário especificar a chave primária do item que você deseja excluir.
- `BatchWriteItem`: exclui até 25 itens de uma ou mais tabelas. Isso é mais eficiente do que chamar `DeleteItem` várias vezes, pois seu aplicativo precisa apenas de uma única viagem de ida e volta na rede para excluir os itens.

> **Note**
>
> Você pode usar `BatchWriteItem` para criar e excluir dados.

DynamoDB Streams

As operações do DynamoDB Streams permitem que você habilite ou desabilite um fluxo em uma tabela e permitem o acesso a registros de modificação de dados contidos em um fluxo.

- `ListStreams`: retorna uma lista de todos os seus fluxos ou somente o fluxo para uma tabela específica.
- `DescribeStream`: retorna informações sobre um fluxo, como seu nome do recurso da Amazon (ARN) e onde sua aplicação pode começar a ler os primeiro registros de fluxo.
- `GetShardIterator`:retorna um iterador de fragmentos, uma estrutura de dados que sua aplicação usa para recuperar os registros de fluxo.
- `GetRecords`: recupera um ou mais registros de fluxo, usando um determinado iterador de fragmentos.

Transações

Transações fornecem atomicidade, consistência, isolamento e durabilidade (ACID), permitindo que você mantenha a exatidão dos dados em suas aplicações com mais facilidade.

Você pode usar o PartiQL: uma linguagem de consultas compatível com SQL para o Amazon DynamoDB para executar operações transacionais ou pode usar as APIs CRUD clássicas do DynamoDB que separam cada operação em uma chamada de API distinta.

PartiQL: uma linguagem de consultas compatível com SQL

- `ExecuteTransaction`: uma operação em lote que permite operações CRUD em vários itens dentro e entre tabelas com um resultado garantido de tudo ou nada.

APIs clássicas

- `TransactWriteItems`: uma operação em lote que permite operações `Put`, `Update` e `Delete` em vários itens dentro e entre tabelas com um resultado garantido de tudo ou nada.
- `TransactGetItems`: uma operação em lote que permite que operações `Get` recuperem vários itens de uma ou mais tabelas.

Tipos de dados compatíveis e regras de nomenclatura no Amazon DynamoDB

Esta seção descreve as regras de nomenclatura do Amazon DynamoDB e os vários tipos de dados compatíveis com o DynamoDB. Há limites que se aplicam a tipos de dados. Para obter mais informações, consulte Tipos de dados.

Tópicos

- Regras de nomenclatura
- Tipos de dados
- Descritores de tipo de dados

Regras de nomenclatura

Tabelas, atributos e outros objetos no DynamoDB devem ter nomes. Os nomes devem ser concisos e significativos; por exemplo, nomes como Products, Books e Authors são autoexplicativos.

Estas são as regras de nomenclatura do DynamoDB:

- Todos os nomes devem ser codificados usando UTF-8 e diferenciam maiúsculas de minúsculas.
- Nomes de tabelas e nomes de índices devem ter entre 3 e 255 caracteres e podem conter somente os seguintes caracteres:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (sublinhado)
  - `-` (traço)

  - . (ponto)
- Os nomes de atributos devem ter pelo menos um caractere e no mínimo 64 KB. Manter os nomes dos atributos o mais curtos possível é considerada uma prática recomendada. Isso ajuda a reduzir as unidades de solicitação de leitura consumidas, pois os nomes dos atributos são incluídos na medição do uso do armazenamento e do throughput.

  Veja as exceções a seguir. Estes nomes de atributo não devem ser maiores do que 255 caracteres:

  - Nomes de chave de partição de índices secundários
  - Nomes de chave de classificação de índices secundários
  - Nomes de qualquer atributo projetado especificado pelo usuário (aplicável apenas a índices secundários locais)

Palavras reservadas e caracteres especiais

O DynamoDB tem uma lista de palavras reservadas e caracteres especiais. Para obter uma lista completa, consulte Palavras reservadas no DynamoDB. Além disso, os seguintes caracteres têm um significado especial no DynamoDB: # (hash) e : (dois-pontos).

Embora o DynamoDB permita usar essas palavras reservadas e caracteres especiais nos nomes, é recomendável evitar, pois será necessário definir variáveis de espaço reservado sempre que esses nomes forem utilizados em uma expressão. Para obter mais informações, consulte Nomes (aliases) de atributo de expressão no DynamoDB.

Tipos de dados

O DynamoDB oferece suporte a vários tipos de dados diferentes para atributos dentro de uma tabela. Eles podem ser categorizados da seguinte maneira:

- Tipos escalares: um tipo escalar pode representar exatamente um valor. Os tipos escalares são número, string, binário, booliano e nulo.
- Tipos de documento: um tipo de documento pode representar uma estrutura complexa com atributos aninhados, como aqueles que você encontraria em um documento JSON. Os tipos de documentos são lista e mapa.
- Tipos de conjuntos: um tipo de conjunto pode representar vários valores escalares. Os tipos de conjuntos são conjunto de strings, conjunto de números e conjunto de binários.

Ao criar uma tabela ou um índice secundário, você deve especificar os nomes e os tipos de dados de cada atributo de chave primária (chave de partição e chave de classificação). Além disso, cada atributo de chave primária deve ser definido como o tipo string, número ou binário.

O DynamoDB é um banco de dados NoSQL sem esquema. Isso significa que, além dos atributos de chave primária, não é necessário definir nenhum atributo nem tipos de dados ao criar tabelas. Por comparação, bancos de dados relacionais exigem que você defina os nomes e os tipos de dados de cada coluna ao criar uma tabela.

As seguintes descrições representam cada tipo de dados, juntamente com exemplos no formato JSON.

Tipos escalares

Os tipos escalares são número, string, binário, booliano e nulo.

Número

Números podem ser positivos, negativo ou zero. Os números podem ter uma precisão de até 38 dígitos. Se esse limite for excedido, uma exceção será gerada. Se precisar de uma precisão maior do que 38 dígitos, você poderá usar strings.

- Intervalo positivo: 1E-130 a 9,99999999999999999999999999999999999999E+125
- Intervalo negativo: -9.99999999999999999999999999999999999999E+125 a -1E-130

No DynamoDB, os números são representados como tamanho variável. Zeros iniciais e finais são cortados.

Todos os números são enviados pela rede ao DynamoDB como strings para maximizar a compatibilidade entre linguagens e bibliotecas. No entanto, o DynamoDB os trata como atributos numéricos para operações matemáticas.

É possível usar o tipo de dados Número para representar uma data ou um carimbo de data/hora. Uma maneira de fazer isso é usando o tempo epoch, o número de segundos desde 00:00:00 UTC em 1.° de janeiro de 1970. Por exemplo, o tempo epoch `1437136300` representa 12:31:40 PM UTC em 17 de julho de 2015.

Para obter mais informações, consulte http://en.wikipedia.org/wiki/Unix_time.

String

Strings são Unicode com codificação binária UTF-8. O tamanho mínimo de uma string poderá ser zero se o atributo não for usado como uma chave para um índice ou uma tabela e ser restrito pelo limite máximo de tamanho de item do DynamoDB de 400 KB.

As seguintes restrições adicionais se aplicam aos atributos de chave primária definidos como string de tipo:

- Para uma chave primária simples, o comprimento máximo do valor do primeiro atributo (a chave de partição) é 2048 bytes.
- Para uma chave primária composta, o comprimento máximo do valor do segundo atributo (a chave de classificação) é 1024 bytes.

O DynamoDB agrupa e compara strings usando os bytes da codificação de strings UTF-8 subjacente. Por exemplo, “a” (0x61) é maior que “A” (0x41) e “¿” (0xC2BF) é maior que “z” (0x7A).

É possível usar o tipo de dados string para representar uma data ou um carimbo de data/hora. Uma maneira de fazer isso é usando strings ISO 8601, conforme mostrado nestes exemplos:

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

Para obter mais informações, consulte http://en.wikipedia.org/wiki/ISO_8601.

> **Note**
>
> Ao contrário dos bancos de dados relacionais convencionais, o DynamoDB não comporta nativamente um tipo de dados de data e hora. Em vez disso, pode ser útil armazenar dados de data e hora como um tipo de dado numérico, usando o horário Unix epoch.

Binário

Atributos do tipo Binário podem armazenar quaisquer dados binários, como texto compactado, dados criptografados ou imagens. Sempre que o DynamoDB compara valores binários, ele trata cada byte dos dados binários como sem sinal.

O tamanho de um atributo binário poderá ser zero, se o atributo não for usado como uma chave para um índice ou uma tabela, e ser restrito pelo limite máximo de tamanho de item do DynamoDB de 400 KB.

Se você definir um atributo de chave primária como um atributo do tipo Binário, as seguintes restrições adicionais serão aplicáveis:

- Para uma chave primária simples, o comprimento máximo do valor do primeiro atributo (a chave de partição) é 2048 bytes.
- Para uma chave primária composta, o comprimento máximo do valor do segundo atributo (a chave de classificação) é 1024 bytes.

Suas aplicações deverão codificar valores binários no formato codificado em base64 antes que eles sejam enviados ao DynamoDB. Após o recebimento desses valores, o DynamoDB decodifica os dados em uma matriz de bytes sem sinal e a utiliza como o comprimento do atributo binário.

O exemplo a seguir é um atributo binário usando texto codificado em base64.

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

Booleano

Um atributo do tipo Booliano pode armazenar `true` ou `false`.

Nulo

Nulo representa um atributo com um estado desconhecido ou indefinido.

Tipos de documentos

Os tipos de documentos são lista e mapa. Esses tipos de dados podem ser aninhados entre si para representar estruturas de dados complexas com até 32 níveis de profundidade.

Não há limite para o número de valores em uma lista ou em um mapa, desde que o item que contém os valores caiba no limite de tamanho de item do DynamoDB (400 KB).

Um valor de atributo poderá ser um valor binário vazio se o atributo não for usado para uma tabela ou chave de índice. Um valor de atributo não pode ser um conjunto vazio (conjunto de strings, conjunto de números e conjunto binário). No entanto, listas e mapas vazios são permitidos. Valores binários e de string vazios são permitidos dentro de listas e mapas. Para obter mais informações, consulte Atributos.

Lista

Um atributo do tipo Lista pode armazenar uma coleção ordenada de valores. Listas são delimitadas por colchetes: `[ ... ]`

Uma lista é semelhante a uma matriz JSON. Não há restrições quanto aos tipos de dados que podem ser armazenados em um elemento de lista, e os elementos de um elemento de lista não precisam ser do mesmo tipo.

O exemplo a seguir mostra uma lista que contém duas strings e um número.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> O DynamoDB permite que você trabalhe com elementos individuais em listas, mesmo que esses elementos estejam profundamente aninhados. Para obter mais informações, consulte Usar expressões no DynamoDB.

Mapa

Um atributo do tipo Mapa pode armazenar uma coleção não ordenada de pares de nome/valor. Mapas são delimitados por chaves: `{ ... }`

Um mapa é semelhante a um objeto JSON. Não há restrições quanto aos tipos de dados que podem ser armazenados em um elemento de mapa, e os elementos de um mapa não precisam ser do mesmo tipo.

Mapas são ideais para armazenar documentos JSON no DynamoDB. O exemplo a seguir mostra um mapa que contém uma string, um número e uma lista aninhada que contém outro mapa.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
```

```
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> O DynamoDB permite que você trabalhe com elementos individuais em mapas, mesmo se esses elementos estiverem profundamente aninhados. Para obter mais informações, consulte Usar expressões no DynamoDB.

Conjuntos

O DynamoDB oferece suporte a tipos que representam conjuntos de valores de número, string ou binário. Todos os elementos de um conjunto devem ser do mesmo tipo. Por exemplo, um conjunto de números só pode conter números, um conjunto de strings pode conter apenas strings.

Não há limite para o número de valores em um conjunto, desde que o item que contém os valores não ultrapasse o limite de tamanho de item do DynamoDB (400 KB).

Cada valor dentro de um conjunto deve ser exclusivo. A ordem dos valores em um conjunto não é preservada. Portanto, seus aplicativos não devem depender de nenhuma ordem específica de elementos no conjunto. O DynamoDB não oferece suporte a conjuntos vazios. No entanto, valores binários e de string vazios são permitidos dentro de um conjunto.

O exemplo a seguir mostra um conjunto de strings, um conjunto de números e um conjunto de binários:

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

Descritores de tipo de dados

O protocolo de API do DynamoDB de baixo nível usa descritores de tipos de dados como tokens que informam ao DynamoDB como interpretar cada atributo.

Veja a seguir uma lista completa dos descritores de tipos de dados do DynamoDB:

- **S** – String
- **N** – Number
- **B** – Binary
- **BOOL** – Boolean
- **NULL** – Null
- **M** – Map
- **L** – List
- **SS** – String Set
- **NS** – Number Set
- **BS** – Binary Set

Classes de tabelas do DynamoDB

O DynamoDB apresenta duas classes de tabela projetadas para ajudar você a otimizar o custo. A classe de tabela Standard do DynamoDB é a padrão e é recomendada para a grande maioria das workloads. A classe de tabela do DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) é otimizada para tabelas em que o armazenamento é o custo dominante. Por exemplo, tabelas que armazenam dados acessados com pouca frequência, como logs da aplicação, postagens antigas de mídia social, histórico de pedidos de comércio eletrônico e antigas conquistas de jogos, são bons candidatos para a classe de tabela Standard-IA. Para obter detalhes de preço, consulte Definição de preço do Amazon DynamoDB.

Cada tabela do DynamoDB está associada a uma classe de tabela (por padrão, DynamoDB Standard). Todos os índices secundários associados à tabela usam a mesma classe de tabela. Cada classe de tabela oferece preços diferentes para armazenamento de dados, bem como para solicitações de leitura e gravação. Você pode selecionar a classe de tabela mais econômica para sua tabela com base nos padrões de uso de armazenamento e do throughput.

A escolha de uma classe de tabela não é permanente — você pode alterar essa configuração usando o AWS Management Console, a CLI do AWS, ou o SDK do AWS. O DynamoDB também permite o gerenciamento de sua classe de tabela usando o AWS CloudFormation para tabelas de região única e tabelas globais. Para saber mais sobre como selecionar sua classe de tabela, consulte Considerações ao escolher uma classe de tabela no DynamoDB.

Partições e distribuição de dados no DynamoDB

O Amazon DynamoDB armazena dados em partições. Uma partição é uma alocação de armazenamento para uma tabela, com suporte de SSDs (unidades de estado sólido) e automaticamente replicada em várias zonas de disponibilidade em uma região da AWS. O gerenciamento de partições é feito inteiramente pelo DynamoDB; você nunca precisará gerenciar partições por conta própria.

Quando você cria uma tabela, seu estado inicial é `CREATING`. Durante essa fase, o DynamoDB aloca partições suficientes para a tabela, para que ela possa lidar com suas necessidades de throughput provisionado. Você pode começar a gravar e ler dados da tabela depois que o status da tabela mudar para `ACTIVE`.

O DynamoDB alocará partições adicionais em uma tabela nas seguintes situações:

- Se você aumentar as configurações de throughput provisionado da tabela além do que as partições existentes podem suportar.
- Se uma partição existente for ocupada até a capacidade máxima e mais espaço de armazenamento for necessário.

O gerenciamento de partições ocorre automaticamente em segundo plano e é transparente para as aplicações. Sua tabela permanece disponível durante todo o processo e oferece suporte total para os seus requisitos de throughput provisionado.

Consulte mais detalhes em Design de chave de partição.

Os índices secundários globais no DynamoDB também são compostos por partições. Os dados em um índice secundário global são armazenados separadamente dos dados em sua tabela-base, mas as partições de índice se comportam da mesma forma que partições de tabela.

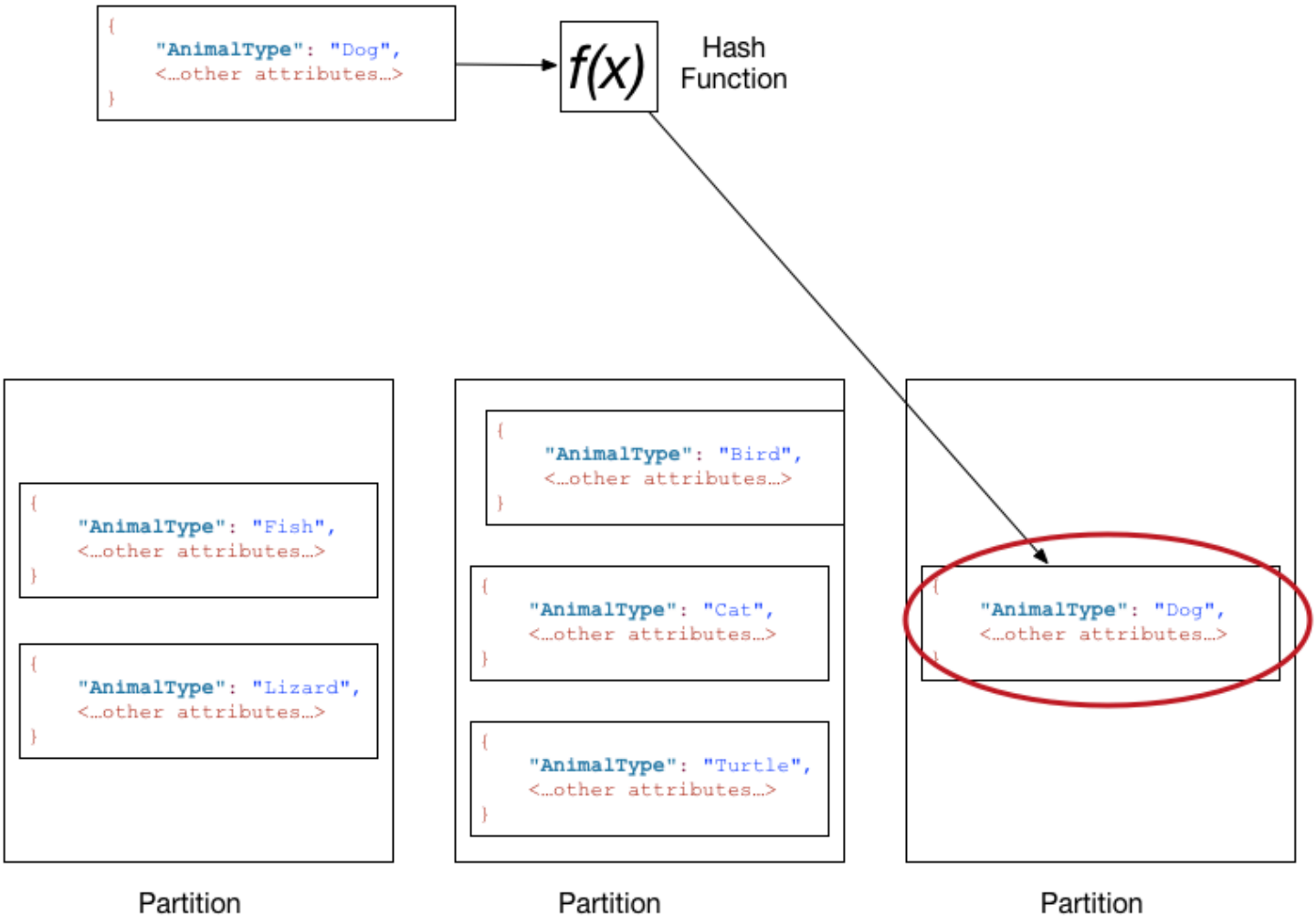
Distribuição de dados: chave de partição

Se a sua tabela tiver uma chave primária simples (somente uma chave de partição), o DynamoDB armazenará e recuperará cada item com base em seu valor de chave de partição.

Para gravar um item na tabela, o DynamoDB usa o valor da chave de partição como entrada para uma função de hash interna. O valor de saída da função de hash determina a partição na qual o item será armazenado.

Para ler um item da tabela, você deve especificar o valor da chave de partição desse item. O DynamoDB usa esse valor como entrada para sua função de hash, resultando na partição na qual o item pode ser encontrado.

O diagrama a seguir mostra uma tabela chamada Pets, que se estende por várias partições. A chave primária da tabela é AnimalType (somente esse atributo de chave é mostrado). O DynamoDB usa sua função de hash para determinar onde armazenar um novo item, neste caso, com base no valor de hash da string Dog. Observe que os itens não são armazenados na ordem classificada. A localização de cada item é determinada pelo valor de hash de sua chave de partição.



> **Note**
>
> O DynamoDB é otimizado para distribuição uniforme de itens entre partições de uma tabela, independentemente de quantas partições possam existir. Recomendamos que você escolha

uma chave de partição que possa ter um grande número de valores distintos em relação ao número de itens da tabela.

Distribuição de dados: chave de partição e chave de classificação

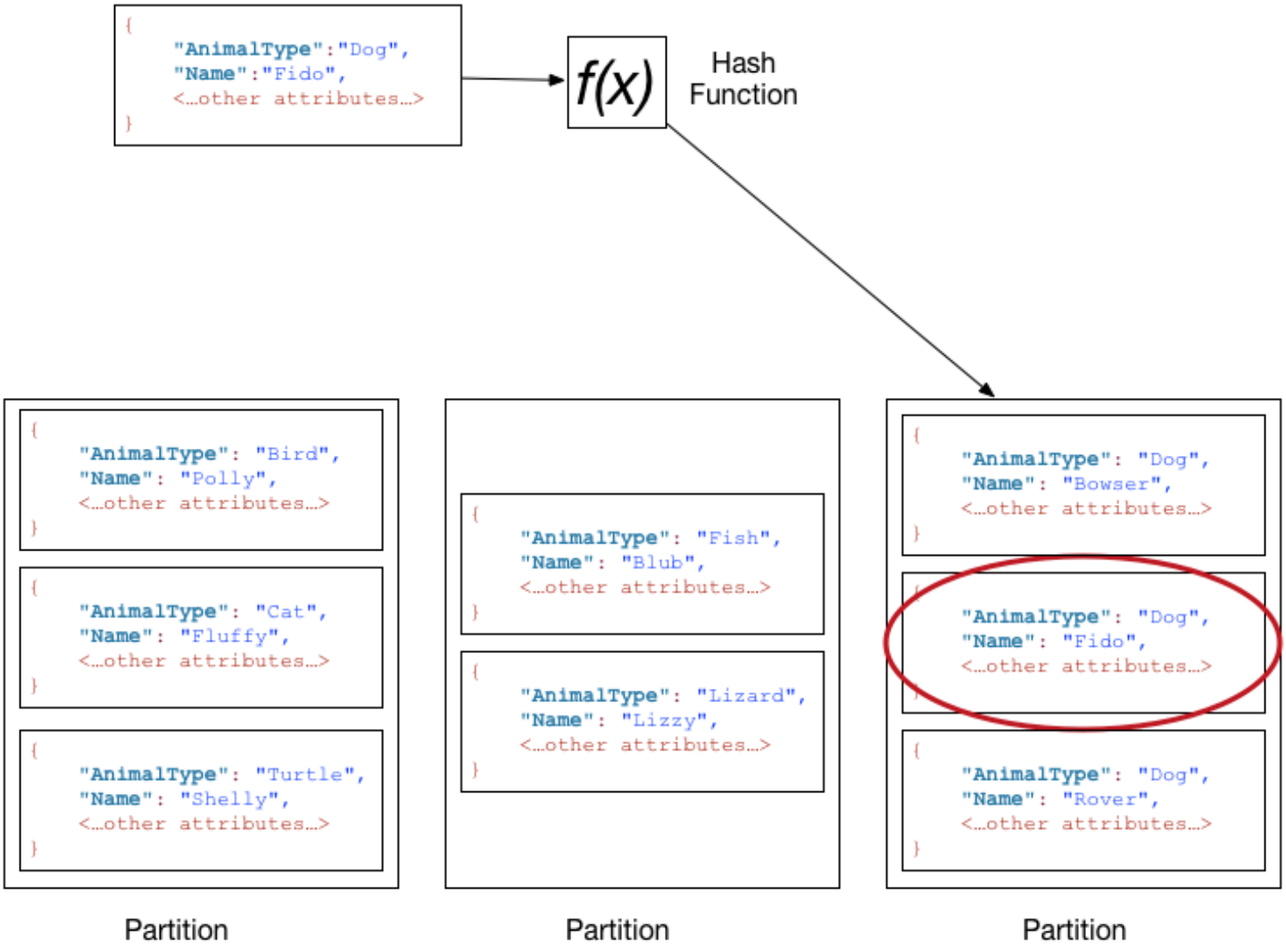
Se a tabela tiver uma chave primária composta (chave de partição e chave de classificação), o DynamoDB calculará o valor de hash da chave de partição da mesma forma que descrito em Distribuição de dados: chave de partição. No entanto, ele tende a manter os itens que têm o mesmo valor de chave de partição próximos e ordenados pelo valor do atributo da chave de classificação. O conjunto de itens que têm o mesmo valor da chave de partição é chamado de coleção de itens. As coleções de itens são otimizadas para a recuperação eficiente de intervalos de itens dentro da coleção. Se sua tabela não tiver índices secundários locais, o DynamoDB dividirá automaticamente sua coleção de itens em quantas partições forem necessárias para armazenar os dados e fornecer o throughput de leitura e gravação.

Para gravar um item na tabela, o DynamoDB calcula o valor de hash da chave de partição a fim de determinar qual partição deve conter esse item. Nessa partição, vários itens poderiam ter o mesmo valor de chave de partição. Portanto, o DynamoDB armazena o item entre os outros com a mesma chave de partição, em ordem ascendente por chave de classificação.

Para ler um item da tabela, você deve especificar seu valor de chave de partição e seu valor de chave de classificação. O DynamoDB calcula o valor de hash da chave de partição, resultando na partição na qual o item pode ser encontrado.

É possível ler vários itens de tabela em uma única operação (`Query`), desde que os itens desejados tenham o mesmo valor de chave de partição. O DynamoDB retorna todos os itens com esse valor de chave de partição. Opcionalmente, você pode aplicar uma condição à chave de classificação para que ela retorne somente os itens em um determinado intervalo de valores.

Suponhamos que a tabela Pets possua uma chave primária composta que consiste em AnimalType (chave de partição) e Name (chave de classificação). O diagrama a seguir mostra o DynamoDB gravando um item com um valor de chave de partição Dog e um valor de chave de classificação Fido.



Para ler esse mesmo item na tabela Pets, o DynamoDB calcula o valor de hash de Dog, gerando assim a partição na qual esses itens estão armazenados. Em seguida, o DynamoDB verifica os valores de atributo de chave de classificação até encontrar Fido.

Para ler todos os itens com um AnimalType de Dog, você pode emitir uma operação `Query` sem especificar uma condição de chave de classificação. Por padrão, os itens são retornados na ordem em que são armazenados (ou seja, em ordem crescente por chave de classificação). Existe a opção de solicitar a ordem decrescente em vez disso.

Para consultar somente alguns itens de Dog, você pode aplicar uma condição à chave de classificação (por exemplo, apenas os itens Dog em que Name comece com uma letra que esteja dentro do intervalo de A a K).

> **Note**
>
> Em uma tabela do DynamoDB, não há limite superior quanto ao número de valores de chave de classificação distintos por valor de chave de partição. Se for necessário armazenar bilhões de itens Dog na tabela Pets, o DynamoDB alocará automaticamente armazenamento suficiente para atender a esse requisito.

Saiba como passar do SQL para o NoSQL

Se você é um desenvolvedor de aplicativos, talvez tenha alguma experiência no uso do sistema de gerenciamento de banco de dados relacional (RDBMS) e Structured Query Language (SQL). Ao começar a trabalhar com o Amazon DynamoDB, você encontrará muita similaridades, mas também muitas diferenças. NoSQL é um termo usado para descrever os sistemas de bancos de dados não relacionais altamente disponíveis, dimensionáveis e otimizados para alta performance. Em vez de usar o modelo relacional, os bancos de dados NoSQL (como o DynamoDB) usam modelos alternativos para o gerenciamento de dados, como pares de chave-valor ou armazenamento de documentos. Para obter mais informações, consulte O que é NoSQL?.

Oferece suporte a PartiQL do Amazon DynamoDB, uma linguagem de consulta compatível com SQL de código aberto que facilita a consulta de dados com eficiência, independentemente do local ou do formato em que estejam armazenados. Com PartiQL, é possível processar facilmente dados estruturados de bancos de dados relacionais, dados semiestruturados e aninhados em formatos de dados abertos e até mesmo dados sem esquema no NoSQL ou em bancos de dados de documentos que permitam atributos diferentes para linhas diferentes. Para obter mais informações, consulte Linguagem de consultas PartiQL.

As seções a seguir descrevem tarefas comuns de banco de dados, comparando e contrastando instruções SQL com as operações equivalentes do DynamoDB.

> **Note**
>
> Os exemplos de SQL nesta seção são compatíveis com o RDBMS MySQL.
> Os exemplos do DynamoDB desta seção mostram o nome da operação do DynamoDB, junto com os parâmetros dessa operação no formato JSON.

Tópicos

- Escolher entre relacional (SQL) e NoSQL
- Diferenças no acesso a um banco de dados relacional (SQL) e ao DynamoDB
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao criar uma tabela
- Diferenças entre obter informações de tabela de um banco de dados relacional (SQL) e do DynamoDB
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao gravar dados em uma tabela
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao ler dados de uma tabela
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao gerenciar índices
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao modificar dados de uma tabela
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao excluir dados de uma tabela
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao remover uma tabela

Escolher entre relacional (SQL) e NoSQL

Atualmente, os aplicativos têm exigências maiores do que nunca. Por exemplo, um jogo online pode começar com apenas alguns usuários e uma pequena quantidade de dados. No entanto, se o jogo obtiver sucesso, ele poderá facilmente superar os recursos do sistema de gerenciamento de banco de dados subjacente. É comum que aplicativos baseados na web tenham centenas, milhares ou milhões de usuários simultâneos, com terabytes ou mais de novos dados gerados por dia. Os bancos de dados para tais aplicativos devem lidar com dezenas (ou centenas) de milhares de leituras e gravações por segundo.

O Amazon DynamoDB é perfeitamente adequado para esses tipo de workloads. Como um desenvolvedor, você pode começar pequeno e aumentar gradualmente sua utilização conforme a aplicação se tornar mais popular. O DynamoDB escala perfeitamente para lidar com quantidades muito grandes de dados e um número muito grande de usuários.

Para obter mais informações sobre a modelagem tradicional de banco de dados relacional e como adaptá-la para o DynamoDB, consulte Práticas recomendadas para modelagem de dados relacionais no DynamoDB.

A tabela a seguir mostra algumas diferenças gerais entre um sistema de gerenciamento de banco de dados relacional (RDBMS) e o DynamoDB.

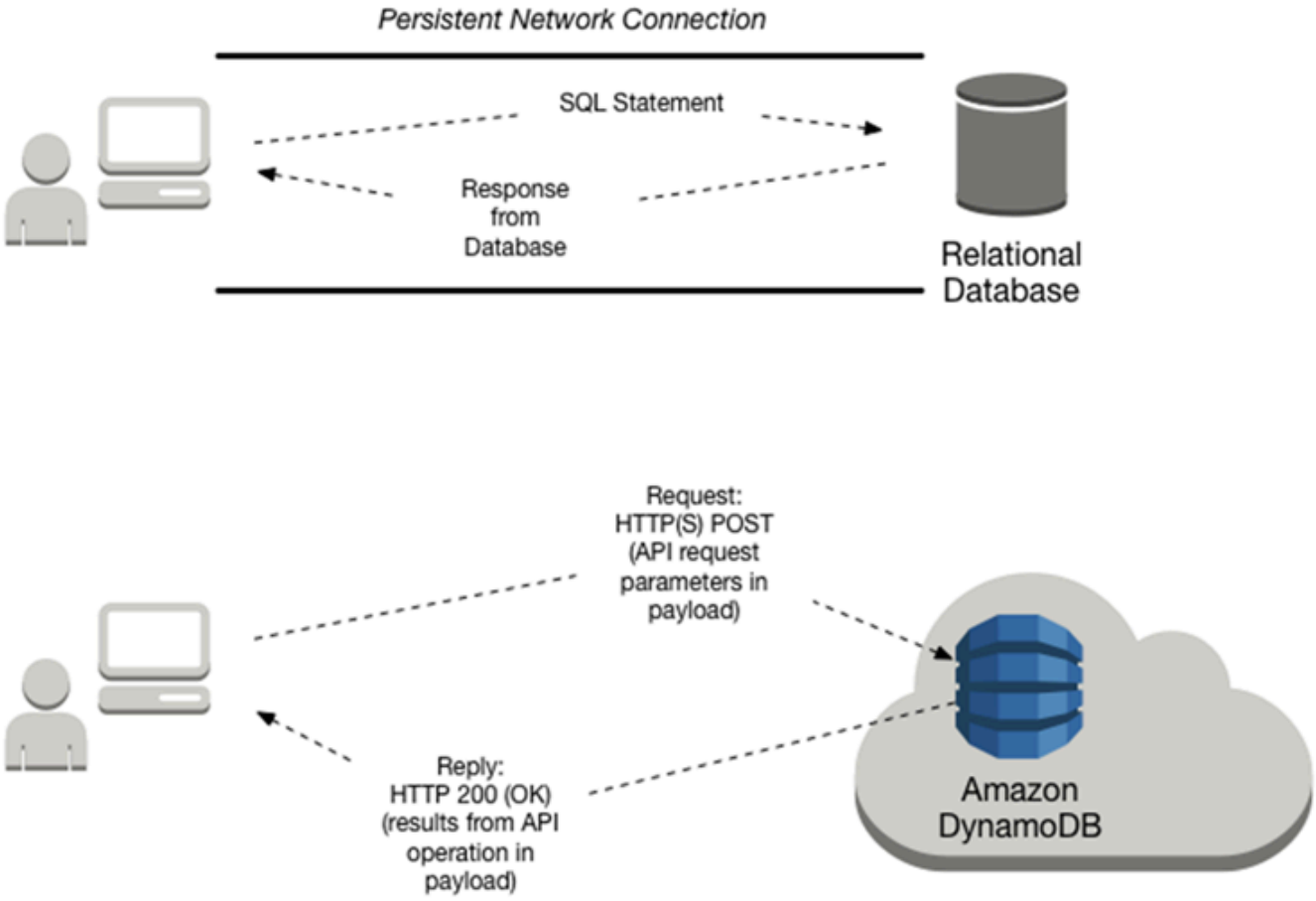
Característica	Sistema de gerenciamento de banco de dados relacional (RDBMS)	Amazon DynamoDB
Workloads ideais	Consultas ad hoc; data warehouse; OLAP (processa mento analítico online).	Aplicativos em escala da web, incluindo redes sociais, jogos, compartilhamento de mídia e Internet das Coisas (IoT).
Modelo de dados	O modelo relacional exige um esquema bem definido, onde os dados são normalizados em tabelas, linhas e colunas. Além disso, todos os relaciona mentos são definidos entre tabelas, colunas, índices e outros elementos de banco de dados.	O DynamoDB é sem esquema. Cada tabela deve ter uma chave primária para identificar exclusivamente cada item de dados, mas não há restrições semelhant es em outros atributos não chave. O DynamoDB pode gerenciar dados estruturados ou semiestruturados, incluindo documentos JSON.
Acesso aos dados	SQL é o padrão para armazenar e recuperar dados. Os bancos de dados relaciona is oferecem um conjunto completo de ferramentas para simplificar o desenvolvimento de aplicativos para banco de dados, mas todas essas ferramentas usam o SQL.	Você pode usar o AWS Management Console, a AWS CLI ou o NoSQL WorkBench para trabalhar com o DynamoDB e executar tarefas ad hoc. A linguagem PartiQL, uma linguagem de consultas compatível com SQL, permite selecionar, inserir, atualizar e excluir dados no DynamoDB. As aplicações podem utilizar os kits de desenvolvimento de software (SDKs) da AWS para trabalhar com o DynamoDB usando interfaces baseadas

Característica	Sistema de gerenciamento de banco de dados relacional (RDBMS)	Amazon DynamoDB
		em objeto, centradas em documentos ou de baixo nível.
Desempenho	Os bancos de dados relacionais são otimizados para armazenamento, portanto, a performance geralmente depende do subsistema de disco. Os desenvolvedores e os administradores de banco de dados devem otimizar as consultas, os índices e as estruturas de tabela para obter a máxima performance.	O DynamoDB é otimizado para computação, portanto, a performance é principalmente uma função do hardware subjacente e da latência de rede. Como um serviço gerenciado, o DynamoDB protege você e suas aplicações contra esses detalhes de implementação para que você possa se concentrar em projetar e construir aplicações robustas e de alta performance.
Escalabilidade	É mais fácil aumentar a escala com hardware mais rápido. Também é possível que as tabelas de banco de dados se expandam em vários hosts em um sistema distribuído, mas isso exige investimentos adicionais. Os bancos de dados relacionais têm tamanhos máximos para o número e o tamanho dos arquivos, o que impõe limites superiores na escalabilidade.	O DynamoDB é projetado para aumentar a escala horizontalmente usando clusters distribuídos de hardware. Esse design permite aumentar o throughput sem aumentar a latência. Os clientes especificam seus requisitos de throughput e o DynamoDB aloca recursos suficientes para atender a esses requisitos. Não há limites superiores no número de itens por tabela, nem no tamanho total da tabela.

Diferenças no acesso a um banco de dados relacional (SQL) e ao DynamoDB

Para que seu aplicativo possa acessar um banco de dados, ele deve ser autenticado para garantir que o aplicativo tenha permissão para usar o banco de dados. Ele deve ser autorizado para que o aplicativo possa realizar somente as ações para as quais tem permissões.

O diagrama a seguir mostra a interação do cliente com um banco de dados relacional e com o Amazon DynamoDB.



A tabela a seguir tem mais detalhes sobre tarefas de interação com o cliente.

Característica	Sistema de gerenciamento de banco de dados relacional (RDBMS)	Amazon DynamoDB
Ferramentas para acessar o banco de dados	A maioria dos bancos de dados relacionais fornecem uma interface de linha de comando (CLI), para que você possa inserir instruções SQL ad hoc e visualizar os resultados imediatamente.	Na maioria dos casos, você grava o código da aplicação . Você também pode usar o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou o NoSQL Workbench para enviar solicitações ad hoc ao DynamoDB e visualizar os resultados. A linguagem PartiQL, uma linguagem de consultas compatível com SQL, permite selecionar, inserir, atualizar e excluir dados no DynamoDB.
Conexão ao banco de dados	Um programa de aplicação estabelece e mantém uma conexão de rede com o banco de dados. Quando o aplicativo terminar, ele encerra a conexão.	O DynamoDB é um serviço da Web, e as interações com ele são sem estado. Os aplicativos não precisam manter conexões de rede persistentes. Em vez disso, a interação com o DynamoDB ocorre via solicitações e respostas HTTP(S).
Autenticação	Uma aplicação não pode se conectar ao banco de dados até ser autenticado. O RDBMS pode realizar a autenticação em si, ou pode passar essa tarefa para o	Cada solicitação para o DynamoDB deve ser acompanhada por uma assinatura de criptografia, que autentica essa determina da solicitação. Os AWS SDKs fornecem toda a lógica

Característica	Sistema de gerenciamento de banco de dados relacional (RDBMS)	Amazon DynamoDB
	sistema operacional host ou um serviço de diretório.	necessária para a criação de assinaturas e a assinatura de solicitações. Para obter mais informações, consulte Assinar solicitações de API da AWS no Referência geral da AWS.
Autorização	Os aplicativos podem executar somente as ações para as quais tenham sido autorizados. Os administradores de banco de dados ou os proprietários de aplicativos podem usar as instruções SQL `GRANT` e `REVOKE` para controlar o acesso aos objetos de banco de dados (tais como tabelas), dados (como linhas em uma tabela), ou a habilidade de emitir determinadas instruções SQL.	No DynamoDB, a autorização é controlada pelo AWS Identity and Access Management (IAM). Você pode escrever uma política do IAM para conceder permissões em um recurso do DynamoDB (como uma tabela), depois permitir que os usuários e os perfis usem essa política. O IAM também oferece controle de acesso minucioso para itens de dados individuais em tabelas do DynamoDB. Para obter mais informações, consulte Gerenciamento de identidade e acesso no Amazon DynamoDB.

Característica	Sistema de gerenciamento de banco de dados relacional (RDBMS)	Amazon DynamoDB
Envio de uma solicitação	O aplicativo emite uma instrução SQL para cada operação de banco de dados que ele deseja executar. Após o recebimento da instrução SQL, o RDBMS verifica a sintaxe, cria um plano para realizar a operação e, em seguida, executa o plano.	A aplicação envia solicitações HTTP(S) para o DynamoDB. As solicitações contêm o nome da operação do DynamoDB a ser executada , juntamente com parâmetro s. O DynamoDB executa a solicitação imediatamente.
Recebimento de uma resposta	O RDBMS retorna os resultados da instrução SQL. Se houver um erro, o RDBMS retorna um status e uma mensagem de erro.	O DynamoDB retorna uma resposta HTTP(S) contendo os resultados da operação. Se houver um erro, o DynamoDB retornará um status e mensagens de erro HTTP.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao criar uma tabela

As tabelas são as estruturas de dados fundamentais em bancos de dados relacionais e no Amazon DynamoDB. Um Relational Database Management System (RDBMS – Sistema de gerenciamento de banco de dados relacional) exige que você defina o esquema da tabela ao criá-la. Por outro lado, as tabelas do DynamoDB não têm esquemas: além da chave primária, não é necessário definir nenhum atributo ou tipo de dados extras ao criar uma tabela.

A seção a seguir compara como você criaria uma tabela com o SQL e como a criaria com o DynamoDB.

Tópicos

- Criar uma tabela com o SQL

- Criar uma tabela com o DynamoDB

Criar uma tabela com o SQL

Com o SQL, você usa a instrução `CREATE TABLE` para criar uma tabela, conforme mostrado no exemplo a seguir.

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

A chave primária dessa tabela consiste em Artist e SongTitle.

Você deve definir todas as colunas e os tipos de dados da tabela, e a chave primária da tabela. (Você pode usar a instrução `ALTER TABLE` para alterar essas definições mais tarde, se necessário.)

Muitas implementações de SQL permitem que você defina especificações de armazenamento para a sua tabela, como parte da instrução `CREATE TABLE`. A não ser que você indique de outra forma, a tabela é criada com configurações de armazenamento padrão. Em um ambiente de produção, um administrador de banco de dados pode ajudar a determinar os parâmetros de armazenamento ideais.

Criar uma tabela com o DynamoDB

Use a operação `CreateTable` para criar uma tabela de modo provisionado, especificando parâmetros conforme mostrado abaixo:

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH" //Partition key
```

```
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
    ],
    ProvisionedThroughput: {       // Only specified if using provisioned mode
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

A chave primária dessa tabela consiste em Artist (chave de partição) e SongTitle (chave de classificação).

Você deve fornecer os parâmetros a seguir para `CreateTable`:

- `TableName`: nome da tabela.
- `KeySchema`: os atributos que são usados para a chave primária. Para obter mais informações, consulte Tabelas, itens e atributos e Chave primária.
- `AttributeDefinitions`: os tipos de dados dos atributos de esquema de chaves.
- `ProvisionedThroughput (for provisioned tables)`: o número de leituras e gravações por segundo que você precisa para esta tabela. O DynamoDB reserva recursos do sistema e de armazenamento suficientes para que seus requisitos de throughput sejam sempre atendidos. Você pode usar a operação `UpdateTable` para alterar essas configurações mais tarde, se necessário. Você não precisa especificar os requisitos de armazenamento de uma tabela porque a alocação de armazenamento é gerenciada inteiramente pelo DynamoDB.

Diferenças entre obter informações de tabela de um banco de dados relacional (SQL) e do DynamoDB

Você pode verificar se uma tabela foi criada de acordo com suas especificações. Em um banco de dados relacional, todo o esquema da tabela é mostrado. As tabelas do Amazon DynamoDB não têm esquemas, portanto, somente os atributos de chave primária são mostrados.

Tópicos

- Obter informações sobre uma tabela com o SQL
- Obter informações sobre uma tabela no DynamoDB

Obter informações sobre uma tabela com o SQL

A maioria dos sistemas de gerenciamento de bancos de dados relacionais (RDBMS) permitem que você descreva a estrutura de uma tabela – colunas, tipos de dados, definição de chave primária, e assim por diante. Não há nenhuma maneira padrão de fazer isso no SQL. No entanto, muitos sistemas de banco de dados fornecem um comando `DESCRIBE`. Veja a seguir um exemplo do MySQL.

```
DESCRIBE Music;
```

Isso retorna a estrutura de sua tabela, com todos os nomes de colunas, tipos de dados e tamanhos.

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Artist     | varchar(20) | NO   | PRI | NULL    |       |
| SongTitle  | varchar(30) | NO   | PRI | NULL    |       |
| AlbumTitle | varchar(25) | YES  |     | NULL    |       |
| Year       | int(11)     | YES  |     | NULL    |       |
| Price      | float       | YES  |     | NULL    |       |
| Genre      | varchar(10) | YES  |     | NULL    |       |
| Tags       | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

A chave primária dessa tabela consiste em Artist e SongTitle.

Obter informações sobre uma tabela no DynamoDB

O DynamoDB tem uma operação `DescribeTable`, que é semelhante. O único parâmetro é o nome da tabela.

```
{
    TableName : "Music"
}
```

A resposta de `DescribeTable` é semelhante ao seguinte.

```
{
  "Table": {
    "AttributeDefinitions": [
      {
        "AttributeName": "Artist",
        "AttributeType": "S"
      },
      {
        "AttributeName": "SongTitle",
        "AttributeType": "S"
      }
    ],
    "TableName": "Music",
    "KeySchema": [
      {
        "AttributeName": "Artist",
        "KeyType": "HASH"  //Partition key
      },
      {
        "AttributeName": "SongTitle",
        "KeyType": "RANGE"  //Sort key
      }
    ],

    ...
```

`DescribeTable` também retorna informações sobre índices na tabela, configurações de throughput provisionado, uma contagem aproximada de itens e outros metadados.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao gravar dados em uma tabela

As tabelas de bancos de dados relacionais contêm linhas de dados. As linhas são compostas de colunas. As tabelas do Amazon DynamoDB contêm itens. Os itens são compostos de atributos.

Esta seção descreve como gravar uma linha (ou item) em uma tabela.

Tópicos

- Gravar dados em uma tabela com o SQL
- Gravar dados em uma tabela no DynamoDB

Gravar dados em uma tabela com o SQL

A tabela em um banco de dados relacional é uma estrutura de dados bidimensional composta por linhas e colunas. Alguns sistemas de gerenciamento de banco de dados também fornecem suporte para dados semiestruturados, geralmente com tipos de dados nativos JSON ou XML. No entanto, os detalhes de implementação variam entre fornecedores.

No SQL, você usa a instrução `INSERT` para adicionar uma linha a uma tabela.

```
INSERT INTO Music
    (Artist, SongTitle, AlbumTitle,
    Year, Price, Genre,
    Tags)
VALUES(
    'No One You Know', 'Call Me Today', 'Somewhat Famous',
    2015, 2.14, 'Country',
    '{"Composers": ["Smith", "Jones", "Davis"],"LengthInSeconds": 214}'
);
```

A chave primária dessa tabela consiste em Artist e SongTitle. Você deve especificar valores para essas colunas.

> **Note**
>
> Este exemplo usa a coluna Tags para armazenar dados semiestruturados sobre as músicas na tabela Music. A coluna Tags está definida como tipo TEXT, que pode armazenar até 65.535 caracteres em MySQL.

Gravar dados em uma tabela no DynamoDB

No Amazon DynamoDB, é possível usar a API do DynamoDB ou do PartiQL (uma linguagem de consultas compatível com SQL) para adicionar um item a uma tabela.

DynamoDB API

Com a API do DynamoDB, você usa a operação `PutItem` para adicionar um item a uma tabela.

```
{
    TableName: "Music",
    Item: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today",
        "AlbumTitle":"Somewhat Famous",
        "Year": 2015,
        "Price": 2.14,
        "Genre": "Country",
        "Tags": {
            "Composers": [
                  "Smith",
                  "Jones",
                  "Davis"
            ],
            "LengthInSeconds": 214
        }
    }
}
```

A chave primária dessa tabela consiste em Artist e SongTitle. Você deve especificar valores para esses atributos.

Aqui estão algumas coisas importantes para saber sobre este exemplo de `PutItem`:

- O DynamoDB oferece suporte nativo a documentos usando JSON. Isso torna o DynamoDB ideal para armazenar dados semiestruturados, como Tags. Você também pode recuperar e manipular dados de dentro de documentos JSON.
- A tabela Music não tem atributos predefinidos, além da chave primária (Artist e SongTitle).
- A maioria dos bancos de dados SQL são orientados a transação. Quando você emite uma instrução `INSERT`, as modificações de dados não são permanentes até que você execute uma instrução `COMMIT`. Com o Amazon DynamoDB, os efeitos de uma operação `PutItem` são permanentes quando o DynamoDB responde com um código de status HTTP 200 (0K).

A seguir há alguns outros exemplos de `PutItem`.

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "My Dog Spot",
        "AlbumTitle":"Hey Now",
        "Price": 1.98,
        "Genre": "Country",
        "CriticRating": 8.4
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "Somewhere Down The Road",
        "AlbumTitle":"Somewhat Famous",
        "Genre": "Country",
        "CriticRating": 8.4,
        "Year": 1984
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
```

```
        "SongTitle": "Still In Love",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 2.47,
        "Genre": "Rock",
        "PromotionInfo": {
            "RadioStationsPlaying":[
                 "KHCR", "KBQX", "WTNR", "WJJH"
            ],
            "TourDates": {
                "Seattle": "20150625",
                "Cleveland": "20150630"
            },
            "Rotation": "Heavy"
        }
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Look Out, World",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 0.99,
        "Genre": "Rock"
    }
}
```

> **Note**
>
> Além de `PutItem`, o DynamoDB comporta uma operação `BatchWriteItem` para gravar vários itens ao mesmo tempo.

PartiQL for DynamoDB

Com PartiQL, você usa a operação `ExecuteStatement` para adicionar um item a uma tabela, usando a instrução `Insert` PartiQL.

```
INSERT into Music value {
    'Artist': 'No One You Know',
```

```
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

A chave primária dessa tabela consiste em Artist e SongTitle. Você deve especificar valores para esses atributos.

> **Note**
>
> Para obter exemplos de código que usam `Insert` e `ExecuteStatement`, consulte Instruções Insert em PartiQL para DynamoDB.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao ler dados de uma tabela

Com o SQL, você usa a instrução `SELECT` para recuperar uma ou mais linhas de uma tabela. Use a cláusula `WHERE` para determinar os dados que são retornados para você.

Esse recurso é diferente ao usar o Amazon DynamoDB, que oferece as seguintes operações para leitura de dados:

- `ExecuteStatement` recupera um ou vários itens de uma tabela. `BatchExecuteStatement` recupera vários itens de tabelas diferentes em uma única operação. Ambas as operações usam PartiQL, uma linguagem de consulta compatível com SQL.
- `GetItem`: recupera um único item de uma tabela. Esta é a maneira mais eficiente de ler um único item, pois ele dá acesso direto à localização física do item. (O DynamoDB também oferece a operação `BatchGetItem`, permitindo que você realize até 100 chamadas `GetItem` em uma única operação.)
- `Query`: recupera todos os itens que têm uma chave de partição específica. Nesses itens, você pode aplicar uma condição à chave de classificação e recuperar apenas um subconjunto dos dados. `Query` fornece acesso rápido e eficiente às partições em que os dados são armazenados. (Para ter mais informações, consulte Partições e distribuição de dados no DynamoDB.)
- `Scan`: recupera todos os itens da tabela especificada. (Essa operação não deve ser usada com tabelas grandes, pois ela pode consumir grandes quantidades de recursos do sistema.)

> **Note**
>
> Com um banco de dados relacional, você pode usar a instrução SELECT para juntar os dados de várias tabelas e retornar os resultados. Junções são fundamentais para o modelo relacional. Para garantir que as junções sejam executadas de forma eficiente, o banco de dados e suas aplicações devem ter a performance ajustada de forma contínua. O DynamoDB é um banco de dados NoSQL não relacional que não é compatível com junções de tabelas. Em vez disso, os aplicativos leem dados de uma tabela por vez.

As seções a seguir descrevem diferentes casos de uso para a leitura de dados e como executar essas tarefas com um banco de dados relacional e com o DynamoDB.

Tópicos

- Diferenças na leitura de um item usando a respectiva chave primária
- Diferenças na consulta de uma tabela
- Diferenças na varredura de uma tabela

Diferenças na leitura de um item usando a respectiva chave primária

Um padrão de acesso comuns para bancos de dados é ler um único item de uma tabela. Você deve especificar a chave primária do item que deseja.

Tópicos

- Ler um item usando a respectiva chave primária com o SQL
- Ler um item usando a respectiva chave primária no DynamoDB

Ler um item usando a respectiva chave primária com o SQL

No SQL, você usa a instrução SELECT para recuperar dados de uma tabela. Você pode solicitar uma ou mais colunas no resultado (ou todas elas, se você usar o operador *). A cláusula WHERE determina quais linhas devem ser retornadas.

A seguinte é uma instrução SELECT para recuperar uma única linha da tabela Music. A cláusula WHERE especifica os valores de chave primária.

```
SELECT *
```

```
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

É possível modificar a consulta para recuperar somente um subconjunto de colunas.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Observe que a chave primária dessa tabela consiste em Artist e SongTitle.

Ler um item usando a respectiva chave primária no DynamoDB

No Amazon DynamoDB, você pode usar a API do DynamoDB ou do PartiQL (uma linguagem de consultas compatível com SQL) para ler um item de uma tabela.

DynamoDB API

Com a API do DynamoDB, você usa a operação `PutItem` para adicionar um item a uma tabela.

O DynamoDB fornece a operação `GetItem` para recuperar um item por meio da respectiva chave primária. `GetItem` é altamente eficiente, pois dá acesso direto à localização física do item. (Para ter mais informações, consulte Partições e distribuição de dados no DynamoDB.)

Por padrão, `GetItem` retorna todo o item com todos os seus atributos.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

É possível adicionar um parâmetro `ProjectionExpression` para retornar apenas alguns dos atributos.

```
{
    TableName: "Music",
    Key: {
```

```
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

Observe que a chave primária dessa tabela consiste em Artist e SongTitle.

A operação `GetItem` do DynamoDB é muito eficiente. Ela usa os valores da chave primária para determinar o local de armazenamento exato do item em questão e recupera-o diretamente dali. A instrução SQL `SELECT` é similarmente eficiente, no caso da recuperação de itens por valores de chave primária.

A instrução SQL `SELECT` é compatível com muitos tipos de consultas e verificações de tabela. O DynamoDB oferece funcionalidade semelhante com suas operações `Query` e `Scan`, que são descritas em Diferenças na consulta de uma tabela e Diferenças na varredura de uma tabela.

A instrução SQL `SELECT` pode realizar junções de tabela, permitindo que você recupere dados de várias tabelas ao mesmo tempo. As junções são mais eficazes onde as tabelas de banco de dados são normalizadas e os relacionamentos entre as tabelas são claros. No entanto, se você juntar muitas tabelas em uma instrução `SELECT`, a performance do aplicativo pode ser afetada. Você pode resolver esses problemas usando replicação de banco de dados, visualizações materializadas ou regravações de consulta.

O DynamoDB é um banco de dados não relacional e não oferece suporte a junções de tabela. Se você estiver migrando uma aplicação existente de um banco de dados relacional para o DynamoDB, será necessário desnormalizar o seu modelo de dados para eliminar a necessidade de junções.

PartiQL for DynamoDB

Com PartiQL, você usa a operação `ExecuteStatement` para ler um item de uma tabela, usando a instrução `Select` PartiQL.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Observe que a chave primária dessa tabela consiste em Artist e SongTitle.

> **Note**
>
> A instrução Select PartiQL também pode ser usada para consultar ou digitalizar uma tabela do DynamoDB

Para obter exemplos de código que usam `Select` e `ExecuteStatement`, consulte Instruções Select em PartiQL para DynamoDB.

Diferenças na consulta de uma tabela

Outro padrão de acesso comum é a leitura de vários itens de uma tabela, com base em seus critérios de consulta.

Tópicos

- Criar uma tabela com o SQL
- Consultar uma tabela no DynamoDB

Criar uma tabela com o SQL

O uso da instrução `SELECT` do SQL permite consultar colunas-chave, colunas que não são chave ou qualquer combinação. A cláusula `WHERE` determina quais linhas são retornadas, conforme mostrado nos exemplos a seguir.

```
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
/* Return all of the songs by an artist */

SELECT * FROM Music
WHERE Artist='No One You Know';
```

```
/* Return all of the songs by an artist, matching first part of title */

SELECT * FROM Music
```

```
WHERE Artist='No One You Know' AND SongTitle LIKE 'Call%';
```

```
/* Return all of the songs by an artist, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

Observe que a chave primária dessa tabela consiste em Artist e SongTitle.

Consultar uma tabela no DynamoDB

No Amazon DynamoDB, você pode usar a API do DynamoDB ou do PartiQL (uma linguagem de consultas compatível com SQL) para consultar um item de uma tabela.

DynamoDB API

Com o Amazon DynamoDB, você pode usar a operação `Query` para recuperar dados de forma semelhante. A operação `Query` oferece acesso rápido e eficiente aos locais físicos onde os dados estão armazenados. Para obter mais informações, consulte Partições e distribuição de dados no DynamoDB.

É possível usar a `Query` com qualquer tabela ou índice secundário. É possível especificar uma condição de igualdade para o valor da chave de partição e, se desejar, fornecer outra condição, se definida, ao atributo da chave de classificação.

O parâmetro `KeyConditionExpression` especifica os valores de chave que você deseja consultar. Você pode usar uma `FilterExpression` opcional para remover determinados itens dos resultados antes que eles sejam retornados para você.

No DynamoDB, você deve usar `ExpressionAttributeValues` como espaços reservados em parâmetros de expressão (como `KeyConditionExpression` e `FilterExpression`). Isso é análogo ao uso de variáveis de ligação em bancos de dados relacionais, onde você substitui os valores reais na instrução `SELECT` em tempo de execução.

Observe que a chave primária dessa tabela consiste em Artist e SongTitle.

A seguir há alguns exemplos de `Query` no DynamoDB.

```
// Return a single song, by primary key
```

```
{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

```
// Return all of the songs by an artist

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    ExpressionAttributeValues: {
        ":a": "No One You Know"
    }
}
```

```
// Return all of the songs by an artist, matching first part of title

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call"
    }
}
```

PartiQL for DynamoDB

Com o PartiQL, é possível realizar uma consulta usando a operação `ExecuteStatement` e a instrução `Select` na chave de partição.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

Usar a instrução SELECT desse modo retornará todas as músicas associadas a este `Artist` específico.

Para obter exemplos de código que usam `Select` e `ExecuteStatement`, consulte Instruções Select em PartiQL para DynamoDB.

Diferenças na varredura de uma tabela

No SQL, uma instrução `SELECT` sem uma cláusula `WHERE` retornará cada linha em uma tabela. No Amazon DynamoDB, a operação `Scan` faz a mesma coisa. Em ambos os casos, é possível recuperar todos os itens ou somente alguns deles.

Se você estiver usando um banco de dados SQL ou um NoSQL, as verificações devem ser usadas com moderação, pois elas podem consumir grandes quantidades de recursos do sistema. Às vezes, uma verificação é apropriada (como a verificação de uma pequena tabela) ou inevitável (como a execução de uma exportação de dados em massa). No entanto, como uma regra geral, você deve projetar seus aplicativos para evitar a execução de verificações. Para obter mais informações, consulte Consultar tabelas no DynamoDB.

> **Note**
>
> Fazer uma exportação em massa também cria pelo menos um arquivo por partição. Todos os itens em cada arquivo são do espaço de chave com hash dessa partição específica.

Tópicos

- Verificar uma tabela com o SQL
- Verificar uma tabela no DynamoDB

Verificar uma tabela com o SQL

Ao usar o SQL, você pode verificar uma tabela e recuperar todos os respectivos dados usando uma instrução `SELECT` sem especificar uma cláusula `WHERE`. Você pode solicitar uma ou mais colunas no resultado. Ou é possível solicitar todas elas se usar o caractere curinga (*).

Veja a seguir exemplos do uso de uma instrução `SELECT`.

```
/* Return all of the data in the table */
SELECT * FROM Music;
```

```
/* Return all of the values for Artist and Title */
```

```
SELECT Artist, Title FROM Music;
```

Verificar uma tabela no DynamoDB

No Amazon DynamoDB, você pode usar a API do DynamoDB ou do PartiQL (uma linguagem de consultas compatível com SQL) para executar uma operação de verificação em uma tabela.

DynamoDB API

Com a API do DynamoDB, use a operação `Scan` para retornar um ou mais itens e atributos de item ao acessar cada item em uma tabela ou um índice secundário.

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

A operação `Scan` também oferece um parâmetro `FilterExpression`, que pode ser usado para descartar itens que você não deseja que sejam exibidos nos resultados. Uma `FilterExpression` é aplicada após a verificação ser realizada, mas antes que os resultados sejam retornados para você. (Isso não é recomendado com tabelas grandes: você ainda será cobrado por toda a ação `Scan`, mesmo se apenas alguns itens correspondentes forem retornados.)

PartiQL for DynamoDB

Com PartiQL, você executa uma varredura usando a operação `ExecuteStatement` para retornar todo o conteúdo de uma tabela usando a instrução `Select`.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

Observe que esta instrução retornará todos os itens para a tabela Music.

Para obter exemplos de código que usam `Select` e `ExecuteStatement`, consulte Instruções Select em PartiQL para DynamoDB.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao gerenciar índices

Índices dão a você acesso a padrões de consulta alternativos, e você pode agilizar as consultas. Esta seção compara e contrasta a criação e o uso de índices no SQL e no Amazon DynamoDB.

Caso esteja usando um banco de dados relacional ou o DynamoDB, você deve ser criterioso com a criação do índice. Sempre que uma gravação ocorre em uma tabela, todos os índices da tabela devem ser atualizados. Em um ambiente que exige muita gravação com tabelas grandes, isso pode consumir grandes quantidades de recursos do sistema. Em um ambiente somente leitura ou em sua maioria de leitura, essa não é uma grande preocupação. Entretanto, você deve garantir que os índices estejam realmente sendo usados por seu aplicativo, e não simplesmente ocupando espaço.

Tópicos

- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao criar um índice
- Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao consultar e varrer um índice

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao criar um índice

Compare a instrução `CREATE INDEX` em SQL com a operação `UpdateTable` no Amazon DynamoDB.

Tópicos

- Criar um índice com o SQL
- Criar um índice no DynamoDB

Criar um índice com o SQL

Em um banco de dados relacional, índice é uma estrutura de dados que permite realizar consultas rápidas em diferentes colunas em uma tabela. Você pode usar a instrução SQL `CREATE INDEX` para adicionar um índice a uma tabela existente, especificando as colunas a serem indexadas. Após

a criação do índice, você pode consultar os dados na tabela como sempre, mas agora o banco de dados pode usar o índice para localizar rapidamente as linhas especificadas na tabela, em vez de verificar toda a tabela.

Depois que você cria um índice, o banco de dados o mantém para você. Sempre que você modifica os dados na tabela, o índice é modificado automaticamente para refletir as alterações da tabela.

No MySQL, você pode criar um índice da forma a seguir.

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

Criar um índice no DynamoDB

No DynamoDB você pode criar e usar um índice secundário para atender a objetivos semelhantes.

Os índices no DynamoDB são diferentes dos seus equivalentes relacionais. Ao criar um índice secundário, você deve especificar os atributos de chave: uma chave de partição e uma chave de classificação. Depois de criar o índice secundário, você pode usar a operação `Query` ou `Scan` da mesma forma como faria com uma tabela. O DynamoDB não tem um otimizador de consultas. Portanto, um índice secundário é usado apenas quando você usa a operação `Query` ou `Scan`.

O DynamoDB aceita dois tipos diferentes de índices:

- Os índices secundários globais: a chave primária do índice pode ser qualquer dois atributos de sua tabela.
- Índices secundários locais: a chave de partição do índice deve ser a mesma que a chave de partição de sua tabela. No entanto, a chave de classificação pode ser qualquer outro atributo.

O DynamoDB garante que os dados em um índice secundário sejam finais consistentes com sua tabela. Você pode solicitar operações `Query` ou `Scan` altamente consistentes em uma tabela ou em um índice secundário local. No entanto, índices secundários globais somente oferecem suporte à consistência eventual.

É possível adicionar um índice secundário global a uma tabela existente usando a operação `UpdateTable` e especificando `GlobalSecondaryIndexUpdates`.

```
{
    TableName: "Music",
```

```
    AttributeDefinitions:[
        {AttributeName: "Genre", AttributeType: "S"},
        {AttributeName: "Price", AttributeType: "N"}
    ],
    GlobalSecondaryIndexUpdates: [
        {
            Create: {
                IndexName: "GenreAndPriceIndex",
                KeySchema: [
                    {AttributeName: "Genre", KeyType: "HASH"}, //Partition key
                    {AttributeName: "Price", KeyType: "RANGE"}, //Sort key
                ],
                Projection: {
                    "ProjectionType": "ALL"
                },
                ProvisionedThroughput: {                                // Only
 specified if using provisioned mode
                    "ReadCapacityUnits": 1,"WriteCapacityUnits": 1
                }
            }
        }
    ]
}
```

Você deve fornecer os parâmetros a seguir para `UpdateTable`:

- `TableName`: a tabela à qual o índice será associado.
- `AttributeDefinitions`: os tipos de dados dos atributos de esquema de chaves do índice.
- `GlobalSecondaryIndexUpdates`: detalhes sobre o índice que você deseja criar:
  - `IndexName`: um nome para o índice.
  - `KeySchema`: os atributos que são usados para a chave primária do índice.
  - `Projection`: atributos da tabela que são copiados para o índice. Neste caso, `ALL` significa que todos os atributos são copiados.
  - `ProvisionedThroughput (for provisioned tables)`: o número de leituras e gravações por segundo que são necessários para este índice. (Isso é separado das configurações de throughput provisionado da tabela.)

Parte dessa operação envolve backfilling de dados da tabela para o novo índice. Durante o backfilling, a tabela permanece disponível. No entanto, o índice não está pronto até que seu atributo

`Backfilling` mude de verdadeiro para falso. Você pode usar a operação `DescribeTable` para visualizar esse atributo.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao consultar e varrer um índice

Compare a consulta e a verificação de um índice usando a instrução SELECT em SQL com as operações `Query` e `Scan` no Amazon DynamoDB.

Tópicos

- Consultar e verificar um índice com o SQL
- Consultar e verificar um índice no DynamoDB

Consultar e verificar um índice com o SQL

Em um banco de dados relacional, você não trabalha diretamente com índices. Em vez disso, você consulta tabelas, emitindo instruções `SELECT`, e o otimizador de consultas pode fazer uso de índices.

Um otimizador de consultas é um componente do sistema de gerenciamento de banco de dados relacional (RDBMS - relational database management system) que avalia os índices disponíveis e determina se eles podem ser usados para agilizar uma consulta. Se os índices puderem ser usados para acelerar uma consulta, o RDBMS acessará o índice primeiro e, em seguida, o usará para localizar os dados na tabela.

Estas são algumas instruções SQL que podem usar GenreAndPriceIndex para melhorar a performance. Presumimos que a tabela Music tenha dados suficientes para que o otimizador de consultas decida usar esse índice, em vez de simplesmente verificar a tabela inteira.

```
/* All of the rock songs */

SELECT * FROM Music
WHERE Genre = 'Rock';
```

```
/* All of the cheap country songs */

SELECT Artist, SongTitle, Price FROM Music
WHERE Genre = 'Country' AND Price < 0.50;
```

Consultar e verificar um índice no DynamoDB

No DynamoDB, execute as operações `Query` e `Scan` diretamente no índice, como faria em uma tabela. É possível usar a API do DynamoDB ou do PartiQL (uma linguagem de consultas compatível com SQL) para consultar ou verificar o índice. Você deve especificar `TableName` e `IndexName`.

As seguintes são algumas consultas sobre GenreAndPriceIndex no DynamoDB. (O esquema de chaves desse índice consiste em Genre e Price.)

DynamoDB API

```
// All of the rock songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

Este exemplo usa uma `ProjectionExpression` para indicar que somente alguns dos atributos, em vez de todos eles, são exibidos nos resultados.

```
// All of the cheap country songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre and Price < :price",
    ExpressionAttributeValues: {
        ":genre": "Country",
        ":price": 0.50
    },
    ProjectionExpression: "Artist, SongTitle, Price"
};
```

Veja a seguir uma verificação em GenreAndPriceIndex.

```
// Return all of the data in the index
```

```
{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

Com PartiQL, use a instrução `Select` PartiQL para realizar consultas e varreduras no índice.

```
// All of the rock songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock'
```

```
// All of the cheap country songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock' AND Price < 0.50
```

Veja a seguir uma verificação em GenreAndPriceIndex.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> Para obter exemplos de códigos que usam `Select`, consulte Instruções Select em PartiQL para DynamoDB.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao modificar dados de uma tabela

A linguagem SQL fornece a instrução `UPDATE` para modificar dados. O Amazon DynamoDB usa a operação `UpdateItem` para realizar tarefas semelhantes.

Tópicos

- Modificar dados em uma tabela com o SQL
- Modificar dados em uma tabela no DynamoDB

Modificar dados em uma tabela com o SQL

No SQL, você usa a instrução `UPDATE` para modificar uma ou mais linhas. A cláusula `SET` especifica novos valores para uma ou mais colunas, e a cláusula `WHERE` determina quais linhas são modificadas. Veja um exemplo a seguir.

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

Se nenhuma linha corresponder à cláusula `WHERE`, a instrução `UPDATE` não terá efeito.

Modificar dados em uma tabela no DynamoDB

No DynamoDB, você pode usar a API do DynamoDB ou do PartiQL (uma linguagem de consultas compatível com SQL) para modificar um único item. Caso deseje modificar vários itens, você deve usar várias operações.

DynamoDB API

Com a API do DynamoDB, use a operação `UpdateItem` para modificar um único item.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

Você deve especificar os atributos `Key` do item a ser modificado, e uma `UpdateExpression` para especificar valores de atributo. `UpdateItem` comporta-se como uma operação “upsert”. O item será atualizado se existir na tabela. Do contrário, um novo item será adicionado (inserido).

`UpdateItem` oferece suporte a gravações condicionais, nas quais a operação será bem-sucedida apenas se uma `ConditionExpression` específica for verdadeira. Por exemplo, a operação `UpdateItem` a seguir não realiza a atualização, a menos que o preço da música seja maior ou igual a 2,00.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ConditionExpression: "Price >= :p",
    ExpressionAttributeValues: {
        ":label": "Global Records",
        ":p": 2.00
    }
}
```

`UpdateItem` também oferece suporte a contadores atômicos ou a atributos do tipo `Number` que podem ser aumentados ou reduzidos. Os contadores atômicos são semelhantes de muitas maneiras a geradores de sequências, colunas de identidade ou campos de incremento automático em bancos de dados SQL.

Veja a seguir um exemplo de uma operação `UpdateItem` para inicializar um novo atributo (Plays) e controlar o número de vezes em que uma música é reproduzida.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
```

```
}
```

O parâmetro `ReturnValues` é definido como `UPDATED_NEW`, o que retorna os novos valores de todos os atributos que foram atualizados. Neste caso, ele retorna 0 (zero).

Sempre que alguém reproduzir essa música, podemos usar a operação `UpdateItem` a seguir para aumentar Plays por um.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = Plays + :incr",
    ExpressionAttributeValues: {
        ":incr": 1
    },
    ReturnValues: "UPDATED_NEW"
}
```

PartiQL for DynamoDB

Com PartiQL, use a operação `ExecuteStatement` para modificar um item em uma tabela, usando a instrução `Update` PartiQL.

A chave primária dessa tabela consiste em Artist e SongTitle. Você deve especificar valores para esses atributos.

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Também é possível modificar vários campos de uma só vez, como no exemplo a seguir.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

`Update` também oferece suporte a contadores atômicos ou a atributos do tipo `Number` que podem ser aumentados ou reduzidos. Os contadores atômicos são semelhantes de muitas

maneiras a geradores de sequências, colunas de identidade ou campos de incremento automático em bancos de dados SQL.

Veja a seguir um exemplo de instrução `Update` para inicializar um novo atributo (Plays) para controlar o número de vezes em que uma música foi reproduzida.

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Sempre que alguém reproduzir essa música, poderemos usar a instrução `Update` a seguir para aumentar Plays (Reproduções) em um.

```
UPDATE Music
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> Para obter exemplos de código que usam `Update` e `ExecuteStatement`, consulte Instruções Update em PartiQL para DynamoDB.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao excluir dados de uma tabela

No SQL, a instrução `DELETE` remove uma ou mais linhas de uma tabela. O Amazon DynamoDB usa a operação `DeleteItem` para excluir um item de cada vez.

Tópicos

- Excluir dados de uma tabela com o SQL
- Excluir dados de uma tabela no DynamoDB

Excluir dados de uma tabela com o SQL

No SQL, você usa a instrução `DELETE` para excluir uma ou mais linhas. A cláusula `WHERE` determina as linhas que você deseja modificar. Veja um exemplo a seguir.

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

Você pode modificar a cláusula `WHERE` para excluir várias linhas. Por exemplo, é possível excluir todas as músicas de um determinado artista, conforme mostrado no exemplo a seguir.

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

Excluir dados de uma tabela no DynamoDB

No DynamoDB, você pode usar a API do DynamoDB ou do PartiQL (uma linguagem de consultas compatível com SQL) para excluir um único item. Caso deseje modificar vários itens, você deve usar várias operações.

DynamoDB API

Com a API do DynamoDB, use a operação `DeleteItem` para excluir dados de uma tabela, um item de cada vez. Você deve especificar os valores de chave primária do item.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> Além do `DeleteItem`, o Amazon DynamoDB comporta uma operação `BatchWriteItem` para excluir vários itens ao mesmo tempo.

`DeleteItem` oferece suporte a gravações condicionais, nas quais a operação será bem-sucedida apenas se uma `ConditionExpression` específica for verdadeira. Por exemplo, a operação `DeleteItem` a seguir exclui o item somente se ele tiver um atributo RecordLabel.

```
{
    TableName: "Music",
    Key: {
```

```
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

PartiQL for DynamoDB

Com PartiQL, use a instrução `Delete` por meio da operação `ExecuteStatement` para excluir dados de uma tabela, um item de cada vez. Você deve especificar os valores de chave primária do item.

A chave primária dessa tabela consiste em Artist e SongTitle. Você deve especificar valores para esses atributos.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

Você também pode especificar condições adicionais para a operação. A operação `DELETE` a seguir somente excluirá o item se ele tiver mais de 11 Awards (Prêmios).

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> Para obter exemplos de código que usam `DELETE` e `ExecuteStatement`, consulte Instruções Delete em PartiQL para DynamoDB.

Diferenças entre um banco de dados relacional (SQL) e o DynamoDB ao remover uma tabela

No SQL, você usa a instrução `DROP TABLE` para remover uma tabela. No Amazon DynamoDB, você usa a operação `DeleteTable`.

Tópicos

- Remover uma tabela com o SQL
- Remover uma tabela no DynamoDB

Remover uma tabela com o SQL

Quando você não precisar mais de uma tabela e quiser descartá-la permanentemente, use a instrução `DROP TABLE` no SQL.

```
DROP TABLE Music;
```

Depois que uma tabela é eliminada, ela não pode ser recuperada. (Alguns bancos de dados relacionais permitem que você desfaça uma operação `DROP TABLE`, mas essa funcionalidade é específica do fornecedor e não é amplamente implementada.)

Remover uma tabela no DynamoDB

No DynamoDB, `DeleteTable` é uma operação semelhante. No exemplo a seguir, a tabela é excluída permanentemente.

```
{
    TableName: "Music"
}
```

Recursos e ferramentas de aprendizado do Amazon DynamoDB

É possível usar os recursos adicionais a seguir para conhecer o DynamoDB e trabalhar com ele.

Tópicos

- Ferramentas para codificação e visualização
- Artigos de Recomendações
- Artigos do Centro de Conhecimentos
- Postagens em blogs, repositórios e guias
- Apresentações sobre modelagem de dados e padrões de design
- Cursos de treinamento

Ferramentas para codificação e visualização

É possível usar as seguintes ferramentas de codificação e visualização para trabalhar com o DynamoDB:

- NoSQL Workbench para Amazon DynamoDB: uma ferramenta visual unificada que ajuda a projetar, criar, consultar e gerenciar tabelas do DynamoDB. Ela fornece recursos de modelagem de dados, visualização de dados e desenvolvimento de consultas.
- Dynobase: uma ferramenta de desktop que facilita a visualização e o trabalho com tabelas do DynamoDB, a criação de código de aplicação e a edição de registros com validação em tempo real.
- DynamoDB Toolbox: um projeto de Jeremy Daly que fornece utilitários ideais para trabalhar com modelagem de dados e JavaScript e Node.js.
- Processador de fluxos do DynamoDB: uma ferramenta simples que torna extremamente fácil trabalhar com fluxos do DynamoDB.

Artigos de Recomendações

O recurso Recomendações da AWS fornece estratégias, guias e padrões comprovados para ajudar a acelerar seus projetos. Esses recursos foram desenvolvidos por especialistas em tecnologia da AWS e pela comunidade global de parceiros da AWS, com base em seus vários anos de experiência ajudando os clientes a alcançar seus objetivos empresariais.

Modelagem e migração de dados

- Um modelo de dados hierárquico no DynamoDB
- Modelagem de dados com o DynamoDB
- Migrar um banco de dados Oracle para o DynamoDB usando o AWS DMS

Tabelas globais

- Usar as tabelas globais do Amazon DynamoDB

Sem servidor

- Implementar o padrão de saga sem servidor com o AWS Step Functions

Arquitetura de SaaS

- Gerenciar locatários em vários produtos de SaaS em um único ambiente de gerenciamento
- Integração de locatários na arquitetura de SaaS para o modelo de silo usando C# e o AWS CDK

Proteção e movimentação de dados

- Configurar o acesso entre contas ao Amazon DynamoDB
- Opções completas de cópia de tabela para o DynamoDB
- Estratégia de recuperação de desastres para bancos de dados na AWS

Diversos

- Ajude a aplicar a marcação no DynamoDB

Demonstração em vídeo das recomendações

- Usar a arquitetura sem servidor para criar pipelines de dados
- Novartis - – mecanismo de compra: portal de compras baseado em IA
- Veritiv: habilite insights para prever a demanda de vendas em data lakes da AWS
- mimik: nuvem de borda híbrida que utiliza a AWS para atender à malha de microsserviços de borda
- Alterar a captura de dados com o Amazon DynamoDB

Para ver mais artigos e vídeos de recomendações para o DynamoDB, consulte Recomendações.

Artigos do Centro de Conhecimentos

Os artigos e vídeos do Centro de Conhecimentos da AWS abrangem as perguntas e solicitações mais frequentes que recebemos dos clientes da AWS. Veja a seguir alguns artigos atuais do Centro de Conhecimentos sobre tarefas específicas relacionadas ao DynamoDB:

Otimização de custo

- Como otimizo os custos com o Amazon DynamoDB?

Controle de utilização e latência

- Como posso solucionar problemas de alta latência em uma tabela do Amazon DynamoDB?
- Por que minha tabela do DynamoDB está com controle de utilização?
- Por que minha tabela sob demanda do DynamoDB está com controle de utilização?

Paginação

- Como implementar a paginação no DynamoDB

Transações

- Por que minha chamada de API `TransactWriteItems` falha no DynamoDB

Solução de problemas

- Como resolver problemas com o ajuste de escala automático do DynamoDB?
- Como solucionar erros HTTP 4XX no DynamoDB

Para ver mais artigos e vídeos sobre o DynamoDB, consulte os artigos do Centro de Conhecimentos.

Postagens em blogs, repositórios e guias

Além do Guia do desenvolvedor do DynamoDB, há muitos recursos úteis para trabalhar com o DynamoDB. Veja abaixo algumas publicações de blog, repositórios e guias selecionados para trabalhar com o DynamoDB:

- AWSrepositório de Exemplos de código do DynamoDB em várias linguagens do SDK do AWS: Node.js, Java, Python, .Net, Go e Rust.
- The DynamoDB Book: um guia abrangente de Alex DeBrie que ensina uma abordagem orientada por estratégias para modelagem de dados com o DynamoDB.
- DynamoDB Guide: um guia aberto de Alex DeBrie que explica os conceitos básicos e recursos avançados do banco de dados NoSQL do DynamoDB.
- How to switch from RDBMS to DynamoDB in 20 easy steps: uma lista de etapas úteis para aprender sobre a modelagem de dados de Jeremy Daly.
- Página de dicas do DynamoDB JavaScript DocumentClient: dicas para ajudar você a começar a criar aplicações com o DynamoDB em um ambiente Node.js ou JavaScript.
- Vídeos dos principais conceitos do DynamoDB: essa playlist aborda muitos dos principais conceitos do DynamoDB.

Apresentações sobre modelagem de dados e padrões de design

É possível usar os seguintes recursos sobre modelagem de dados e padrões de design para aproveitar ao máximo o DynamoDB:

- AWS re:Invent 2019: Data modeling with DynamoDB
  - Uma palestra de Alex DeBrie que apresenta os princípios da modelagem de dados do DynamoDB.
- AWS re:Invent 2020: Modelagem de dados com o DynamoDB – Parte 1
- AWS re:Invent 2020: Modelagem de dados com o DynamoDB – Parte 2
- AWS re:Invent 2017: Advanced design patterns
- AWS re:Invent 2018: Advanced design patterns
- AWS re:Invent 2019: Advanced design patterns
  - Jeremy Daly compartilha 12 key takeaways nesta sessão.
- AWS re:Invent 2020: Padrões de design avançado do DynamoDB – Parte 1
- AWS re:Invent 2020: Padrões de design avançado do DynamoDB – Parte 2
- DynamoDB Office Hours on Twitch

> **Note**
>
> Cada sessão abrange diferentes casos de uso e exemplos.

Cursos de treinamento

Há muitos cursos de treinamento e opções educacionais diferentes para aprender mais sobre o DynamoDB. Veja aqui alguns exemplos atuais:

- Desenvolvimento com o Amazon DynamoDB: projetado pela AWS para levar você de iniciante a especialista em desenvolvimento de aplicações do mundo real com modelagem de dados para o Amazon DynamoDB.
- DynamoDB deep dive course: um curso da Pluralsight.
- Amazon DynamoDB: desenvolvendo aplicações guiadas por banco de dados NoSQL: um curso da equipe AWS Training and Certification realizado no edX.

Leituras e gravações do DynamoDB

As leituras e as gravações do DynamoDB referem-se às operações que recuperam dados de uma tabela (leituras) e inserem, atualizam ou excluem dados em uma tabela (gravações). Quando você trabalha com o DynamoDB, é essencial entender os conceitos de leitura e de gravação, pois eles afetam diretamente a performance e o custo da aplicação.

Este tópico fornece detalhes sobre os diferentes tipos de consistência de leitura que se aplicam ao DynamoDB. Também descreve o consumo unitário para diferentes operações de leitura e de gravação que você pode realizar.

Tópicos

- Consistência de leitura do DynamoDB
- Operações de leitura e de gravação do DynamoDB

Consistência de leitura do DynamoDB

O Amazon DynamoDB lê dados de tabelas, índices secundários locais (LSIs), índices secundários globais (GSIs) e fluxos. Para obter mais informações, consulte Componentes principais do Amazon DynamoDB. Tanto as tabelas quanto os LSIs oferecem duas opções de consistência de leitura: leituras finais consistentes (padrão) e leituras altamente consistentes. Todas as leituras de GSIs e fluxos são finais consistentes.

Quando sua aplicação grava dados em uma tabela do DynamoDB e recebe uma resposta HTTP 200 (OK), isso significa que a gravação foi concluída com êxito e persistida de forma durável. O DynamoDB oferece isolamento confirmado para leitura e garante que as operações de leitura sempre retornem valores confirmados de um item. A leitura nunca apresentará uma visão do item a partir de uma gravação que, em última análise, não foi concluída. Isolamento confirmado para leitura não previne modificações do item imediatamente após operação de leitura.

Leituras finais consistentes

O tipo final consistente é o modelo de consistência padrão para todas as operações de leitura. Ao emitir leituras finais consistentes para uma tabela ou um índice do DynamoDB, as respostas podem não refletir os resultados de uma operação de gravação recém-concluída. Se você repetir sua solicitação de leitura após um curto período, em algum momento a resposta deverá retornar o item mais recente. As leituras finais consistentes são compatíveis com tabelas, índices secundários locais

e índices secundários globais. Observe também que todas as leituras de um fluxo do DynamoDB também são do tipo final consistente.

As leituras finais consistentes custam metade do preço das leituras altamente consistentes. Para obter mais informações, consulte a Definição de preço do Amazon DynamoDB.

Leituras altamente consistentes

As operações de leitura, como `GetItem`, `Query` e `Scan`, fornecem um parâmetro `ConsistentRead` opcional. Se você definir `ConsistentRead` como true, o DynamoDB retornará uma resposta com os dados mais atualizados, refletindo as atualizações de todas as operações de gravação anteriores que tiveram êxito. Leituras altamente consistentes só são compatíveis com tabelas e índices secundários locais. Não há suporte para leituras altamente consistentes de um índice secundário global ou de um fluxo do DynamoDB.

Consistência de leitura de tabelas globais

O DynamoDB também oferece suporte a tabelas globais para replicação multiativa e multirregional. Uma tabela global é composta por várias tabelas de réplica em diferentes regiões da AWS. Todas as alterações feitas em qualquer item de qualquer tabela de réplica serão replicadas para todas as outras réplicas dentro da mesma tabela global, geralmente em menos de um segundo, e serão do tipo final consistente. Para obter mais informações, consulte Modos de consistência.

Operações de leitura e de gravação do DynamoDB

As operações de leitura do DynamoDB permitem que você recupere um ou mais itens de uma tabela especificando o valor da chave de partição e, opcionalmente, o valor da chave de classificação. Usando operações de gravação do DynamoDB, é possível inserir, atualizar ou excluir itens em uma tabela. Este tópico explica o consumo unitário de capacidade para essas duas operações.

Tópicos

- Consumo unitário de capacidade para operações de leitura
- Consumo unitário de capacidade para operações de gravação

Consumo unitário de capacidade para operações de leitura

As solicitações de leitura do DynamoDB podem ser altamente consistentes, consistentes ao final ou transacionais.

- Uma solicitação de leitura altamente consistente de um item de até 4 KB exige uma unidade de leitura.
- Uma solicitação de leitura final consistente de um item de até 4 KB exige meia unidade de leitura.
- Uma solicitação de leitura transacional de um item de até 4 KB exige duas unidades de leitura.

Para saber mais sobre os modelos de consistência de leitura do DynamoDB, consulte Consistência de leitura do DynamoDB.

Os tamanhos de item para leituras são arredondados para o próximo múltiplo de 4 KB. Por exemplo, ler um item com 3.500 bytes consome a mesmo throughput que a leitura de um item de 4 KB.

Se você precisar ler um item com mais de 4 KB, o DynamoDB necessitará de unidades de leitura adicionais. O número total de unidades de leitura depende do tamanho do item e do tipo de leitura desejada (isto é, final consistente ou altamente consistente). Por exemplo, se o tamanho do item for 8 KB, serão necessárias duas unidades de leitura para comportar uma leitura altamente consistente. Você precisará de uma unidade se escolher leituras finais consistentes ou quatro unidades para uma solicitação de leitura transacional.

A lista a seguir descreve como as operações de leitura do DynamoDB consomem unidades de leitura:

- GetItem: lê um único item de uma tabela. Para determinar o número de unidades de leitura que `GetItem` consumirá, arredonde o tamanho do item para o próximo limite de 4 KB. Esse será o número de unidades de capacidade necessárias se você tiver especificado uma leitura altamente consistente. Em relação a uma leitura final consistente, que é o padrão, divida esse número por dois.

  Por exemplo, se você ler um item que tem 3,5 KB, o DynamoDB arredondará o tamanho do item para 4 KB. Se você ler um item de 10 KB, o DynamoDB arredondará o tamanho do item para 12 KB.
- BatchGetItem: lê até cem itens de uma ou mais tabelas. O DynamoDB processa cada item no lote como uma solicitação `GetItem` individual. O DynamoDB primeiro arredonda o tamanho de cada item para o próximo limite de 4 KB e, depois, calcula o tamanho total. O resultado não é necessariamente o mesmo que o tamanho total de todos os itens. Por exemplo, se `BatchGetItem` ler dois itens com os tamanhos 1,5 KB e 6,5 KB, o DynamoDB calculará o tamanho como 12 KB (4 KB + 8 KB). O DynamoDB não calcula o tamanho como 8 KB (1,5 KB + 6,5 KB).

- Query: lê vários itens que têm o mesmo valor de chave de partição. Todos os itens exibidos são tratados como uma única operação de leitura, em que o DynamoDB calcula o tamanho total de todos os itens. Em seguida, o DynamoDB arredonda o tamanho para o próximo limite de 4 KB. Por exemplo, suponha que a consulta retorne 10 itens cujo tamanho combinado é 40,8 KB. O DynamoDB arredonda o tamanho do item da operação para 44 KB. Se uma consulta retornar 1.500 itens de 64 bytes cada, o tamanho cumulativo será 96 KB.
- Scan: lê todos os itens em uma tabela. O DynamoDB considera o tamanho dos itens que são avaliados, não o tamanho dos itens retornados pela verificação. Para ter mais informações sobre as operações Scan, consulte Verificar tabelas no DynamoDB.

> **Important**
>
> Se você executar uma operação de leitura em um item inexistente, o DynamoDB ainda assim consumirá o throughput de leitura, conforme descrito acima. Para as operações `Query`/`Scan`, você ainda pagará por um throughput de leitura adicional com base na consistência da leitura e no número de partições pesquisadas para atender à solicitação, mesmo que não existam dados.

Para qualquer operação que retorna itens, você pode solicitar um subconjunto de atributos a serem recuperados. No entanto, isso não tem nenhum impacto sobre os cálculos de tamanho dos itens. Além disso, `Query` e `Scan` podem retornar as contagens de itens, em vez de valores de atributo. A contagem de itens exige a mesma quantidade de unidades de leitura e está sujeita aos mesmos cálculos de tamanho de item. Isso ocorre porque o DynamoDB precisa ler cada item para aumentar a contagem.

Consumo unitário de capacidade para operações de gravação

Uma unidade de gravação representa uma gravação para um item com até 1 KB. Se você precisar gravar um item maior que 1 KB, o DynamoDB precisará consumir unidades de gravação adicionais. As solicitações de gravação transacional exigem duas unidades de gravação para executar uma gravação para itens de até 1 KB. O número total de unidades de solicitação de gravação necessárias varia de acordo com o tamanho do item. Por exemplo, se o tamanho do item for 2 KB, serão necessárias duas unidades de gravação para comportar uma solicitação de gravação ou quatro unidades de gravação para uma solicitação de gravação transacional.

Os tamanhos de item para gravações são arredondados até o próximo múltiplo de 1 KB. Por exemplo, gravar um item de 500 bytes consome a mesmo throughput que gravar um item de 1 KB.

A lista a seguir descreve como as operações de gravação do DynamoDB consomem unidades de gravação:

- PutItem: grava um único item em uma tabela. Se um item com a mesma chave primária existe na tabela, a operação substitui o item. Para calcular o consumo de throughput provisionado, o tamanho do item que importa é o maior dos dois.
- UpdateItem: modifica um único item na tabela. O DynamoDB considera o tamanho do item como ele aparece antes e depois da atualização. O throughput provisionado consumido reflete o maior desses tamanhos de item. Mesmo se você atualizar um subconjunto dos atributos do item, `UpdateItem` ainda consumirá a quantidade total do throughput provisionado (o maior tamanho de item de “antes” e “depois”).
- DeleteItem: remove um único item de uma tabela. O consumo de throughput provisionado é baseado no tamanho do item excluído.
- BatchWriteItem: grava até 25 itens em uma ou mais tabelas. O DynamoDB processa cada item no lote como uma solicitação `PutItem` ou `DeleteItem` individual (atualizações não são compatíveis). O DynamoDB primeiro arredonda o tamanho de cada item para o próximo limite de 1 KB e, depois, calcula o tamanho total. O resultado não é necessariamente o mesmo que o tamanho total de todos os itens. Por exemplo, se `BatchWriteItem` ler dois itens com os tamanhos 500 bytes e 3,5 KB, o DynamoDB calculará o tamanho como 5 KB (1 KB + 4 KB). O DynamoDB não calcula o tamanho como 4 KB (500 KB + 3,5 KB).

Para operações `PutItem`, `UpdateItem` e `DeleteItem`, o DynamoDB arredonda o tamanho do item até o próximo 1 KB. Por exemplo, se você inserir ou excluir um item de 1,6 KB, o DynamoDB arredondará o tamanho do item até 2 KB.

As operações `PutItem`, `UpdateItem` e `DeleteItem` permitem gravações condicionais, em que você especifica uma expressão que deve ser verdadeira para a operação ter êxito. Se a expressão for falsa, o DynamoDB ainda consumirá unidades de capacidade de gravação da tabela. A quantidade de unidades de capacidade de gravação consumida depende do tamanho do item. Esse item já pode existir na tabela ou ser um novo que você esteja tentando criar ou atualizar. Por exemplo, vamos supor que um item que já existe tenha 300 KB. O novo item que você está tentando criar ou atualizar tem 310 KB. As unidades de capacidade de gravação consumidas terão 310 KB para o novo item.

Capacidade de throughput do DynamoDB

Esta seção apresenta uma visão geral dos dois modos de throughput disponíveis para a tabela do DynamoDB e considerações sobre a seleção do modo de capacidade adequado para a aplicação. O modo de throughput de uma tabela determina como a capacidade de uma tabela é gerenciada. O modo de throughput também determina como é realizada a cobrança de operações de leitura e de gravação nas tabelas. No Amazon DynamoDB, é possível escolher entre o modo sob demanda e o modo provisionado para que as tabelas possam atender a diferentes requisitos de workload.

Tópicos

- Modo sob demanda
- Modo provisionado
- Modo de capacidade sob demanda do DynamoDB
- Modo de capacidade provisionada do DynamoDB
- Noções básicas sobre o throughput a quente do DynamoDB
- Capacidade de expansão e capacidade adaptável do DynamoDB
- Considerações ao alternar os modos de capacidade no DynamoDB

Modo sob demanda

O modo sob demanda do Amazon DynamoDB é uma opção de throughput sem servidor que simplifica o gerenciamento do banco de dados e escala automaticamente para comportar as aplicações mais exigentes dos clientes. O DynamoDB sob demanda permite que você crie uma tabela sem se preocupar com planejamento de capacidade, monitoramento de uso e configuração de políticas de ajuste de escala. Para solicitações de leitura e de gravação, o DynamoDB sob demanda oferece o modelo de preço de pagamento por solicitação para que você pague apenas pelo que usar. Para tabelas do modo sob demanda, não é necessário especificar o throughput de leitura e gravação que você espera que sua aplicação execute.

O modo sob demanda é a opção de throughput padrão e recomendada para a maioria das workloads do DynamoDB. O DynamoDB lida com todos os aspectos do gerenciamento e do ajuste de escala de throughput para comportar workloads que podem começar pequenas e chegar a milhões de solicitações por segundo. É possível ler e gravar em suas tabelas do DynamoDB conforme necessário sem gerenciar a capacidade de throughput na tabela. Para obter mais informações, consulte Modo de capacidade sob demanda do DynamoDB.

Modo provisionado

No modo provisionado, especifique o número de leituras e de gravações por segundo de que você precisa para a aplicação. Você será cobrado com base na capacidade de leitura e de gravação por hora provisionada, não na quantidade de capacidade provisionada que você realmente consumiu. Isso ajuda a governar seu uso do DynamoDB para permanecer no lugar ou abaixo de uma taxa de solicitação definida para obter previsão de custos.

É possível optar por usar a capacidade provisionada se tiver workloads estáveis com crescimento previsível e se puder prever com segurança os requisitos de capacidade para sua aplicação. Para obter mais informações, consulte Modo de capacidade provisionada do DynamoDB.

Modo de capacidade sob demanda do DynamoDB

O Amazon DynamoDB sob demanda oferece uma experiência de banco de dados verdadeiramente sem servidor que escala automaticamente para acomodar as workloads mais exigentes sem planejamento de capacidade. O modo sob demanda simplifica o processo de configuração, elimina o gerenciamento e o monitoramento da capacidade e oferece rápido ajuste de escala automático. Com o preço de pagamento por solicitação, você não precisa se preocupar com a capacidade ociosa, pois paga apenas pelo throughput que realmente usa. Você é cobrado por solicitação de leitura ou de gravação, portanto, seus custos refletem diretamente o uso real.

Quando você escolhe o modo sob demanda, o DynamoDB acomoda instantaneamente o crescimento e a redução das workloads para qualquer nível de tráfego previamente registrado. Se o nível de tráfego de uma workload atingir um novo pico, o DynamoDB escalará automaticamente para acomodar os maiores requisitos de throughput. O modo sob demanda é a opção de throughput padrão, recomendada porque simplifica a criação de aplicações modernas e sem servidor que podem começar pequenas e chegar a milhões de solicitações por segundo. Depois que a escala de sua tabela sob demanda for aumentada horizontalmente, você poderá voltar a ter instantaneamente o mesmo throughput no futuro, sem controle de utilização. Se você não estiver direcionando tráfego para sua tabela, com o sistema sob demanda, não haverá cobranças por throughput. Para ter mais informações sobre as propriedades de escalabilidade do modo sob demanda, consulte Throughput inicial e propriedades de escalabilidade.

Tabelas que usam o modo sob demanda entregam a mesma latência de milissegundo de digito único, o acordo de serviço (SLA) e a segurança já oferecidos pelo modo provisionado do DynamoDB.

> **Note**
>
> Por padrão, o DynamoDB protege você contra o uso indesejado e descontrolado. Para escalar além dos limites de throughput de leitura e gravação em nível de tabela de 40.000 para todas as tabelas em sua conta, você pode solicitar um aumento dessa cota. As solicitações de throughput que excedem a cota de throughput de tabela padrão têm controle de utilização. Para obter mais informações, consulte Cotas padrão de throughput.

Você também pode configurar o throughput máximo de leitura ou de gravação (ou de ambas) por segundo para tabelas individuais sob demanda e índices secundários globais. Ao configurar o throughput, é possível manter o uso e os custos por tabela limitados, proteger-se contra o aumento inadvertido nos recursos consumidos e evitar o uso excessivo para ter um gerenciamento previsível dos custos. As solicitações de throughput que excedem o throughput máximo da tabela são limitadas. É possível modificar o throughput máximo específico da tabela a qualquer momento, com base nos requisitos da aplicação. Para obter mais informações, consulte Throughput máximo do DynamoDB para tabelas sob demanda.

Para começar, crie ou atualize um modo sob demanda. Para obter mais informações, consulte Operações básicas em tabelas do DynamoDB.

É possível alternar as tabelas do modo de capacidade provisionada para o modo sob demanda até quatro vezes em um período contínuo de 24 horas. É possível alternar as tabelas do modo sob demanda para o modo de capacidade provisionada a qualquer momento.

Para ter mais informações sobre como alternar entre os modos de capacidade de leitura e de gravação, consulte Considerações ao alternar os modos de capacidade no DynamoDB. Para se informar sobre cotas de tabela sob demanda, consulte Throughput de leitura e gravação.

Tópicos

- Unidades de solicitação de leitura e unidades de solicitação de gravação
- Throughput inicial e propriedades de escalabilidade
- Throughput máximo do DynamoDB para tabelas sob demanda

Unidades de solicitação de leitura e unidades de solicitação de gravação

O DynamoDB cobra pelas leituras e gravações que sua aplicação realiza nas tabelas em termos de unidades de solicitação de leitura e unidades de solicitação de gravação.

Uma unidade de solicitação de leitura representa uma leitura altamente consistente por segundo, ou duas leituras finais consistentes por segundo, para um item com até 4 KB de tamanho. Para ter mais informações sobre os modelos de consistência de leitura do DynamoDB, consulte Consistência de leitura do DynamoDB.

Uma unidade de solicitação de gravação representa uma operação de gravação por segundo para um item com até 1 KB de tamanho.

Para ter mais informações sobre como as unidades de leitura e de gravação são consumidas, consulte Operações de leitura e de gravação do DynamoDB.

Throughput inicial e propriedades de escalabilidade

Tabelas do DynamoDB usando modo de capacidade sob demanda automaticamente adaptam-se ao volume de tráfego da sua aplicação. Novas tabelas sob demanda poderão comportar até 4 mil gravações por segundo e 12 mil leituras por segundo. O modo de capacidade sob demanda acomoda instantaneamente até o dobro do pico de tráfego anterior em uma tabela. Por exemplo, suponha que o padrão de tráfego da aplicação varie entre 25 mil e 50 mil leituras altamente consistentes por segundo, e 50 mil leituras por segundo seja o pico de tráfego atingido anteriormente. O modo de capacidade sob demanda atende instantaneamente ao tráfego continuado de até cem mil leituras por segundo. Se a aplicação comportar o tráfego de cem mil leituras por segundo, esse pico vai se tornar o novo pico anterior. Esse pico anterior possibilita que o tráfego subsequente alcance até duzentas mil leituras por segundo.

Se a workload gerar mais do que o dobro do pico anterior em uma tabela, o DynamoDB alocará automaticamente uma capacidade maior à medida que o volume de tráfego aumentar. Essa alocação de capacidade ajuda a garantir que a workload não sofra controle de utilização. No entanto, pode ocorrer controle de utilização se você exceder o dobro de seu pico anterior dentro de 30 minutos. Por exemplo, suponha que o padrão de tráfego da aplicação varie entre 25 mil e 50 mil leituras altamente consistentes por segundo, e 50 mil leituras por segundo sejam o pico de tráfego atingido anteriormente. Recomendamos pré-preparar a tabela ou espaçar o aumento do tráfego por pelo menos trinta minutos antes de gerar mais de cem mil leituras por segundo. Para ter mais informações sobre pré-preparação, consulte Noções básicas sobre o throughput a quente do DynamoDB.

O DynamoDB não impõe a restrição de controle de utilização de 30 minutos se o pico de tráfego da workload permanecer dentro do dobro do pico anterior. Se o pico de tráfego exceder o dobro do pico, garanta que esse aumento ocorra 30 minutos depois da última vez que você atingiu o pico.

Throughput máximo do DynamoDB para tabelas sob demanda

Em relação a tabelas sob demanda, é possível especificar o throughput máximo de leitura ou de gravação (ou de ambas) por segundo em tabelas individuais e índices secundários globais (GSIs) associados. Especificar um throughput máximo sob demanda ajuda a manter o uso e os custos por tabela limitados. Por padrão, as configurações de throughput máximo não se aplicam e sua taxa de throughput sob demanda é limitada pelas cotas de serviço da AWS de 40.000 referente a throughput de leitura e gravação em nível de tabela para todas as tabelas na conta. Se necessário, é possível solicitar um aumento na cota do serviço.

Ao configurar o throughput máximo para uma tabela sob demanda, as solicitações de throughput que excederem a quantidade máxima especificada terão controle de utilização. É possível modificar as configurações de throughput por tabela a qualquer momento, com base nos requisitos da aplicação.

Veja a seguir alguns casos de uso comuns que podem se beneficiar do uso do throughput máximo para tabelas sob demanda:

- Otimização do custo de throughput: o uso de throughput máximo para tabelas sob demanda oferece um nível adicional de previsibilidade e capacidade de gerenciamento de custos. Além disso, oferece maior flexibilidade para usar o modo sob demanda para comportar workloads com diferentes padrões de tráfego e orçamentos.
- Proteção contra uso excessivo: ao definir o throughput máximo, é possível evitar um aumento acidental no consumo de leitura ou de gravação, que pode surgir de código não otimizado ou de processos não autorizados, em uma tabela sob demanda. Essa configuração por tabela pode proteger as organizações contra o consumo excessivo de recursos em determinado período.
- Proteger os serviços subsequentes: uma aplicação do cliente pode incluir tecnologias com e sem servidor. A parte sem servidor da arquitetura pode ser escalada rapidamente para atender à demanda. No entanto, os componentes subsequentes com capacidades fixas podem ficar sobrecarregados. A implementação de configurações de throughput máximo para tabelas sob demanda pode impedir a propagação de um grande volume de eventos para vários componentes subsequentes com efeitos secundários inesperados.

É possível configurar o throughput máximo para o modo sob demanda para tabelas de região única novas e existentes, além de tabelas globais e GSIs. Também é possível configurar o throughput máximo durante a restauração da tabela e a importação de dados dos fluxos de trabalho do Amazon S3.

É possível especificar as configurações de throughput máximo para tabelas sob demanda usando o console do DynamoDB, a AWS CLI, o AWS CloudFormation ou a API do DynamoDB.

> **Note**
>
> O throughput máximo de uma tabela sob demanda é aplicado de acordo com o melhor esforço e deve ser considerado como alvo e não como limite máximo garantido de solicitações. A workload pode exceder temporariamente o throughput máximo especificado devido à capacidade de expansão. Em alguns casos, o DynamoDB usa a capacidade de expansão para atender a leituras ou gravações que ultrapassam as configurações de throughput máximo da tabela. Com a capacidade de expansão, as solicitações de leitura ou gravação inesperadas podem ter êxito onde, de outra forma, seriam limitadas.

Tópicos

- Considerações ao usar o throughput máximo para o modo sob demanda
- Controle de utilização de solicitações e métricas do CloudWatch

Considerações ao usar o throughput máximo para o modo sob demanda

Ao usar o throughput máximo para tabelas no modo sob demanda, as seguintes considerações se aplicam:

- É possível definir de forma independente o throughput máximo de leituras e gravações de qualquer tabela sob demanda ou índice secundário global individual dentro dessa tabela para ajustar a abordagem com base em requisitos específicos.
- Você pode usar o Amazon CloudWatch para monitorar e entender as métricas de uso por tabela do DynamoDB e para determinar as configurações de throughput máximo apropriadas para o modo sob demanda. Para obter mais informações, consulte Métricas e dimensões do DynamoDB.
- Ao especificar as configurações de throughput máximo de leitura ou de gravação (ou de ambas) em uma réplica de tabela global, as mesmas configurações de throughput máximo são aplicadas automaticamente a todas as tabelas de réplica. É importante que as tabelas de réplica e os índices secundários em uma tabela global tenham configurações idênticas de throughput para garantir a replicação apropriada dos dados. Para obter mais informações, consulte Melhores práticas para tabelas globais.

- O menor throughput máximo de leitura ou de gravação que você pode especificar é de uma unidade de solicitação por segundo.
- O throughput máximo especificado deve ser menor do que a cota de throughput padrão que está disponível para qualquer tabela sob demanda ou índice secundário global individual dentro dessa tabela.

Controle de utilização de solicitações e métricas do CloudWatch

Se a aplicação exceder o throughput máximo de leitura ou de gravação definido na tabela sob demanda, o DynamoDB começará a controlar a utilização dessas solicitações. Quando o DynamoDB limita uma leitura ou gravação, ele retorna uma `ThrottlingException` para o chamador. Depois, você poderá tomar as medidas apropriadas, se necessário. Por exemplo, é possível aumentar ou desabilitar a configuração de throughput máximo da tabela ou aguardar um curto intervalo antes de repetir a solicitação.

Para simplificar o monitoramento do throughput máximo configurado para uma tabela ou um índice secundário global, o CloudWatch fornece as seguintes métricas: OnDemandMaxReadRequestUnits e OnDemandMaxWriteRequestUnits.

Modo de capacidade provisionada do DynamoDB

Ao criar uma tabela provisionada no DynamoDB, é necessário especificar a capacidade de throughput provisionado. Essa é a quantidade de throughput de leitura e de gravação que a tabela pode comportar. Você será cobrado com base na capacidade de leitura e de gravação por hora provisionada, não na quantidade de capacidade provisionada que você realmente consumiu.

À medida que os dados da aplicação e os requisitos de acesso mudarem, talvez seja necessário ajustar as configurações de throughput da tabela. Você pode usar Auto Scaling para ajustar a capacidade provisionada da tabela automaticamente em resposta às alterações de tráfego. O ajuste de escala automático do DynamoDB usa uma política de ajuste de escala no Application Auto Scaling. Para configurar o ajuste de escala automático no DynamoDB, você define os níveis mínimo e máximo de capacidade de leitura e de gravação, além da porcentagem de utilização pretendida. O Application Auto Scaling cria e gerencia os alarmes do CloudWatch que acionam eventos de escalabilidade quando a métrica se desvia do destino. O ajuste de escala automático monitora a atividade da tabela e ajusta suas configurações de capacidade para cima ou para baixo com base em limites predefinidos. O ajuste de escala automático é acionado quando a capacidade consumida ultrapassa a meta de utilização configurada em dois minutos consecutivos. Os alarmes

do CloudWatch podem ter um pequeno atraso de até alguns minutos antes de acionar o ajuste de escala automático. Para obter mais informações, consulte Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB.

Se você estiver usando o Auto Scaling do DynamoDB as configurações de throughput serão automaticamente ajustadas em resposta às workloads reais. Também é possível usar a operação UpdateTable para ajustar manualmente a capacidade de throughput da tabela. Por exemplo, você pode optar por fazer isso se precisar fazer o carregamento em massa de dados de um datastore existente para sua nova tabela do DynamoDB. Você pode criar a tabela com uma configuração alta de throughput de gravação e, em seguida, reduzir essa configuração depois que o carregamento de dados em massa for concluído.

> **Note**
>
> Por padrão, o DynamoDB protege você contra o uso indesejado e descontrolado. Para escalar além dos limites de throughput de leitura e gravação em nível de tabela de 40.000 para todas as tabelas em sua conta, você pode solicitar um aumento dessa cota. As solicitações de throughput que excedem a cota de throughput de tabela padrão têm controle de utilização. Para obter mais informações, consulte Cotas padrão de throughput.

É possível alternar as tabelas do modo de capacidade provisionada para o modo sob demanda até quatro vezes em um período contínuo de 24 horas. É possível alternar as tabelas do modo sob demanda para o modo de capacidade provisionada a qualquer momento.

Para ter mais informações sobre como alternar entre os modos de capacidade de leitura e de gravação, consulte Considerações ao alternar os modos de capacidade no DynamoDB.

Tópicos

- Unidades de capacidade de leitura e unidades de capacidade de gravação
- Escolher as configurações iniciais de throughput
- Ajuste de escala automático do DynamoDB
- Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB
- Capacidade reservada do DynamoDB

Unidades de capacidade de leitura e unidades de capacidade de gravação

Para tabelas do modo provisionado, você precisa especificar os requisitos de throughput em termos de unidades de capacidade. Essas unidades representam o volume de dados que a aplicação precisa ler ou gravar por segundo. Você pode modificar essas configurações mais tarde, se necessário, ou permitir que o Auto Scaling do DynamoDB os modifique automaticamente.

Em relação a um item de até 4 KB, uma unidade de capacidade de leitura (RCU) representa uma operação de leitura altamente consistente por segundo ou duas operações de leitura final consistente por segundo. Para ter mais informações sobre os modelos de consistência de leitura do DynamoDB, consulte Consistência de leitura do DynamoDB.

Uma unidade de capacidade de gravação (WCU) representa uma gravação por segundo para um item de até 1 KB. Para ter mais informações sobre as diferentes operações de gravação e de leitura, consulte Operações de leitura e de gravação do DynamoDB.

Escolher as configurações iniciais de throughput

Cada aplicação tem diferentes requisitos para operações de leitura e gravação em um banco de dados. Quando estiver determinando as configurações iniciais de throughput de uma tabela do DynamoDB, leve em conta o seguinte:

- Taxas esperadas de solicitações de leitura e de gravação: você precisa calcular o número de leituras e de gravações que precisa realizar por segundo.
- Tamanhos de item: alguns itens são pequenos o suficiente para que possam ser lidos ou gravados usando uma única unidade de capacidade. Itens maiores exigem várias unidades de capacidade. Ao estimar o tamanho médio dos itens que estarão na tabela, você poderá especificar configurações precisas para o throughput provisionado.
- Requisitos da consistência de leitura: as unidades de capacidade de leitura baseiam-se em operações de leitura altamente consistente, que consomem duas vezes mais recursos de banco de dados que as leituras finais consistentes. Você deve determinar se o seu aplicativo requer leituras altamente consistentes, ou se ele pode ignorar essa exigência e executar leituras finais consistentes em vez disso. As operações de leitura no DynamoDB são finais consistentes por padrão. É possível solicitar leituras altamente consistentes para essas operações, se necessário.

Por exemplo, vamos supor que você queira ler oitenta itens por segundo de uma tabela. O tamanho desses itens é 3 KB, e você deseja leituras altamente consistentes. Nesse caso, cada leitura requer

uma unidade de capacidade de leitura provisionada. Para determinar esse número, divida o tamanho do item da operação por 4 KB. Depois, arredonde o resultado para o número inteiro mais próximo, conforme mostrado no seguinte exemplo:

- 3 KB/4 KB = 0,75 ou 1 unidade de capacidade de leitura

Portanto, para ler oitenta itens por segundo de uma tabela, defina o throughput de leitura provisionado da tabela como oitenta unidades de capacidade de leitura, conforme mostrado no seguinte exemplo:

- 1 unidade de capacidade de leitura por item × 80 leituras por segundo = 80 unidades de capacidade de leitura

Agora, vamos supor que você queira gravar cem itens por segundo na tabela e que cada item tenha 512 bytes. Nesse caso, cada gravação requer uma unidade de capacidade de gravação provisionada. Para determinar esse número, divida o tamanho do item da operação por 1 KB. Depois, arredonde o resultado para o número inteiro mais próximo, conforme mostrado no seguinte exemplo:

- 512 bytes/1 KB = 0,5 ou 1 unidade de capacidade de gravação.

Para gravar cem itens por segundo na tabela, defina o throughput de gravação provisionado da tabela como cem unidades de capacidade de gravação:

- 1 unidade de capacidade de gravação por item × 100 gravações por segundo = 100 unidades de capacidade de gravação

Ajuste de escala automático do DynamoDB

O ajuste de escala automático do DynamoDB gerencia ativamente a capacidade de throughput de tabelas e de índices secundários globais. Com o Auto Scaling, você define um intervalo (limites superior e inferior) para unidades de capacidade de leitura e gravação. Você também define um percentual de utilização-alvo dentro desse intervalo. O Auto Scaling do DynamoDB procura manter sua utilização alvo, mesmo que a workload da sua aplicação aumente ou diminua.

Com o Auto Scaling do DynamoDB, uma tabela ou um índice secundário global pode aumentar sua capacidade provisionada de leitura e gravação para lidar com aumentos repentinos no tráfego, sem

a controle de utilização de solicitações. Quando a workload diminuir, o Auto Scaling do DynamoDB diminuirá o throughput para que você não precise pagar por uma capacidade provisionada não utilizada.

> **Note**
>
> Se você usar o AWS Management Console para criar uma tabela ou um índice secundário global, o Auto Scaling do DynamoDB será habilitado por padrão.
> É possível gerenciar as configurações de Auto Scaling a qualquer momento usando o console, a AWS CLI ou um dos AWS SDKs. Para obter mais informações, consulte Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB.

Taxa de utilização

A taxa de utilização pode ajudar você a determinar se está sobrecarregando a capacidade de provisionamento. Nesse caso, você deve reduzir a capacidade da tabela para reduzir os custos. Inversamente, também pode ajudar você a determinar se está com pouca capacidade de provisionamento. Nesse caso, é necessário aumentar a capacidade da tabela para evitar um possível controle de utilização de solicitações durante instâncias de alto tráfego inesperado. Para ter mais informações, consulte Amazon DynamoDB auto scaling: Performance and cost optimization at any scale.

Se você estiver usando o ajuste de escala automático do DynamoDB, também precisará definir uma meta de porcentagem de utilização. O ajuste de escala automático usará essa porcentagem como meta para ajustar a capacidade para cima ou para baixo. Recomendamos definir a meta de utilização como 70%. Para obter mais informações, consulte Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB.

Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB

Muitas workloads de banco de dados são cíclicas por natureza, enquanto outras são difíceis de prever com antecedência. Por exemplo, considere um aplicativo de rede social na qual a maioria dos usuários está ativa durante o horário diurno. O banco de dados deve ser capaz de lidar com a atividade durante o dia, mas não há necessidade dos mesmos níveis de throughput à noite. Outro exemplo: considere um novo aplicativo de jogos para celular que está sendo adotado de maneira

rápida e inesperada. Se o jogo se tornar muito comum, talvez ele exceda os recursos de banco de dados disponíveis, resultando em performance lenta e clientes insatisfeitos. Esses tipos de workloads muitas vezes exigem intervenção manual para dimensionar recursos de banco de dados, aumentando-os ou diminuindo-os em resposta a diferentes níveis de uso.

O Auto Scaling do Amazon DynamoDB usa o serviço AWS Application Auto Scaling para ajustar dinamicamente a capacidade de throughput provisionado em seu nome em resposta aos padrões de tráfego reais. Isso permite que uma tabela ou um índice secundário global (GSI) aumente a capacidade provisionada de leitura e de gravação para processar aumentos repentinos no tráfego, sem controle de utilização. Quando a workload diminuir, o Application Auto Scaling diminuirá o throughput para que você não precise pagar por uma capacidade provisionada não utilizada.

> **Note**
>
> Se você usar o AWS Management Console para criar uma tabela ou um índice secundário global, o Auto Scaling do DynamoDB será habilitado por padrão. É possível modificar as configurações de Auto Scaling a qualquer momento. Para obter mais informações, consulte Usar o AWS Management Console com o Auto Scaling do DynamoDB.
> Ao remover manualmente uma tabela ou uma réplica de tabela global, você não remove automaticamente as metas escaláveis associadas, as políticas de escalabilidade nem os alarmes do CloudWatch.

Com o Application Auto Scaling, você cria um política de escalabilidade para uma tabela ou um índice secundário global. A política de escalabilidade especifica se você deseja dimensionar a capacidade de leitura ou a capacidade de gravação (ou ambas), bem como as configurações mínimas e máximas de unidades de capacidade provisionadas para a tabela ou o índice.

A política de escalabilidade também contém uma utilização pretendida: o percentual de throughput provisionado consumida em um determinado ponto no tempo. O Application Auto Scaling usa um algoritmo de rastreamento para ajustar o throughput provisionado da tabela (ou índice) para cima ou para baixo em resposta a workloads reais para que a utilização da capacidade real permaneça em ou perto de sua utilização pretendida.

As saídas do DynamoDB consumiram o throughput provisionado por períodos de um minuto. O ajuste de escala automático é acionado quando a capacidade consumida ultrapassa a meta de utilização configurada em dois minutos consecutivos. Os alarmes do CloudWatch podem ter um pequeno atraso de até alguns minutos antes de acionar o ajuste de escala automático. Esse atraso

garante uma avaliação precisa da métrica do CloudWatch. Se os picos de throughput consumidos tiverem mais de um minuto de intervalo, o ajuste de escala automático poderá não ser acionado. Da mesma forma, um evento de redução de escala verticalmente pode ocorrer quando 15 pontos de dados consecutivos estão abaixo da meta de utilização. Nos dois casos, depois que o ajuste de escala automático é acionado, a API UpdateTable é invocada. Depois, leva alguns minutos para atualizar a capacidade provisionada da tabela ou do índice. Durante esse período, todas as solicitações que excederem a capacidade provisionada anterior das tabelas terão controle de utilização.

> **Important**
>
> Não é possível ajustar o número de pontos de dados a serem violados antes do acionamento do alarme subjacente (embora o número atual possa mudar no futuro).

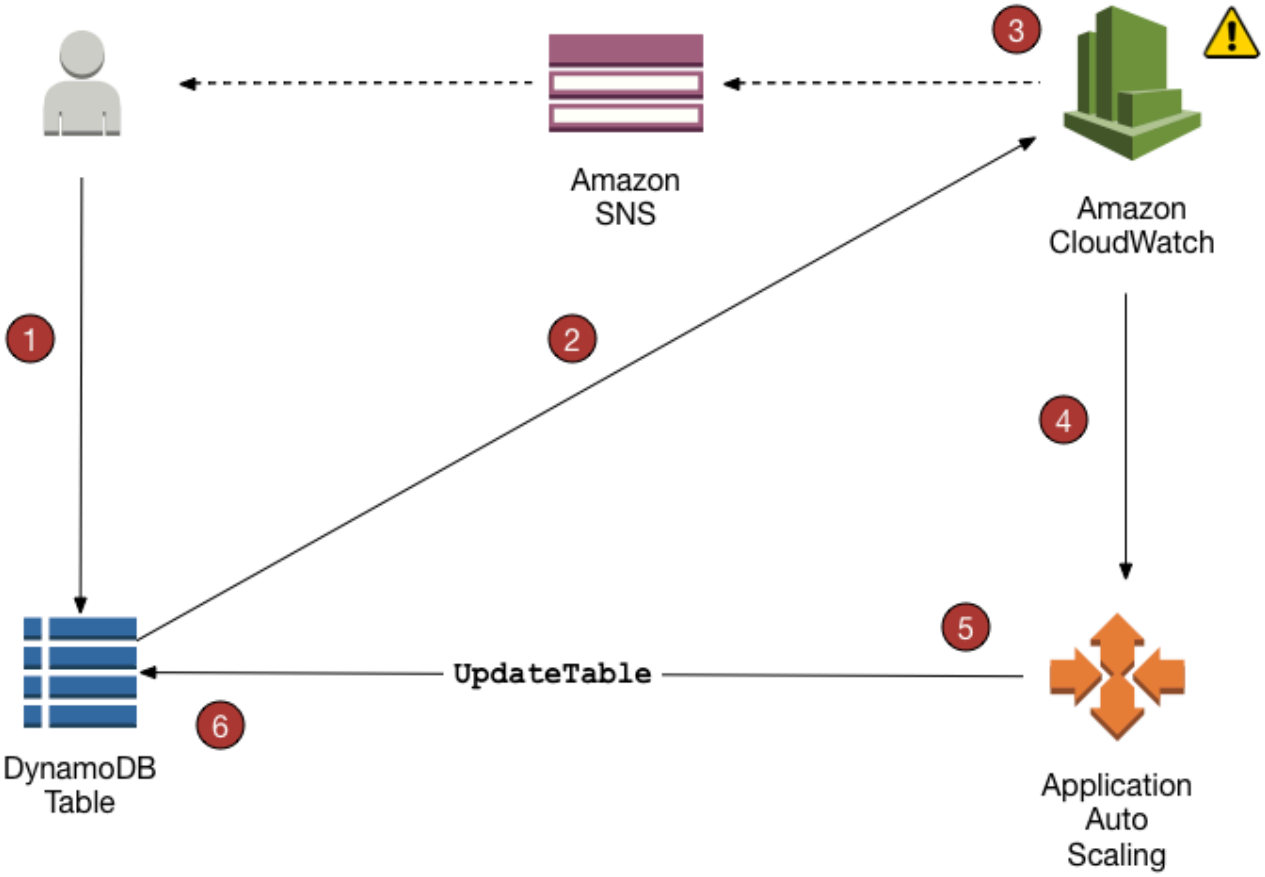
É possível definir os valores de utilização do destino de Auto Scaling entre 20% e 90% como sua capacidade de gravação e leitura.

> **Note**
>
> Além de tabelas, o Auto Scaling do DynamoDB também oferece suporte a índices secundários globais. Cada índice secundário global tem sua própria capacidade de throughput provisionado separadamente daquela da tabela-base. Quando você cria uma política de escalabilidade para um índice secundário global, o Application Auto Scaling ajusta as configurações de throughput provisionado do índice para garantir que sua utilização real permaneça em ou perto da proporção de utilização desejada.

Como o Auto Scaling do DynamoDB funciona

> **Note**
>
> Para começar a trabalhar rapidamente com o Auto Scaling do DynamoDB, consulte Usar o AWS Management Console com o Auto Scaling do DynamoDB.

O diagrama a seguir fornece uma visão geral de alto nível de como o Auto Scaling do DynamoDB gerencia a capacidade de throughput de uma tabela.



As etapas a seguir resumem o processo de Auto Scaling, conforme mostrado no diagrama anterior:

1. Você cria uma política do Application Auto Scaling para sua tabela do DynamoDB.
2. O DynamoDB publica métricas de capacidade consumida no Amazon CloudWatch.
3. Se a capacidade consumida da tabela exceder sua utilização pretendida (ou cair abaixo do valor desejado) por um período específico, o Amazon CloudWatch disparará um alarme. Você pode visualizar o alarme no console receber notificações por SMS usando o Amazon Simple Notification Service (Amazon SNS).
4. O alarme do CloudWatch invoca o Application Auto Scaling para avaliar sua política de escalabilidade.
5. O Application Auto Scaling emite uma solicitação `UpdateTable` para ajustar o throughput provisionado da sua tabela.
6. O DynamoDB processa a solicitação `UpdateTable`, aumentando (ou diminuindo) dinamicamente a capacidade de throughput provisionado da tabela de forma que ela se aproxime da sua utilização pretendida.

Para entender como o Auto Scaling do DynamoDB funciona, suponha que você tenha uma tabela chamada `ProductCatalog`. A tabela é raramente carregada em massa com dados e, por isso, não provoca muita atividade de gravação. No entanto, ela é submetida a um alto grau de atividades de leitura, que variam ao longo do tempo. Ao monitorar as métricas do Amazon CloudWatch para o `ProductCatalog`, você determina que a tabela requer 1.200 unidades de capacidade de leitura (para evitar que o controle de utilização do DynamoDB leia solicitações durante picos de atividades). Você também determina que o `ProductCatalog` requer no mínimo 150 unidades de capacidade de leitura quando o tráfego de leitura está em seu ponto mais baixo. Para obter mais informações sobre como evitar o controle de utilização, consulte Solução de problemas de controle de utilização no Amazon DynamoDB.

No intervalo de 150 a 1.200 unidades de capacidade de leitura, você decide que uma utilização de destino de 70% seria apropriada para a tabela `ProductCatalog`. A utilização pretendida é a proporção entre unidades de capacidade consumidas e unidades de capacidade provisionadas, expressa como um percentual. O Application Auto Scaling usa seu algoritmo de rastreamento de capacidade pretendida para garantir que a capacidade de leitura provisionada de `ProductCatalog` seja ajustada conforme o necessário para que a utilização permaneça em ou perto de 70%.

> **Note**
>
> A autoescalabilidade do DynamoDB modifica as configurações de throughput provisionado somente quando a workload real permanece elevada ou baixa por um período constante de vários minutos. O algoritmo de rastreamento de alvo do Application Auto Scaling procura manter a utilização pretendida em ou perto do seu valor escolhido em longo prazo.
> Picos de atividade súbitos de curta duração são acomodados pela capacidade de expansão interna da tabela. Para obter mais informações, consulte Capacidade de expansão.

Para habilitar o Auto Scaling do DynamoDB para a tabela `ProductCatalog`, você cria uma política de escalabilidade. A política especifica o seguinte:

- A tabela ou o índice secundário global que você deseja gerenciar
- Qual tipo de capacidade gerenciar (capacidade de leitura ou capacidade de gravação)
- Os limites superior e inferior das configurações de throughput provisionado
- Utilização de destino

Quando você criar uma política de escalabilidade, o Application Auto Scaling cria um par de alarmes do Amazon CloudWatch em seu nome. Cada par representa os limites superiores e inferiores das configurações de throughput provisionado. Esses alarmes do CloudWatch são disparados quando a utilização real da tabela se desvia da utilização pretendida por um período prolongado.

Quando um dos alarmes do CloudWatch for disparado, o Amazon SNS enviará uma notificação (caso você tenha habilitado esse recurso). O alarme do CloudWatch chama o Application Auto Scaling, que, por sua vez, notifica o DynamoDB para ajustar a capacidade provisionada da tabela `ProductCatalog` para cima ou para baixo conforme apropriado.

Durante um evento de ajuste de escala, o AWS Config é cobrado por item de configuração registrado. Quando ocorre um evento de ajuste de escala, quatro alarmes do CloudWatch são criados para cada evento de ajuste de escala automático de leitura e gravação: alarmes ProvisionedCapacity (ProvisionedCapacityLow e ProvisionedCapacityHigh) e alarmes ConsumedCapacity (AlarmHigh e AlarmLow). Isso soma um total de oito alarmes. Portanto, o AWS Config registra oito itens de configuração para cada evento de ajuste de escala.

> **Note**
>
> Também é possível programar o ajuste de escala do DynamoDB para que ele ocorra em determinados horários. Conheça as etapas básicas aqui.

Observações de uso

Antes de começar a usar o Auto Scaling do DynamoDB, você deve estar ciente do seguinte:

- O Auto Scaling do DynamoDB pode aumentar a capacidade de leitura ou gravação sempre que necessário, de acordo com a sua política de Auto Scaling. Todas as cotas do DynamoDB permanecem em vigor, conforme descrito em Cotas no Amazon DynamoDB.
- O Auto Scaling do DynamoDB não impede a modificação manual de configurações de throughput provisionado. Esses ajustes manuais não afetam alarmes existentes do CloudWatch que estejam relacionados ao Auto Scaling do DynamoDB.
- Se você habilitar o Auto Scaling do DynamoDB para uma tabela que tenha um ou mais índices secundários globais, é altamente recomendável que você também aplique o Auto Scaling uniformemente e esses índices. Isso ajudará a garantir uma performance melhor para gravações e leituras de tabelas e ajudará a evitar o controle de utilização. É possível habilitar a autoescalabilidade selecionando Apply same settings to global secondary indexes (Aplicar as

mesmas configurações a índices secundários globais) no AWS Management Console. Para obter mais informações, consulte Habilitar o Auto Scaling do DynamoDB em tabelas existentes.

- Ao remover manualmente uma tabela ou uma réplica de tabela global, você não remove automaticamente as metas escaláveis associadas, as políticas de dimensionamento nem os alarmes do CloudWatch.
- Ao criar um GSI para uma tabela existente, o Auto Scaling não está habilitado para GSI. Você terá que gerenciar manualmente a capacidade enquanto o GSI está sendo compilado. Quando o provisionamento no GSI for concluído e atingir o status ativo, a autoescalabilidade funcionará normalmente.

Usar o AWS Management Console com o Auto Scaling do DynamoDB

Quando você usa o AWS Management Console para criar uma nova tabela, o Auto Scaling do Amazon DynamoDB é habilitado para essa tabela por padrão. Você também pode usar o console para habilitar o Auto Scaling de tabelas existentes, modificar as configurações de Auto Scaling ou desabilitar o Auto Scaling.

> **Note**
>
> Para obter recursos mais avançados, como a definição de redução e expansão de períodos de cooldown, use a AWS Command Line Interface (AWS CLI) para gerenciar o dimensionamento automático do DynamoDB. Para obter mais informações, consulte Usar a AWS CLI para gerenciar o Auto Scaling do Amazon DynamoDB.

Tópicos

- Antes de começar: concessão de permissões de usuário ao Auto Scaling do DynamoDB
- Criar uma nova tabela com Auto Scaling habilitado
- Habilitar o Auto Scaling do DynamoDB em tabelas existentes
- Visualizar atividades de Auto Scaling no console
- Modificar ou desabilitar configurações de Auto Scaling do DynamoDB

Antes de começar: concessão de permissões de usuário ao Auto Scaling do DynamoDB

No AWS Identity and Access Management (IAM), a política `DynamoDBFullAccess`, gerenciada pela AWS, fornece as permissões necessárias para usar o console do DynamoDB. No entanto, para a autoescalabilidade do DynamoDB, os usuários precisam de permissões adicionais.

> **Important**
>
> Para excluir uma tabela habilitada para ajuste de escala automático, são necessárias permissões `application-autoscaling:*`. A política `DynamoDBFullAccess`, gerenciada pela AWS, inclui essas permissões.

Para configurar um usuário para acesso ao console do DynamoDB e autoescalabilidade do DynamoDB, crie um perfil e adicione a política AmazonDynamoDBFullAccess a esse perfil. Depois, atribua o perfil a um usuário.

Criar uma nova tabela com Auto Scaling habilitado

> **Note**
>
> A autoescalabilidade do DynamoDB requer a presença de um perfil vinculado ao serviço (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) que realize ações de auescalabilidade em seu nome. Esta função é criada automaticamente para você. Para ter mais informações, consulte Service-linked roles for Application Auto Scaling, no Manual do usuário do Application Auto Scaling.

Para criar uma nova tabela com Auto Scaling habilitada

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. Escolha Create table.
3. Na página Criar tabela, insira o Nome de tabela e os detalhes da chave primária.
4. Se você escolher Configurações padrão, o ajuste de escala automático será ativado na nova tabela.

   Caso contrário, selecione Personalizar configurações e faça o seguinte para especificar configurações personalizadas para a tabela:

a. Em Classe de tabela, mantenha a seleção padrão do DynamoDB Standard.

b. Em Configurações da capacidade de leitura/gravação, mantenha a seleção padrão de Provisionado e faça o seguinte:

   i. Em Capacidade de leitura, verifique se o Ajuste de escala automático está definido como Ativado.

   ii. Em Capacidade de gravação, verifique se o Ajuste de escala automático está definido como Ativado.

   iii. Em Capacidade de leitura e Capacidade de gravação, defina a política de escalabilidade desejada para a tabela e, opcionalmente, todos os índices secundários globais da tabela.

   - Unidades de capacidade mínima: insira o limite inferior para o intervalo de autoescalabilidade.
   - Unidades de capacidade máxima: insira o limite superior para o intervalo de autoescalabilidade.
   - Utilização pretendida: insira a porcentagem de utilização pretendida para a tabela.

> **Note**
>
> Se você criar um índice secundário global para a nova tabela, a capacidade do índice no momento da criação será a mesma da capacidade da tabela base. Você pode alterar a capacidade do índice nas configurações da tabela depois de criar a tabela.

5. Escolha Create table. Isso cria a tabela com os parâmetros de ajuste de escala automático especificados.

Habilitar o Auto Scaling do DynamoDB em tabelas existentes

> **Note**
>
> A autoescalabilidade do DynamoDB requer a presença de um perfil vinculado ao serviço (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) que realize ações de auescalabilidade em seu nome. Esta função é criada automaticamente para você. Para

obter mais informações, consulte Funções vinculadas ao serviço para o Application Auto Scaling.

Para habilitar o Auto Scaling do DynamoDB para uma tabela existente

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Selecione a tabela na qual você deseja habilitar o ajuste de escala automático e, depois, faça o seguinte:

   a. Selecione a guia Configurações adicionais.

   b. Na seção Capacidade de leitura/gravação, selecione Editar.

   c. Na seção Modo de capacidade, selecione Provisionada.

   d. Na seção Table capacity (Capacidade da tabela), deixe Auto Scaling (Autoescalabilidade) no modo On (Ativado) para Read capacity (Capacidade de leitura), Write capacity (Capacidade de gravação) ou ambos. Para cada um deles, defina a política de escalabilidade desejada para a tabela e, opcionalmente, todos os índices secundários globais da tabela.

      - Unidades de capacidade mínima: insira o limite inferior para o intervalo de autoescalabilidade.
      - Unidades de capacidade máxima: insira o limite superior para o intervalo de autoescalabilidade.
      - Utilização pretendida: insira a porcentagem de utilização pretendida para a tabela.
      - Usar as mesmas configurações de capacidade de leitura/gravação para todos os índices secundários globais: escolha se os índices secundários globais devem usar a mesma política de autoescalabilidade que a tabela de base.

      > **Note**
      >
      > Para obter uma melhor performance, recomendamos habilitar Use the same read/write capacity settings for all global secondary indexes (Usar as mesmas configurações de capacidade de leitura/gravação para todos os índices secundários globais). Essa opção permite que o Auto Scaling do DynamoDB dimensione uniformemente todos os índices secundários globais na tabela-base.

Isso inclui índices secundários globai existentes e quaisquer outros que você crie no futuro para essa tabela.
Com essa opção habilitada, não é possível definir uma política de escalabilidade em um índice secundário global individual.

4. Quando estiver satisfeito com as configurações, clique em Salvar.

Visualizar atividades de Auto Scaling no console

À medida que a sua aplicação direciona tráfego de leitura e gravação para a sua tabela, o Auto Scaling do DynamoDB modifica dinamicamente as configurações de throughput da tabela. O Amazon CloudWatch acompanha a capacidade provisionada e consumida, eventos limitados, latência e outras métricas para todas as tabelas do DynamoDB e índices secundários.

Para visualizar essas métricas no console do DynamoDB, escolha a tabela com a qual você deseja trabalhar e selecione a guia Monitor. Para criar uma visualização personalizável das métricas de tabela, selecione View all in CloudWatch (Visualizar tudo no CloudWatch).

Modificar ou desabilitar configurações de Auto Scaling do DynamoDB

É possível usar o AWS Management Console para modificar configurações de Auto Scaling do DynamoDB. Para fazer isso, vá até a guia Configurações adicionais referente à sua tabela e selecione Editar na seção Capacidade de leitura/gravação. Para ter mais informações sobre essas configurações, consulte Habilitar o Auto Scaling do DynamoDB em tabelas existentes.

Usar a AWS CLI para gerenciar o Auto Scaling do Amazon DynamoDB

Em vez de usar o AWS Management Console, você pode usar a AWS Command Line Interface (AWS CLI) para gerenciar o Auto Scaling do Amazon DynamoDB. O tutorial desta seção demonstra como instalar e configurar a AWS CLI para gerenciar o Auto Scaling do DynamoDB. Neste tutorial, você faz o seguinte:

- Crie uma tabela do DynamoDB chamada `TestTable`. As configurações de throughput iniciais são 5 unidades de capacidade de leitura e 5 unidades de capacidade de gravação.
- Crie uma política do Application Auto Scaling para `TestTable`. A política procura manter uma taxa de destino de 50% entre a capacidade de gravação consumida e a capacidade de gravação provisionada. O intervalo dessa métrica é entre 5 e 10 unidades de capacidade de gravação. (O Application Auto Scaling não tem permissão para ajustar o throughput além desse intervalo.)

- Execute um programa Python para direcionar o tráfego de gravação para `TestTable`. Quando a taxa pretendida ultrapassar 50% por período contínuo, o Application Auto Scaling notificará o DynamoDB para aumentar o throughput de `TestTable` para que os 50% da utilização pretendida possam ser mantidos.
- Verifique se o DynamoDB ajustou com sucesso a capacidade de gravação provisionada de `TestTable`.

> **Note**
>
> Também é possível programar o ajuste de escala do DynamoDB para que ele ocorra em determinados horários. Conheça as etapas básicas aqui.

Tópicos

- Antes de começar
- Etapa 1: Crie uma tabela do DynamoDB
- Etapa 2: registrar uma capacidade pretendida escalável
- Etapa 3: criar uma política de dimensionamento
- Etapa 4: direcionar o tráfego de gravação para TestTable
- Etapa 5: visualizar ações do Application Auto Scaling
- Etapa 6 (opcional): limpar

Antes de começar

Antes de iniciar o tutorial, é necessário concluir as tarefas a seguir.

Instalar a AWS CLI

Caso ainda não tenha feito isso, você deve instalar e configurar a AWS CLI. Para fazer isso, siga estas instruções no Guia do usuário do AWS Command Line Interface:

- Instalar a AWS CLI
- Configurar a AWS CLI

Instalar o Python

Parte deste tutorial requer que você execute um programa Python (consulte Etapa 4: direcionar o tráfego de gravação para TestTable). Caso ainda não esteja instalado, você poderá fazer download do Python.

Etapa 1: Crie uma tabela do DynamoDB

Nesta etapa, use a AWS CLI para criar uma `TestTable`. A chave primária de `pk` consiste em (chave de partição) e `sk` (chave de classificação). Ambos os atributos são do tipo `Number`. As configurações de throughput iniciais são 5 unidades de capacidade de leitura e 5 unidades de capacidade de gravação.

1. Use o comando AWS CLI a seguir para criar a tabela:

```
aws dynamodb create-table \
    --table-name TestTable \
    --attribute-definitions \
        AttributeName=pk,AttributeType=N \
        AttributeName=sk,AttributeType=N \
    --key-schema \
        AttributeName=pk,KeyType=HASH \
        AttributeName=sk,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

2. Para verificar o status da tabela, use o comando a seguir.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

   A tabela está pronta para uso quando seu status é `ACTIVE`.

Etapa 2: registrar uma capacidade pretendida escalável

Agora você registrará a capacidade de gravação da tabela como uma capacidade pretendida escalável com o Application Auto Scaling. Isso permite que o Application Auto Scaling ajuste a capacidade de gravação provisionada de TestTable, mas apenas no intervalo de 5 a 10 unidades de capacidade.

> **Note**
>
> O Auto Scaling do DynamoDB requer a presença de uma função vinculada ao serviço (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) que realize ações de Auto Scaling em seu nome. Esta função é criada automaticamente para você. Para ter mais informações, consulte Funções vinculadas a serviço do Application Auto Scaling, no Guia do usuário do Application Auto Scaling.

1. Agora, insira o comando a seguir para registrar a capacidade pretendida.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

2. Para verificar o registro, use o comando a seguir:

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> Você também pode registrar um valor pretendido escalável em relação a um índice secundário global. Por exemplo, em um índice secundário global (“test-index”), o ID do recurso e os argumentos de dimensão escalável são atualizados adequadamente.
>
> ```
> aws application-autoscaling register-scalable-target \
>     --service-namespace dynamodb \
>     --resource-id "table/TestTable/index/test-index" \
>     --scalable-dimension "dynamodb:index:WriteCapacityUnits" \
>     --min-capacity 5 \
>     --max-capacity 10
> ```

Etapa 3: criar uma política de dimensionamento

Nesta etapa, você cria uma política de escalabilidade para `TestTable`. A política define os detalhes sob os quais o Application Auto Scaling pode ajustar o throughput provisionado da tabela e as ações que serão executadas quando ele fizer isso. Você associa essa política à capacidade pretendida que definiu na etapa anterior (unidades de capacidade de gravação da tabela `TestTable`).

A política contém os elementos a seguir:

- `PredefinedMetricSpecification`: a métrica que o Application Auto Scaling tem permissão para ajustar. Para o DynamoDB, os valores a seguir são válidos para `PredefinedMetricType`:
  - `DynamoDBReadCapacityUtilization`
  - `DynamoDBWriteCapacityUtilization`
- `ScaleOutCooldown`: a quantidade mínima de tempo (em segundos) entre cada evento do Application Auto Scaling que aumenta o throughput provisionado. Esse parâmetro permite que o Application Auto Scaling aumente o throughput de forma contínua, mas não agressivamente, em resposta às workloads do mundo real. A configuração padrão de `ScaleOutCooldown` é 0.
- `ScaleInCooldown`: a quantidade mínima de tempo (em segundos) entre cada evento do Application Auto Scaling que diminui o throughput provisionado. Esse parâmetro permite que o Application Auto Scaling diminua o throughput de forma gradual e previsível. A configuração padrão de `ScaleInCooldown` é 0.
- `TargetValue`: o Application Auto Scaling garante que o índice de capacidade consumida para capacidade provisionada permaneça nesse valor ou próximo a ele. Você define `TargetValue` como uma porcentagem.

> **Note**
>
> Para compreender melhor como o `TargetValue` funciona, suponha que você tenha uma tabela com uma configuração de throughput provisionado de 200 unidades de capacidade de gravação. Você decide criar uma política de dimensionamento para essa tabela, com um `TargetValue` de 70%.
>
> Agora, suponha que você comece a direcionar tráfego de gravação para a tabela de forma que o throughput de gravação real seja de 150 unidades de capacidade. A taxa consumida-para-provisionada agora é (150/200), ou 75%. Essa taxa excede o valor pretendido. Portanto, o Application Auto Scaling aumenta a capacidade de gravação provisionada para 215 de

modo que a taxa seja (150/215) ou 69,77% — tão próxima ao seu `TargetValue` quanto possível, mas sem excedê-lo.

Para `TestTable`, defina `TargetValue` como 50%. O Application Auto Scaling ajusta o throughput provisionado da tabela dentro do intervalo de 5 a 10 unidades de capacidade (consulte Etapa 2: registrar uma capacidade pretendida escalável) para que a taxa consumida-para-provisionada permaneça em 50% ou próxima a isso. Você define os valores de `ScaleOutCooldown` e `ScaleInCooldown` para 60 segundos.

1. Crie um arquivo denominado `scaling-policy.json` com o seguinte conteúdo:

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. Use o comando da AWS CLI a seguir para criar a política.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy" \
    --policy-type "TargetTrackingScaling" \
    --target-tracking-scaling-policy-configuration file://scaling-policy.json
```

3. Na saída, observe que o Application Auto Scaling criou dois alarmes do Amazon CloudWatch: um para o limite superior e outro para o limite inferior da faixa de escalabilidade prevista.
4. Use o comando da AWS CLI a seguir para visualizar mais detalhes sobre a política de escalabilidade.

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. Na saída, verifique se as configurações da política correspondem às suas especificações de Etapa 2: registrar uma capacidade pretendida escalável e Etapa 3: criar uma política de dimensionamento.

Etapa 4: direcionar o tráfego de gravação para TestTable

Agora, você pode testar a política de escalabilidade gravando dados em `TestTable`. Para fazer isso, execute um programa Python.

1. Crie um arquivo denominado `bulk-load-test-table.py` com o seguinte conteúdo:

```
import boto3
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table("TestTable")

filler = "x" * 100000

i = 0
while (i < 10):
    j = 0
    while (j < 10):
        print (i, j)

        table.put_item(
            Item={
                'pk':i,
                'sk':j,
                'filler':{"S":filler}
            }
        )
        j += 1
    i += 1
```

2. Para executar o programa, insira o comando a seguir.

```
python bulk-load-test-table.py
```

A capacidade de gravação provisionada de `TestTable` é muito baixa (5 unidades de capacidade de gravação), para que o programa pare ocasionalmente devido a limitações de gravação. Esse comportamento é esperado.

Deixe o programa continuar em execução enquanto você avança para a próxima etapa.

Etapa 5: visualizar ações do Application Auto Scaling

Nesta etapa, você visualiza as ações do Application Auto Scaling que são iniciadas em seu nome. Você também pode verificar se o Application Auto Scaling atualizou a capacidade de gravação provisionada de `TestTable`.

1. Insira o comando a seguir para visualizar as ações do Application Auto Scaling.

```
aws application-autoscaling describe-scaling-activities \
    --service-namespace dynamodb
```

Execute esse comando ocasionalmente, enquanto o programa Python está em execução. (Alguns minutos serão necessários antes que sua política de escalabilidade seja invocada.) Você deve finalmente ver a saída a seguir.

```
...
{
    "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
    "Description": "Setting write capacity units to 10.",
    "ResourceId": "table/TestTable",
    "ActivityId": "0cc6fb03-2a7c-4b51-b67f-217224c6b656",
    "StartTime": 1489088210.175,
    "ServiceNamespace": "dynamodb",
    "EndTime": 1489088246.85,
    "Cause": "monitor alarm AutoScaling-table/TestTable-
AlarmHigh-1bb3c8db-1b97-4353-baf1-4def76f4e1b9 in state ALARM triggered policy
 MyScalingPolicy",
    "StatusMessage": "Successfully set write capacity units to 10. Change
 successfully fulfilled by dynamodb.",
    "StatusCode": "Successful"
},
...
```

Isso indica que o Application Auto Scaling emitiu uma solicitação `UpdateTable` para o DynamoDB.

2. Digite o comando a seguir para verificar se o DynamoDB aumentou a capacidade de gravação da tabela.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

As `WriteCapacityUnits` devem ter sido dimensionadas de `5` para `10`.

Etapa 6 (opcional): limpar

Neste tutorial, você criou vários recursos. Você pode excluir esses recursos se eles não forem mais necessários.

1. Excluir a política de escalabilidade de `TestTable`.

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. Cancelar o registro de capacidade pretendida escalável.

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. Exclua a tabela `TestTable`.

```
aws dynamodb delete-table --table-name TestTable
```

Usar o AWS SDK para configurar o Auto Scaling em tabelas do Amazon DynamoDB

Além de usar o AWS Management Console e a AWS Command Line Interface (AWS CLI), você pode escrever aplicações que interagem com o Auto Scaling do Amazon DynamoDB. Esta seção contém dois programas Java que podem ser usados para testar essa funcionalidade:

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

Habilitação do Application Auto Scaling para uma tabela

O programa a seguir mostra um exemplo de configuração de uma política de Auto Scaling para uma tabela do DynamoDB (`TestTable`). Ele procede da seguinte maneira:

- O programa registra unidades de capacidade de gravação como um destino dimensionável para a `TestTable`. O intervalo dessa métrica é entre 5 e 10 unidades de capacidade de gravação.
- Depois que o destino escalável é criado, o programa cria uma configuração de rastreamento de destino. A política procura manter uma taxa de destino de 50% entre a capacidade de gravação consumida e a capacidade de gravação provisionada.
- Em seguida, o programa cria a política de escalabilidade com base na configuração de rastreamento de destino.

> **Note**
>
> Ao remover manualmente uma tabela ou uma réplica de tabela global, você não remove automaticamente os destinos escaláveis associados, políticas de escalabilidade nem alarmes do CloudWatch.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import software.amazon.awssdk.services.applicationautoscaling.model.PolicyType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PredefinedMetricSpecification;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PutScalingPolicyRequest;
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import software.amazon.awssdk.services.applicationautoscaling.model.ScalingPolicy;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import software.amazon.awssdk.services.applicationautoscaling.model.MetricType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.TargetTrackingScalingPolicyCon
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class EnableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <roleARN> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).
               roleARN - The ARN of the role that has ApplicationAutoScaling
 permissions.
               policyName - The name of the policy to create.

            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        System.out.println("This example registers an Amazon DynamoDB table, which
 is the resource to scale.");
        String tableId = args[0];
        String roleARN = args[1];
```

```
        String policyName = args[2];
        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        ApplicationAutoScalingClient appAutoScalingClient =
ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        registerScalableTarget(appAutoScalingClient, tableId, roleARN, ns,
tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        configureScalingPolicy(appAutoScalingClient, tableId, ns, tableWCUs,
policyName);
    }

    public static void registerScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, String roleARN, ServiceNamespace ns,
ScalableDimension tableWCUs) {
        try {
            RegisterScalableTargetRequest targetRequest =
RegisterScalableTargetRequest.builder()
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .roleARN(roleARN)
                .minCapacity(5)
                .maxCapacity(10)
                .build();

            appAutoScalingClient.registerScalableTarget(targetRequest);
            System.out.println("You have registered " + tableId);

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the target was created.
    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
DescribeScalableTargetsRequest.builder()
```

```
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    // Configure a scaling policy.
    public static void configureScalingPolicy(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs, String policyName) {
        // Check if the policy exists before creating a new one.
        DescribeScalingPoliciesResponse describeScalingPoliciesResponse =
appAutoScalingClient.describeScalingPolicies(DescribeScalingPoliciesRequest.builder()
            .serviceNamespace(ns)
            .resourceId(tableId)
            .scalableDimension(tableWCUs)
            .build());

        if (!describeScalingPoliciesResponse.scalingPolicies().isEmpty()) {
            // If policies exist, consider updating an existing policy instead of
creating a new one.
            System.out.println("Policy already exists. Consider updating it
instead.");
            List<ScalingPolicy> polList =
describeScalingPoliciesResponse.scalingPolicies();
            for (ScalingPolicy pol : polList) {
                System.out.println("Policy name:" +pol.policyName());
            }
        } else {
            // If no policies exist, proceed with creating a new policy.
            PredefinedMetricSpecification specification =
PredefinedMetricSpecification.builder()

.predefinedMetricType(MetricType.DYNAMO_DB_WRITE_CAPACITY_UTILIZATION)
                .build();

            TargetTrackingScalingPolicyConfiguration policyConfiguration =
TargetTrackingScalingPolicyConfiguration.builder()
                .predefinedMetricSpecification(specification)
```

```
                .targetValue(50.0)
                .scaleInCooldown(60)
                .scaleOutCooldown(60)
                .build();

            PutScalingPolicyRequest putScalingPolicyRequest =
 PutScalingPolicyRequest.builder()
                .targetTrackingScalingPolicyConfiguration(policyConfiguration)
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .policyName(policyName)
                .policyType(PolicyType.TARGET_TRACKING_SCALING)
                .build();

            try {
                appAutoScalingClient.putScalingPolicy(putScalingPolicyRequest);
                System.out.println("You have successfully created a scaling policy
 for an Application Auto Scaling scalable target");
            } catch (ApplicationAutoScalingException e) {
                System.err.println("Error: " + e.awsErrorDetails().errorMessage());
            }
        }
    }
}
```

Java v1

O programa exige que você forneça um nome do recurso da Amazon (ARN) para uma função vinculada ao serviço Application Auto Scaling válida. (Por exemplo: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`.) No seguinte programa, substitua `SERVICE_ROLE_ARN_GOES_HERE` pelo ARN real.

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClientBuilder;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
```

```
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.MetricType;
import com.amazonaws.services.applicationautoscaling.model.PolicyType;
import
 com.amazonaws.services.applicationautoscaling.model.PredefinedMetricSpecification;
import com.amazonaws.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
import
 com.amazonaws.services.applicationautoscaling.model.TargetTrackingScalingPolicyConfiguratio

public class EnableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = (AWSApplicationAutoScalingClient)
 AWSApplicationAutoScalingClientBuilder
   .standard().build();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Define the scalable target
  RegisterScalableTargetRequest rstRequest = new RegisterScalableTargetRequest()
    .withServiceNamespace(ns)
    .withResourceId(resourceID)
    .withScalableDimension(tableWCUs)
    .withMinCapacity(5)
    .withMaxCapacity(10)
    .withRoleARN("SERVICE_ROLE_ARN_GOES_HERE");

  try {
   aaClient.registerScalableTarget(rstRequest);
  } catch (Exception e) {
   System.err.println("Unable to register scalable target: ");
   System.err.println(e.getMessage());
```

```
 }

 // Verify that the target was created
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);
 try {
  DescribeScalableTargetsResult dsaResult =
aaClient.describeScalableTargets(dscRequest);
  System.out.println("DescribeScalableTargets result: ");
  System.out.println(dsaResult);
  System.out.println();
 } catch (Exception e) {
  System.err.println("Unable to describe scalable target: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Configure a scaling policy
 TargetTrackingScalingPolicyConfiguration targetTrackingScalingPolicyConfiguration
= new TargetTrackingScalingPolicyConfiguration()
   .withPredefinedMetricSpecification(
     new PredefinedMetricSpecification()
       .withPredefinedMetricType(MetricType.DynamoDBWriteCapacityUtilization))
   .withTargetValue(50.0)
   .withScaleInCooldown(60)
   .withScaleOutCooldown(60);

 // Create the scaling policy, based on your configuration
 PutScalingPolicyRequest pspRequest = new PutScalingPolicyRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID)
   .withPolicyName("MyScalingPolicy")
   .withPolicyType(PolicyType.TargetTrackingScaling)

.withTargetTrackingScalingPolicyConfiguration(targetTrackingScalingPolicyConfiguration);

 try {
  aaClient.putScalingPolicy(pspRequest);
 } catch (Exception e) {
  System.err.println("Unable to put scaling policy: ");
```

```
    System.err.println(e.getMessage());
   }

   // Verify that the scaling policy was created
   DescribeScalingPoliciesRequest dspRequest = new DescribeScalingPoliciesRequest()
     .withServiceNamespace(ns)
     .withScalableDimension(tableWCUs)
     .withResourceId(resourceID);

   try {
    DescribeScalingPoliciesResult dspResult =
  aaClient.describeScalingPolicies(dspRequest);
    System.out.println("DescribeScalingPolicies result: ");
    System.out.println(dspResult);
   } catch (Exception e) {
    e.printStackTrace();
    System.err.println("Unable to describe scaling policy: ");
    System.err.println(e.getMessage());
   }

  }

}
```

Desabilitar o Application Auto Scaling para uma tabela

O programa a seguir inverte o processo anterior. Ele remove a política de Auto Scaling e cancela o registro do destino dimensionável.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeregisterScalableTargetReques
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class DisableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).\s
               policyName - The name of the policy (for example, $Music5-scaling-
policy).

            """;
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();
```

```
        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        String tableId = args[0];
        String policyName = args[1];

        deletePolicy(appAutoScalingClient, policyName, tableWCUs, ns, tableId);
        verifyScalingPolicies(appAutoScalingClient, tableId, ns, tableWCUs);
        deregisterScalableTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
    }

    public static void deletePolicy(ApplicationAutoScalingClient
appAutoScalingClient, String policyName, ScalableDimension tableWCUs,
ServiceNamespace ns, String tableId) {
        try {
            DeleteScalingPolicyRequest delSPRequest =
DeleteScalingPolicyRequest.builder()
                .policyName(policyName)
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deleteScalingPolicy(delSPRequest);
            System.out.println(policyName +" was deleted successfully.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the scaling policy was deleted
    public static void verifyScalingPolicies(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalingPoliciesRequest dscRequest =
DescribeScalingPoliciesRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceId(tableId)
            .build();
```

```
        DescribeScalingPoliciesResponse response =
 appAutoScalingClient.describeScalingPolicies(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    public static void deregisterScalableTarget(ApplicationAutoScalingClient
 appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
 tableWCUs) {
        try {
            DeregisterScalableTargetRequest targetRequest =
 DeregisterScalableTargetRequest.builder()
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deregisterScalableTarget(targetRequest);
            System.out.println("The scalable target was deregistered.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    public static void verifyTarget(ApplicationAutoScalingClient
 appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
 tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
 DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
 appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }
}
```

Java v1

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DeregisterScalableTargetRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;

public class DisableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = new
 AWSApplicationAutoScalingClient();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Delete the scaling policy
  DeleteScalingPolicyRequest delSPRequest = new DeleteScalingPolicyRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID)
    .withPolicyName("MyScalingPolicy");

  try {
   aaClient.deleteScalingPolicy(delSPRequest);
  } catch (Exception e) {
   System.err.println("Unable to delete scaling policy: ");
   System.err.println(e.getMessage());
```

```
 }

 // Verify that the scaling policy was deleted
 DescribeScalingPoliciesRequest descSPRequest = new
DescribeScalingPoliciesRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  DescribeScalingPoliciesResult dspResult =
aaClient.describeScalingPolicies(descSPRequest);
  System.out.println("DescribeScalingPolicies result: ");
  System.out.println(dspResult);
 } catch (Exception e) {
  e.printStackTrace();
  System.err.println("Unable to describe scaling policy: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Remove the scalable target
 DeregisterScalableTargetRequest delSTRequest = new
DeregisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  aaClient.deregisterScalableTarget(delSTRequest);
 } catch (Exception e) {
  System.err.println("Unable to deregister scalable target: ");
  System.err.println(e.getMessage());
 }

 // Verify that the scalable target was removed
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);

 try {
```

```
    DescribeScalableTargetsResult dsaResult =
  aaClient.describeScalableTargets(dscRequest);
    System.out.println("DescribeScalableTargets result: ");
    System.out.println(dsaResult);
    System.out.println();
   } catch (Exception e) {
    System.err.println("Unable to describe scalable target: ");
    System.err.println(e.getMessage());
   }

  }

}
```

Capacidade reservada do DynamoDB

Para tabelas de capacidade provisionada que usam a classe de tabela padrão, o DynamoDB oferece a opção de comprar capacidade reservada para leitura e gravação. A compra de capacidade reservada é um acordo para pagar por uma quantidade mínima de capacidade de throughput provisionado, durante a vigência do contrato, em troca de preços com desconto.

> **Note**
>
> Não é possível comprar capacidade reservada para unidades de capacidade de gravação replicada (rWCUs). A capacidade reservada é aplicada somente à região onde foi comprada. A capacidade reservada também não está disponível para tabelas que usam a classe de tabela Standard-IA do DynamoDB ou o modo de capacidade sob demanda.

A capacidade reservada é adquirida em alocações de 100 WCUs ou 100 RCUs. A menor oferta de capacidade reservada é de cem unidades de capacidade (leituras ou gravações). A capacidade reservada do DynamoDB é oferecida como um compromisso de um ano ou, em regiões selecionadas, como um compromisso de três anos. É possível economizar até 54% nas taxas padrão por um período de um ano e 77% nas taxas padrão por um período de três anos. Para ter mais informações sobre como e quando você deve comprar, consulte Amazon DynamoDB Reserved Capacity.

Ao comprar a capacidade reservada do DynamoDB, você realiza um único pagamento antecipado parcial e recebe uma taxa horária reduzida pelo uso provisionado. Você paga por todo o uso provisionado comprometido, independentemente do uso real; portanto, a redução de custos está estreitamente ligada ao uso. Qualquer capacidade que você provisionar além da capacidade reservada comprada será cobrada de acordo com as taxas de capacidade provisionada padrão. Ao reservar suas unidades de capacidade de leitura e gravação com antecedência, há economia de custo significativa nos custos de capacidade provisionada.

Não é permitido vender, cancelar nem transferir a capacidade reservada para outra região ou conta.

Noções básicas sobre o throughput a quente do DynamoDB

Throughput a quente refere-se ao número de operações de leitura e de gravação que sua tabela do DynamoDB pode comportar instantaneamente. Esses valores estão disponíveis por padrão para todas as tabelas e índices secundários globais (GSI) e representam o quanto eles foram ajustados com base no histórico de uso. Se você estiver usando o modo sob demanda ou se atualizar o throughput provisionado para esses valores, sua aplicação poderá emitir solicitações até esses valores instantaneamente.

O DynamoDB ajustará automaticamente os valores de throughput a quente à medida que o uso aumentar. No entanto, você também pode aumentar esses valores de maneira proativa quando necessário, o que é especialmente útil para eventos de pico futuros, como lançamentos de produtos ou vendas. Em relação a eventos de pico planejados, nos quais as taxas de solicitação para sua tabela do DynamoDB podem aumentar em 10x, 100x ou mais, agora você pode avaliar se o throughput a quente atual é suficiente para lidar com o tráfego esperado. Caso não seja, você poderá aumentar o valor de throughput a quente sem alterar suas configurações de throughput nem o modo de faturamento. Esse processo é conhecido como preaquecimento de uma tabela. Ele permite que você defina uma linha de base que suas tabelas possam comportar instantaneamente. Isso garante que as aplicações possam lidar com taxas de solicitação mais altas a partir do momento em que elas ocorrem.

É possível aumentar o valor do throughput a quente para operações de leitura, de gravação ou ambas. É possível aumentar esse valor para tabelas de região única novas e existentes, tabelas globais e GSIs. Em relação a tabelas globais, esse recurso está disponível para a versão 2019.11.21 (atual), e as configurações de throughput a quente que você definir serão aplicadas automaticamente a todas as tabelas de réplica na tabela global. Não há limite para o número de tabelas do DynamoDB que você pode preaquecer a qualquer momento. O tempo para concluir o preaquecimento depende dos valores definidos e do tamanho da tabela ou do índice. É possível enviar solicitações

simultâneas de preaquecimento, e essas solicitações não interferirão em nenhuma operação da tabela. É possível preaquecer a tabela até o limite de cota de tabela ou do índice da sua conta nessa região. Use o console do Service Quotas para conferir seus limites atuais e aumentá-los, se necessário.

Os valores de throughput a quente estão disponíveis por padrão para todas as tabelas e índices secundários, sem nenhum custo. No entanto, se você aumentar proativamente esses valores padrão de throughput a quente para preaquecer as tabelas, haverá cobranças por essas solicitações. Para obter mais informações, consulte a Definição de preço do Amazon DynamoDB.

Para ter mais informações sobre throughput a quente, consulte os seguintes tópicos:

Tópicos

- Conferir o throughput a quente atual da tabela do DynamoDB
- Aumentar o throughput a quente de sua tabela existente do DynamoDB
- Criar uma tabela do DynamoDB com um throughput a quente mais alto
- Noções básicas sobre throughput a quente do DynamoDB em diferentes cenários

Conferir o throughput a quente atual da tabela do DynamoDB

Use as instruções a seguir da AWS CLI e do console da AWS para conferir o valor atual do throughput a quente da tabela ou do índice.

AWS Management Console

Para conferir o throughput a quente da sua tabela do DynamoDB usando o console do DynamoDB:

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).
3. Na página Tabelas, escolha a tabela desejada.
4. Selecione Configurações adicionais para visualizar o valor atual do throughput a quente. Esse valor é mostrado como unidades de leitura e de gravação por segundo.

AWS CLI

O exemplo da AWS CLI a seguir mostra como conferir o throughput a quente da tabela do DynamoDB.

1. Execute a operação `describe-table` na tabela do DynamoDB.

```
aws dynamodb describe-table --table-name GameScores
```

2. Você receberá uma resposta semelhante à seguinte. Suas configurações de `WarmThroughput` serão exibidas como `ReadUnitsPerSecond` e `WriteUnitsPerSecond`. O `Status` será `UPDATING` quando o valor de throughput a quente estiver sendo atualizado, e `ACTIVE` quando o novo valor de throughput a quente for definido.

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1726128388.729,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST",
            "LastUpdateToPayPerRequestDateTime": 1726128388.729
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 0,
                    "WriteCapacityUnits": 0
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 12000,
                    "WriteUnitsPerSecond": 4000,
                    "Status": "ACTIVE"
                }
            }
        ],
```

```
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12000,
            "WriteUnitsPerSecond": 4000,
            "Status": "ACTIVE"
        }
    }
}
```

Aumentar o throughput a quente de sua tabela existente do DynamoDB

Depois de conferir o valor atual do throughput a quente da tabela do DynamoDB, você poderá atualizá-lo por meio das seguintes etapas:

AWS Management Console

Para conferir o throughput a quente da sua tabela do DynamoDB usando o console do DynamoDB:

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação à esquerda, selecione Tables (Tabelas).
3. Na página Tabelas, escolha a tabela desejada.
4. No campo Throughput a quente, selecione Editar.
5. Na página Editar throughput a quente, escolha Aumentar throughput a quente.
6. Ajuste as unidades de leitura por segundo e unidades de gravação por segundo. Essas duas configurações definem o throughput que sua tabela pode processar instantaneamente.
7. Selecione Salvar.
8. Suas unidades de leitura por segundo e unidades de gravação por segundo serão atualizadas no campo Throughput a quente quando o processamento da solicitação for concluído.

> **Note**
>
> Atualizar o valor de throughput a quente é uma tarefa assíncrona. O `Status` mudará de `UPDATING` para `ACTIVE` quando a atualização for concluída.

AWS CLI

O exemplo de AWS CLI a seguir mostra como atualizar o valor de throughput a quente da tabela do DynamoDB.

1. Execute a operação `update-table` na tabela do DynamoDB.

```
aws dynamodb update-table \
    --table-name GameScores \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --global-secondary-index-updates \
        "[
            {
                \"Update\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"WarmThroughput\": {
                        \"ReadUnitsPerSecond\": 88,
                        \"WriteUnitsPerSecond\": 77
                    }
                }
            }
        ]" \
    --region us-east-1
```

2. Você receberá uma resposta semelhante à seguinte. Suas configurações de `WarmThroughput` serão exibidas como `ReadUnitsPerSecond` e `WriteUnitsPerSecond`. O `Status` será `UPDATING` quando o valor de throughput a quente estiver sendo atualizado, e `ACTIVE` quando o novo valor de throughput a quente for definido.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
```

```
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1730242189.965,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
```

```
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 50,
                    "WriteUnitsPerSecond": 25,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12300,
            "WriteUnitsPerSecond": 4500,
            "Status": "UPDATING"
        }
    }
}
```

AWS SDK

Os exemplos de SDK a seguir mostram como atualizar o valor de throughput a quente da tabela do DynamoDB.

Java

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

...
public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
```

```
                                                    final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}

public static void updateDynamoDBTable(DynamoDbClient ddb,
                                       String tableName,
                                       Long tableReadUnitsPerSecond,
                                       Long tableWriteUnitsPerSecond,
                                       String globalSecondaryIndexName,
                                       Long globalSecondaryIndexReadUnitsPerSecond,
                                       Long globalSecondaryIndexWriteUnitsPerSecond)
 {

    final WarmThroughput tableWarmThroughput =
 buildWarmThroughput(tableReadUnitsPerSecond, tableWriteUnitsPerSecond);
    final WarmThroughput gsiWarmThroughput =
 buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
 globalSecondaryIndexWriteUnitsPerSecond);

    final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
 GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                    .indexName(globalSecondaryIndexName)
                    .warmThroughput(gsiWarmThroughput)
                    .build()
            ).build();

    final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

    try {
        ddb.updateTable(request);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }

    System.out.println("Done!");
```

```
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def update_dynamodb_table_warm_throughput(table_name, table_read_units,
 table_write_units, gsi_name, gsi_read_units, gsi_write_units, region_name="us-
east-1"):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = resource('dynamodb', region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {
```

```
                "Update": {
                    "IndexName": gsi_name,
                    "WarmThroughput": gsi_warm_throughput
                }
            }
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput
        }

        # Update the table
        ddb.update_table(**update_table_request)
        print("Table updated successfully!")
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
```

```
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response: ${response}`);
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}
```

Criar uma tabela do DynamoDB com um throughput a quente mais alto

É possível ajustar os valores de throughput a quente ao criar sua tabela do DynamoDB seguindo as etapas abaixo. Essas etapas também se aplicam ao criar uma tabela global ou um índice secundário.

AWS Management Console

Para criar uma tabela do DynamoDB e ajustar os valores de throughput a quente por meio do console:

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. Selecione Criar tabela.
3. Escolha o Nome da tabela, a Chave de partição e a Chave de classificação (opcional).
4. Em Configurações da tabela, selecione Personalizar configurações.
5. No campo Throughput a quente, escolha Aumentar throughput a quente.

6. Ajuste as unidades de leitura por segundo e unidades de gravação por segundo. Essas duas configurações definem o throughput máximo que sua tabela pode processar instantaneamente.
7. Continue adicionando os detalhes restantes da tabela e selecione Criar tabela.

AWS CLI

O exemplo da AWS CLI a seguir mostra como criar uma tabela do DynamoDB com valores personalizados de throughput a quente.

1. Execute a operação `create-table` para criar a tabela do DynamoDB a seguir.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 50,
                    \"WriteCapacityUnits\": 25
                },\"WarmThroughput\": {
                    \"ReadUnitsPerSecond\": 1987,
                    \"WriteUnitsPerSecond\": 543
                }
            }
        ]" \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
```

```
    --region us-east-1
```

2. Você receberá uma resposta semelhante à seguinte. Suas configurações de `WarmThroughput` serão exibidas como `ReadUnitsPerSecond` e `WriteUnitsPerSecond`. O `Status` será `UPDATING` quando o valor de throughput a quente estiver sendo atualizado, e `ACTIVE` quando o novo valor de throughput a quente for definido.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": 1730241788.779,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
```

```
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 1987,
                    "WriteUnitsPerSecond": 543,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12345,
            "WriteUnitsPerSecond": 4567,
            "Status": "UPDATING"
        }
```

```
    }
}
```

AWS SDK

Os exemplos de SDK a seguir mostram como criar uma tabela do DynamoDB com valores personalizados de throughput a quente.

Java

```
import software.amazon.awscdk.services.dynamodb.ProjectionType;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;
...

public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}
private static AttributeDefinition buildAttributeDefinition(final String
 attributeName,
                                                             final
 ScalarAttributeType scalarAttributeType) {
    return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
}
private static KeySchemaElement buildKeySchemaElement(final String attributeName,
                                                      final KeyType keyType) {
```

```
    return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
}
public static void createDynamoDBTable(DynamoDbClient ddb,
                                        String tableName,
                                        String partitionKey,
                                        String sortKey,
                                        String miscellaneousKeyAttribute,
                                        String nonKeyAttribute,
                                        Long tableReadCapacityUnits,
                                        Long tableWriteCapacityUnits,
                                        Long tableWarmReadUnitsPerSecond,
                                        Long tableWarmWriteUnitsPerSecond,
                                        String globalSecondaryIndexName,
                                        Long globalSecondaryIndexReadCapacityUnits,
                                        Long globalSecondaryIndexWriteCapacityUnits,
                                        Long
 globalSecondaryIndexWarmReadUnitsPerSecond,
                                        Long
 globalSecondaryIndexWarmWriteUnitsPerSecond) {

    // Define the table attributes
    final AttributeDefinition partitionKeyAttribute =
 buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
    final AttributeDefinition sortKeyAttribute = buildAttributeDefinition(sortKey,
 ScalarAttributeType.S);
    final AttributeDefinition miscellaneousKeyAttributeDefinition =
 buildAttributeDefinition(miscellaneousKeyAttribute, ScalarAttributeType.N);
    final AttributeDefinition[] attributeDefinitions = {partitionKeyAttribute,
 sortKeyAttribute, miscellaneousKeyAttributeDefinition};

    // Define the table key schema
    final KeySchemaElement partitionKeyElement = buildKeySchemaElement(partitionKey,
 KeyType.HASH);
    final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
 KeyType.RANGE);
    final KeySchemaElement[] keySchema = {partitionKeyElement, sortKeyElement};

    // Define the provisioned throughput for the table
    final ProvisionedThroughput provisionedThroughput =
 ProvisionedThroughput.builder()
            .readCapacityUnits(tableReadCapacityUnits)
```

```
            .writeCapacityUnits(tableWriteCapacityUnits)
            .build();

   // Define the Global Secondary Index (GSI)
   final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
   final KeySchemaElement globalSecondaryIndexSortKeyElement =
buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
   final KeySchemaElement[] gsiKeySchema =
{globalSecondaryIndexPartitionKeyElement, globalSecondaryIndexSortKeyElement};

   final Projection gsiProjection = Projection.builder()
           .projectionType(String.valueOf(ProjectionType.INCLUDE))
           .nonKeyAttributes(nonKeyAttribute)
           .build();
   final ProvisionedThroughput gsiProvisionedThroughput =
ProvisionedThroughput.builder()
           .readCapacityUnits(globalSecondaryIndexReadCapacityUnits)
           .writeCapacityUnits(globalSecondaryIndexWriteCapacityUnits)
           .build();
   // Define the warm throughput for the Global Secondary Index (GSI)
   final WarmThroughput gsiWarmThroughput =
buildWarmThroughput(globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);
   final GlobalSecondaryIndex globalSecondaryIndex = GlobalSecondaryIndex.builder()
           .indexName(globalSecondaryIndexName)
           .keySchema(gsiKeySchema)
           .projection(gsiProjection)
           .provisionedThroughput(gsiProvisionedThroughput)
           .warmThroughput(gsiWarmThroughput)
           .build();

   // Define the warm throughput for the table
   final WarmThroughput tableWarmThroughput =
buildWarmThroughput(tableWarmReadUnitsPerSecond, tableWarmWriteUnitsPerSecond);

   final CreateTableRequest request = CreateTableRequest.builder()
           .tableName(tableName)
           .attributeDefinitions(attributeDefinitions)
           .keySchema(keySchema)
           .provisionedThroughput(provisionedThroughput)
           .globalSecondaryIndexes(globalSecondaryIndex)
           .warmThroughput(tableWarmThroughput)
           .build();
```

```
    CreateTableResponse response = ddb.createTable(request);
    System.out.println(response);
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def create_dynamodb_table_warm_throughput(table_name, partition_key,
 sort_key, misc_key_attr, non_key_attr, table_provisioned_read_units,
 table_provisioned_write_units, table_warm_reads, table_warm_writes, gsi_name,
 gsi_provisioned_read_units, gsi_provisioned_write_units, gsi_warm_reads,
 gsi_warm_writes, region_name="us-east-1"):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned read
 capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's warm
 throughput.
    :param table_warm_writes: The write units per second setting for the table's
 warm throughput.
    :param gsi_name: The name of the Global Secondary Index (GSI) to be created on
 the table.
    :param gsi_provisioned_read_units: The configured Global Secondary Index (GSI)
 provisioned read capacity units.
    :param gsi_provisioned_write_units: The configured Global Secondary Index (GSI)
 provisioned write capacity units.
    :param gsi_warm_reads: The read units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param gsi_warm_writes: The write units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
```

```
    try:
        ddb = resource('dynamodb', region_name)

        # Define the table attributes
        attribute_definitions = [
            { "AttributeName": partition_key, "AttributeType": "S" },
            { "AttributeName": sort_key, "AttributeType": "S" },
            { "AttributeName": misc_key_attr, "AttributeType": "N" }
        ]

        # Define the table key schema
        key_schema = [
            { "AttributeName": partition_key, "KeyType": "HASH" },
            { "AttributeName": sort_key, "KeyType": "RANGE" }
        ]

        # Define the provisioned throughput for the table
        provisioned_throughput = {
            "ReadCapacityUnits": table_provisioned_read_units,
            "WriteCapacityUnits": table_provisioned_write_units
        }

        # Define the global secondary index
        gsi_key_schema = [
            { "AttributeName": sort_key, "KeyType": "HASH" },
            { "AttributeName": misc_key_attr, "KeyType": "RANGE" }
        ]
        gsi_projection = {
            "ProjectionType": "INCLUDE",
            "NonKeyAttributes": [non_key_attr]
        }
        gsi_provisioned_throughput = {
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
```

```
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput
            }
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput
        )

        print(response)
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
```

```
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
```

```
          },
        ],
      });
      const response = await ddbClient.send(command);
      console.log(response);
    } catch (error) {
      console.error(`Error creating table: ${error}`);
      throw error;
    }
}
```

Noções básicas sobre throughput a quente do DynamoDB em diferentes cenários

Veja a seguir alguns cenários diferentes que você pode encontrar ao trabalhar com o throughput a quente do DynamoDB.

Tópicos

- Throughput a quente e padrões de acesso desiguais
- Throughput a quente para uma tabela provisionada
- Throughput a quente para uma tabela sob demanda
- Throughput a quente para uma tabela sob demanda com throughput máximo configurado

Throughput a quente e padrões de acesso desiguais

Uma tabela pode ter um throughput a quente de 30 mil unidades de leitura por segundo e 10 mil unidades de gravação por segundo, mas você ainda pode experimentar controle de utilização em leituras ou gravações antes de atingir esses valores. Isso provavelmente se deve a uma partição sobrecarregada. Embora o DynamoDB possa continuar a escalar para comportar um throughput praticamente ilimitado, cada partição individual é limitada a mil unidades de gravação por segundo e 3 mil unidades de leitura por segundo. Se a aplicação direcionar muito tráfego para uma pequena parte das partições da tabela, poderá ocorrer controle de utilização mesmo antes de você atingir os valores de throughput a quente da tabela. Recomendamos seguir as práticas recomendadas do DynamoDB para garantir uma escalabilidade perfeita e evitar partições sobrecarregadas.

Throughput a quente para uma tabela provisionada

Pense em uma tabela provisionada que tenha um throughput a quente de 30 mil unidades de leitura por segundo e 10 mil unidades de gravação por segundo, mas que atualmente tenha um throughput provisionado de 4 mil RCU e 8 mil WCU. É possível escalar instantaneamente o throughput provisionado da tabela até 30 mil RCU ou 10 mil WCU atualizando suas configurações de throughput provisionado. À medida que você aumenta o throughput provisionado além desses valores, o throughput a quente se ajustará automaticamente aos novos valores mais altos, porque você estabeleceu um novo throughput de pico. Por exemplo, se você definir o throughput provisionado para 50 mil RCU, o throughput a quente aumentará para 50 mil unidades de leitura por segundo.

```
"ProvisionedThroughput":
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Throughput a quente para uma tabela sob demanda

Uma nova tabela sob demanda começa com um throughput a quente de 12 mil unidades de leitura por segundo e 4 mil unidades de gravação por segundo. Sua tabela pode acomodar instantaneamente o tráfego sustentado até esses níveis. Quando suas solicitações excederem 12 mil unidades de leitura por segundo ou 4 mil unidades de gravação por segundo, o throughput a quente se ajustará automaticamente a valores mais altos.

```
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

Throughput a quente para uma tabela sob demanda com throughput máximo configurado

Pense em uma tabela sob demanda com um throughput a quente de 30 mil unidades de leitura por segundo, mas com um throughput máximo configurado em 5 mil unidades de solicitação de leitura (RRU). Nesse cenário, o throughput da tabela será limitado ao máximo de 5 mil RRU que você definiu. Qualquer solicitação de throughput que exceda esse máximo terá controle de utilização. No entanto, é possível modificar o throughput máximo específico da tabela a qualquer momento, com base nas necessidades da aplicação.

```
"OnDemandThroughput":
    {
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Capacidade de expansão e capacidade adaptável do DynamoDB

Para minimizar o controle de utilização decorrente de exceções de throughput, o DynamoDB usa a capacidade de expansão para lidar com picos de uso. O DynamoDB usa a capacidade adaptável para ajudar a acomodar padrões de acesso desiguais.

Capacidade de expansão

O DynamoDB fornece alguma flexibilidade para o provisionamento de throughput com capacidade de expansão. Quando você não está usando totalmente o throughput disponível, o DynamoDB reserva uma parte dessa capacidade não utilizada para expansões posteriores do throughput com o objetivo de lidar com os picos de uso. Com a capacidade de expansão, as solicitações de leitura ou gravação inesperadas podem ter êxito onde, de outra forma, seriam limitadas.

O DynamoDB retém até cinco minutos (trezentos segundos) de capacidade de leitura e de gravação não utilizada. Durante uma expansão ocasional de atividades de leitura e de gravação, essas unidades de capacidade extra podem ser consumidas muito rapidamente, ainda mais depressa do que a capacidade de throughput provisionado por segundo que você definiu para a tabela.

O DynamoDB também pode consumir capacidade de intermitência para manutenção em segundo plano e para outras tarefas sem aviso prévio.

Note que esses detalhes da capacidade de intermitência podem mudar no futuro.

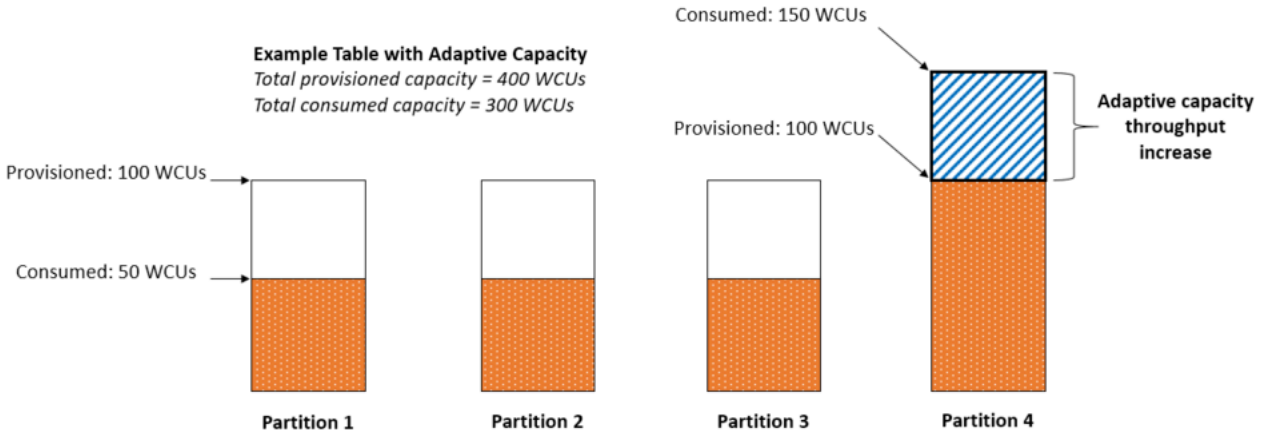
Capacidade adaptável

O DynamoDB distribui automaticamente os dados entre as partições, armazenando-os em vários servidores na Nuvem AWS. Nem sempre é possível distribuir a atividade de leitura e de gravação uniformemente o tempo todo. Quando o acesso aos dados é desequilibrado, uma partição "dinâmica" pode receber um volume mais alto de tráfego de leitura e gravação em comparação com outras partições. Como as operações de leitura e de gravação em uma partição são gerenciadas de forma independente, haverá controle de utilização se uma única partição receber mais de três mil operações de leitura ou mais de mil operações de gravação. A capacidade adaptável funciona por meio do aumento automático da capacidade de throughput para as partições que recebem mais tráfego.

Para acomodar melhor padrões desiguais de acesso, a capacidade adaptável do DynamoDB permite que a aplicação continue a ler e gravar em partições dinâmicas sem ser limitado, desde que o tráfego não exceda a capacidade total provisionada da tabela ou a capacidade máxima da partição. A capacidade adaptável funciona por meio do aumento automático e instantâneo da capacidade de throughput para as partições que recebem mais tráfego.

O diagrama a seguir mostra como a capacidade adaptável funciona. A tabela de exemplo é provisionada com 400 WCUs compartilhadas uniformemente em quatro partições, permitindo que cada partição comporte até 100 WCUs por segundo. As partições 1, 2 e 3 recebem tráfego de gravação de 50 WCU/s. Partição 4 recebe 150 WCU/s. Essa partição dinâmica poderá aceitar tráfego de gravação enquanto tiver capacidade não utilizada, mas em algum momento limitará o tráfego que exceder 100 WCU/s.

A capacidade adaptável do DynamoDB responde por meio do aumento da capacidade da partição 4 para que ela comporte uma workload maior de 150 WCUs/s sem controle de utilização.



A capacidade adaptável é habilitada automaticamente para todas as tabelas do DynamoDB sem custo adicional. Não é necessário habilitá-la ou desabilitá-la explicitamente.

Isolar itens acessados com frequência

Se o aplicativo direcionar tráfego desproporcionalmente alto para um ou mais itens, a capacidade adaptável reequilibrará as partições de modo que os itens acessados com frequência não residam na mesma partição. Esse isolamento dos itens acessados com frequência reduz a probabilidade da limitação de solicitação devido à workload exceder a cota de throughput em uma única partição. Você também pode dividir um conjunto de itens em segmentos por chave de classificação, desde que esse conjunto não seja um tráfego rastreado por uma diminuição ou um aumento monotônico da chave de classificação.

Se a aplicação direciona alto tráfego constantemente a um único item, a capacidade adaptável poderá reequilibrar os dados de modo que uma partição contenha somente esse único item acessado com frequência. Nesse caso, o DynamoDB pode entregar um throughput de no máximo de 3 mil RCUs da partição ou mil WCUs para a chave primária desse único item. A capacidade adaptável não dividirá as coleções de itens em várias partições da tabela quando houver um índice secundário local na tabela.

Considerações ao alternar os modos de capacidade no DynamoDB

Ao criar uma tabela do DynamoDB, é necessário selecionar o modo de capacidade sob demanda ou provisionada.

É possível alternar as tabelas do modo de capacidade provisionada para o modo sob demanda até quatro vezes em um período contínuo de 24 horas. É possível alternar as tabelas do modo sob demanda para o modo de capacidade provisionada a qualquer momento.

Tópicos

- Alternar do modo de capacidade provisionada para o modo de capacidade sob demanda
- Alternar do modo de capacidade sob demanda para o modo de capacidade provisionada

Alternar do modo de capacidade provisionada para o modo de capacidade sob demanda

No modo provisionado, você define a capacidade de leitura e de gravação com base nas necessidades esperadas da aplicação. Quando você atualiza uma tabela de um modo sob demanda provisionado, não é necessário especificar o throughput de leitura e gravação que você espera que seu aplicativo execute. O DynamoDB sob demanda oferece o modelo de preço de pagamento por solicitação (de leitura e de gravação) para que você pague apenas pelo que usar, o que permite contrabalançar com facilidade custos e performance. Você também pode configurar o throughput máximo de leitura ou de gravação (ou de ambas) para tabelas individuais sob demanda e índices secundários globais associados a fim de ajudar a limitar os custos e o uso. Para ter mais informações sobre como definir o throughput máximo para uma tabela ou um índice específico, consulte Throughput máximo do DynamoDB para tabelas sob demanda.

Quando se altera o modo de capacidade provisionado para o modo de capacidade sob demanda, o DynamoDB faz várias alterações na estrutura da tabela e das partições. Esse processo pode levar alguns minutos. Durante o período de troca, sua tabela entrega throughput que é consistente com as unidades de valor de capacidade de gravação provisionada anteriormente e unidade de capacidade de leitura.

Throughput inicial para modo de capacidade sob demanda

Se recentemente você tiver alterado uma tabela existente para o modo de capacidade sob demanda pela primeira vez, ela terá as configurações de pico anterior a seguir, mesmo se não tiver apresentado tráfego anteriormente usando o modo de capacidade sob demanda.

Veja a seguir exemplos de possíveis cenários:

- Qualquer tabela provisionada configurada abaixo de 4.000 WCUs e 12.000 RCUs, que nunca tenha sido provisionada anteriormente para mais. Quando essa tabela for alterada para sob

demanda pela primeira vez, o DynamoDB garantirá que ela aumente a escala horizontalmente para comportar instantaneamente pelo menos 4.000 WCUs/segundo e 12.000 RCUs/segundo.

- Uma tabela provisionada configurada como 8.000 WCUs e 24.000 RCUs. Ao alterar essa tabela para sob demanda, ela continuará comportando pelo menos 8.000 WCUs/segundo e 24.000 RCUs/segundo a qualquer momento.
- Uma tabela provisionada configurada com 8 mil WCU e 24 mil RCU, que consumiu 6 mil unidades de gravação/segundo e 18 mil unidades de leitura/segundo por um período prolongado. Ao alterar essa tabela para sob demanda, ela continuará comportando pelo menos 8.000 WCUs/segundo e 24.000 RCUs/segundo. O tráfego anterior pode ainda permitir que a tabela sustente níveis muito mais altos de tráfego sem controle de utilização.
- Uma tabela anteriormente provisionada com 10 mil WCU e 10 mil RCU, mas atualmente provisionada com 10 RCU e 10 WCU. Ao alterar essa tabela para sob demanda, ela poderá comportar pelo menos 10.000 WCUs/segundo e 10.000 RCUs/segundo.

Configurações de ajuste de escala automático

Quando você atualiza uma tabela do modo provisionado para sob demanda:

- Se estiver usando o console, todas as suas configurações de Auto Scaling (se houver alguma) serão excluídas.
- Se estiver usando a AWS CLI ou o AWS SDK, todas as suas configurações de Auto Scaling serão preservadas. Essas configurações podem ser aplicadas quando você atualiza sua tabela para o modo de cobrança provisionado novamente.

Modo de capacidade de edição em massa no console do DynamoDB.

É possível editar em massa várias tabelas para alternar do modo de capacidade provisionada para o modo de capacidade sob demanda usando o console do DynamoDB. Para editar em massa o modo de capacidade:

1. No console do DynamoDB, acesse a página Tabelas.
2. Marque as caixas de seleção das tabelas que você deseja atualizar para o modo de capacidade sob demanda.
3. Escolha Ações e selecione Atualização para o modo de capacidade sob demanda.

Essa operação em massa permite alternar várias tabelas com eficiência para o modo de capacidade sob demanda sem precisar atualizar cada tabela individualmente.

Alternar do modo de capacidade sob demanda para o modo de capacidade provisionada

Ao voltar para o modo de capacidade sob demanda para modo de capacidade provisionado, sua tabela entrega um throughput consistente com o pico anterior alcançado quando a tabela foi definida como modo de capacidade sob demanda.

Gerenciamento da capacidade

Considere o seguinte ao atualizar uma tabela de modo sob demanda para provisionado:

- Se estiver usando a AWS CLI ou o AWS SDK, escolha as configurações de capacidade provisionadas certas de sua tabela e índices secundários globais usando o Amazon CloudWatch para procurar seu consumo histórico (métricas `ConsumedWriteCapacityUnits` e `ConsumedReadCapacityUnits`) para determinar as novas configurações de throughput.

  > **Note**
  >
  > Se você estiver alternando uma tabela global para o modo provisionado, examine o consumo máximo entre todas as suas réplicas regionais para tabelas de base e índices secundários globais ao determinar as novas configurações de throughput.

- Se você estiver alterando o modo sob demanda de volta para o modo provisionado, não se esqueça de definir as unidades provisionadas iniciais como um valor alto o suficiente para lidar com a capacidade da tabela ou do índice durante a transição.

Gerenciar o Auto Scaling

Quando você atualiza uma tabela do modo sob demanda para provisionado:

- Se estiver usando o console, recomendamos habilitar o ajuste de escala automático com os seguintes padrões:
  - Utilização pretendida: 70%
  - Capacidade provisionada mínima: 5 unidades
  - Capacidade máxima provisionada: o máximo da região

- Se estiver usando a AWS CLI ou o SDK, suas configurações anteriores de Auto Scaling (se houver) serão preservadas.

Programação com o DynamoDB e os AWS SDKs

Esta seção aborda tópicos relacionados a desenvolvedores. Em vez disso, se você deseja executar exemplos de código, consulte Executar os exemplos de código neste Guia do desenvolvedor.

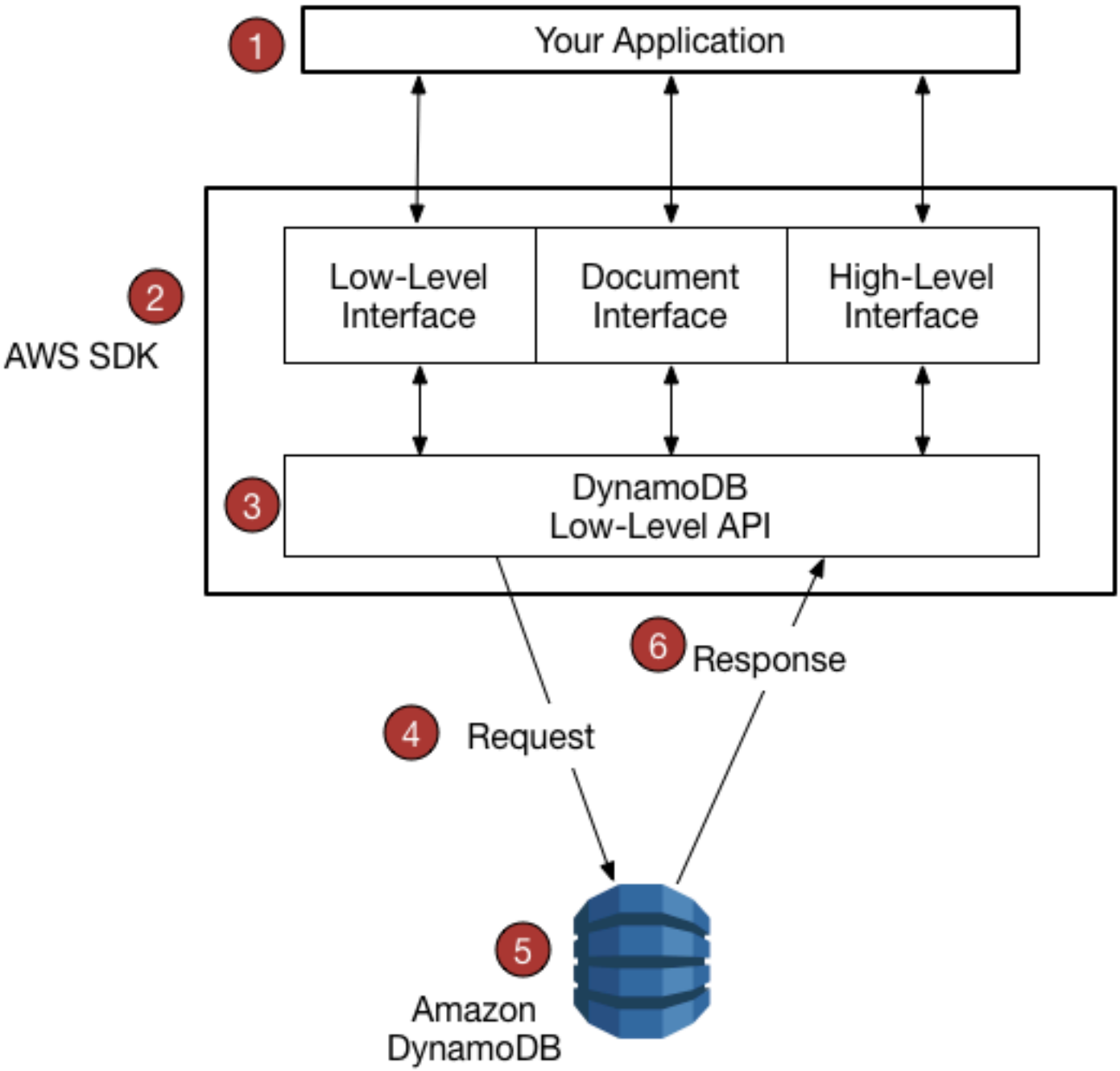
> **Note**
>
> Em dezembro de 2017, a AWS iniciou o processo de migração de todos os endpoints do Amazon DynamoDB para usar certificados seguros emitidos pelo Amazon Trust Services (ATS). Para obter mais informações, consulte Solução de problemas de estabelecimento de conexão SSL/TLS com o DynamoDB.

Tópicos

- Visão geral do suporte a AWS SDKs para o DynamoDB
- Programar o Amazon DynamoDB com Python e Boto3
- Programar o Amazon DynamoDB com JavaScript
- Programar o DynamoDB com o AWS SDK for Java 2.x
- Tratamento de erros com o DynamoDB
- Usar o DynamoDB com um AWS SDK

Visão geral do suporte a AWS SDKs para o DynamoDB

O diagrama a seguir fornece uma visão geral de alto nível da programação de aplicações do Amazon DynamoDB usando os AWS SDKs.



1. Você pode escrever uma aplicação usando um AWS SDK para a sua linguagem de programação.
2. Cada AWS SDK fornece uma ou mais interfaces programáticas para trabalhar com o DynamoDB. As interfaces específicas disponíveis dependem de qual linguagem de programação e AWS SDK você utiliza. Entre as opções estão:
   - Interfaces de baixo nível que funcionam com o DynamoDB
   - Interfaces de documentos que funcionam com o DynamoDB
   - Interfaces de persistência de objetos que funcionam com o DynamoDB

- Interfaces de alto nível

3. O AWS SDKconstrói solicitações HTTP(S) para uso com a API de baixo nível do DynamoDB.
4. O AWS SDK envia a solicitação ao endpoint do DynamoDB.
5. O DynamoDB executa a solicitação. Se a solicitação for bem-sucedida, o DynamoDB retornará um código de resposta HTTP 200 (OK). Se a solicitação não for bem-sucedida, o DynamoDB retornará um código de erro HTTP e uma mensagem de erro.
6. O AWS SDK processa a resposta e a propaga de volta para sua aplicação.

Cada um dos AWS SDKsfornece serviços importantes à sua aplicação, incluindo os seguintes:

- Formatação de solicitações HTTP(S) e serialização de parâmetros de solicitação.
- Geração de uma assinatura criptográfica para cada solicitação.
- Encaminhamento de solicitações a um endpoint do DynamoDB e recebimento de respostas do DynamoDB.
- Extração dos resultados dessas respostas.
- Implementação da lógica de novas tentativas básicas em caso de erros.

Não é necessário escrever código para qualquer uma dessas tarefas.

> **Note**
>
> Para obter mais informações sobre AWS SDKs, incluindo instruções de instalação e documentação, consulte Ferramentas para a Amazon Web Services.

Suporte SDK para endpoints baseados em contas da AWS

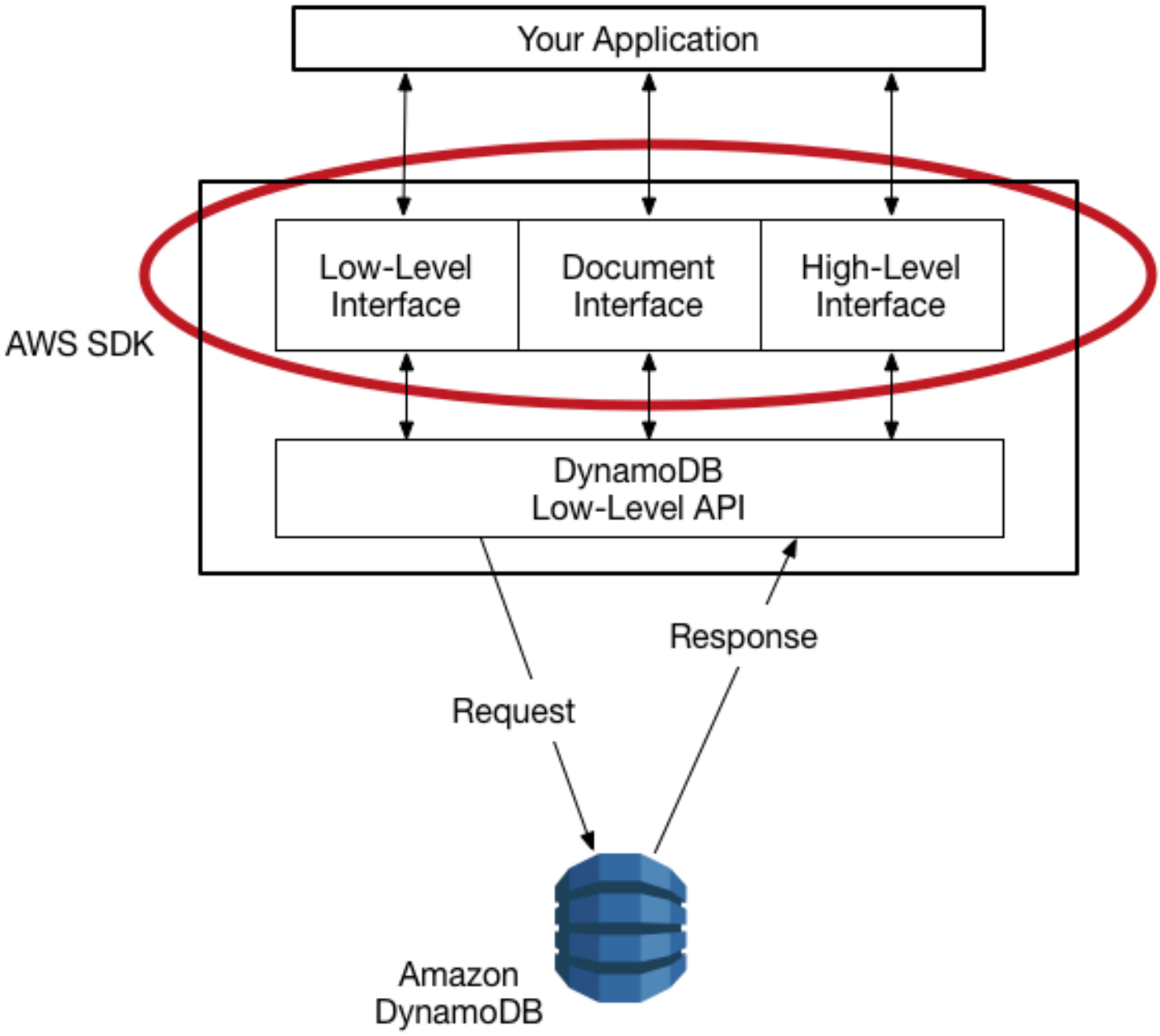
A AWS está implementando o suporte do SDK para endpoints baseados em contas da AWS para o DynamoDB, começando com o AWS SDK for Java V1 em 4 de setembro de 2024. Esses novos endpoints ajudam a AWS garantir alto desempenho e escalabilidade. Os SDKs atualizados usarão automaticamente os novos endpoints, que têm o formato `https://(account-id).ddb.(region).amazonaws.com`

Se você usar uma única instância de um cliente SDK para fazer solicitações para várias contas, sua aplicação terá menos oportunidades de reutilizar conexões. A AWS recomenda modificar as aplicações para se conectar a menos contas por instância do cliente SDK. Uma alternativa é

configurar seu cliente SDK para continuar usando endpoints regionais por meio da configuração `ACCOUNT_ID_ENDPOINT_MODE`, conforme documentado no AWS SDKs and Tools Reference Guide.

Interfaces programáticas que funcionam com o DynamoDB

Cada AWS SDK fornece uma ou mais interfaces programáticas para trabalhar com o Amazon DynamoDB. Essas interfaces variam de wrappers simples de baixo nível do DynamoDB a camadas de persistência orientadas a objetos. As interfaces disponíveis variam dependendo do AWS SDK e da linguagem de programação que você utiliza.



A seção a seguir destaca algumas das interfaces disponíveis, usando o AWS SDK para Java como exemplo. (Nem todas as interfaces estão disponíveis em todos os AWS SDKs.)

Tópicos

- Interfaces de baixo nível que funcionam com o DynamoDB
- Interfaces de documentos que funcionam com o DynamoDB
- Interfaces de persistência de objetos que funcionam com o DynamoDB

Interfaces de baixo nível que funcionam com o DynamoDB

O AWS SDK de cada linguagem específica fornece uma interface de baixo nível para o Amazon DynamoDB, com métodos que se assemelham a solicitações de API de baixo nível do DynamoDB.

Em alguns casos, você precisará identificar os tipos de dados dos atributos usando Descritores de tipo de dados, como `S` para strings ou `N` para números.

> **Note**
>
> Uma interface de baixo nível está disponível no AWS SDK de cada linguagem específica.

O seguinte programa Java usa a interface de baixo nível do AWS SDK para Java.

Exemplo de interface de baixo nível

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName, String
 key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
```

```
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem = ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Interfaces de documentos que funcionam com o DynamoDB

Muitos AWS SDKs fornecem uma interface de documento, permitindo que você realize operações de plano de dados (criar, ler, atualizar, excluir) em tabelas e índices. Com uma interface de documento, você não precisa especificar Descritores de tipo de dados. Os tipos de dados estão implícitos pela semântica dos próprios dados. Estes AWS SDKs também fornecem métodos para converter facilmente documentos JSON de/em tipos de dados nativos do Amazon DynamoDB.

> **Note**
>
> Interfaces de documentos estão disponíveis nos AWS SDKs para Java, .NET, Node.js e JavaScript SDK.

O seguinte programa Java usa a interface de documento do AWS SDK para Java. O programa cria um objeto `Table` que representa a tabela `Music` e depois solicita que esse objeto use `GetItem` para recuperar uma música. Em seguida, o programa imprime o ano em que a canção foi lançada.

A classe `software.amazon.dynamodb.document.DynamoDB` implementa a interface de documento do DynamoDB. Observe como `DynamoDB` atua como um wrapper em torno do cliente de baixo nível (`AmazonDynamoDB`).

Exemplo de interface de documentos

```
package com.amazonaws.codesamples.gsg;

import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.document.DynamoDB;
import software.amazon.dynamodb.document.GetItemOutcome;
import software.amazon.dynamodb.document.Table;

public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");
        System.out.println("The song was released in " + year);

    }
}
```

Interfaces de persistência de objetos que funcionam com o DynamoDB

Alguns AWS SDKs fornecem uma interface de persistência de objetos em que você não realiza operações de plano de dados diretamente. Em vez disso, você cria objetos que representam itens em índices e tabelas do Amazon DynamoDB e interage apenas com esses objetos. Isso permite que você escreva um código centrado em objetos, em vez de um código centrado no banco de dados.

> **Note**
>
> Interfaces de persistência de objetos estão disponíveis nos AWS SDKs for Java e .NET. Para obter mais informações, consulte Interfaces de programação de nível superior para o DynamoDB para o DynamoDB.

Exemplo de interface de persistência de objetos

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;

/*
 * Before running this code example, create an Amazon DynamoDB table named Customer
 with these columns:
 *   - id - the id of the record that is the key. Be sure one of the id values is
 `id101`
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table. These
 values
 *                        need to be in the form of `YYYY-MM-DDTHH:mm:ssZ`, such as
 2022-07-11T00:00:00Z
 *
```

```
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class EnhancedGetItem {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

        getItem(enhancedClient);
        ddb.close();
    }

    public static String getItem(DynamoDbEnhancedClient enhancedClient) {
        Customer result = null;
        try {
            DynamoDbTable<Customer> table = enhancedClient.table("Customer",
 TableSchema.fromBean(Customer.class));
            Key key = Key.builder()
                    .partitionValue("id101").sortValue("tred@noserver.com")
                    .build();

            // Get the item by using the key.
            result = table.getItem(
                    (GetItemEnhancedRequest.Builder requestBuilder) ->
 requestBuilder.key(key));
            System.out.println("******* The description value is " +
 result.getCustName());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return result.getCustName();
```

```
    }
}
```

Interfaces de programação de nível superior para o DynamoDB

Os AWS SDKs fornecem aplicações com interfaces de baixo nível para trabalhar com o Amazon DynamoDB. Estes métodos e classes no lado do cliente correspondem diretamente à API de baixo nível do DynamoDB. No entanto, muitos desenvolvedores experimentam uma sensação de desconexão, ou incompatibilidade de impedância, quando precisam mapear tipos de dados complexos para itens em uma tabela de banco de dados. Com uma interface de banco de dados de baixo nível, os desenvolvedores precisam escrever métodos para a leitura ou a gravação de dados de objetos em tabelas de banco de dados e vice-versa. A quantidade de código extra necessária para cada combinação de tipo de objeto e tabela de banco de dados pode parecer esmagadora.

Para simplificar o desenvolvimento, os AWS SDKs for Java e .NET fornecem interfaces adicionais com níveis mais altos de abstração. As interfaces de nível superior para o DynamoDB permitem que você defina as relações entre objetos no seu programa e as tabelas de banco de dados que armazenam os dados desses objetos. Depois de definir esse mapeamento, você chama métodos de objeto simples, como `save`, `load` ou `delete`, e as operações de baixo nível do DynamoDB subjacente são invocadas automaticamente em seu nome. Isso permite que você escreva um código centrado em objetos, em vez de um código centrado no banco de dados.

Interfaces de programação de nível superior para o DynamoDB estão disponíveis nos AWS SDKs for Java e .NET.

Java

- Java 1.x: DynamoDBMapper
- Java 2.x: Cliente Aprimorado do DynamoDB

.NET

- Trabalhar com o modelo de documento do .NET no DynamoDB
- Trabalhar com o modelo de persistência de objetos do .NET e o DynamoDB

Java 1.x: DynamoDBMapper

> **Note**
>
> O SDK para Java tem duas versões: 1.x e 2.x. O fim do suporte para a versão 1.x foi anunciado em 12 de janeiro de 2024. Ele está previsto para 31 de dezembro de 2025. Para novos desenvolvimentos, é altamente recomendável que você use a versão 2.x.

O AWS SDK para Java fornece uma classe `DynamoDBMapper`, permitindo mapear classes no lado do cliente para tabelas do Amazon DynamoDB. Para usar `DynamoDBMapper`, defina a relação entre os itens em uma tabela do DynamoDB e as instâncias de objeto correspondentes no seu código. A classe `DynamoDBMapper` permite que você realize várias operações de criação, leitura, atualização e exclusão (CRUD) em itens e execute consultas e verificações em tabelas.

Tópicos

- Classe DynamoDBMapper
- Tipos de dados compatíveis com DynamoDBMapper para Java
- Anotações Java para DynamoDB
- Definições de configuração opcionais para DynamoDBMapper
- DynamoDB e bloqueio positivo com número de versão
- Mapear dados arbitrários no DynamoDB
- Exemplos de DynamoDBMapper

> **Note**
>
> A classe `DynamoDBMapper` não permite criar, atualizar ou excluir tabelas. Para realizar essas tarefas, use em vez disso a interface do SDK para Java de baixo nível.

O SDK para Java fornece um conjunto de tipos de anotações, para que você possa mapear suas classes em tabelas. Por exemplo, considere uma tabela `ProductCatalog` cujo `Id` seja a chave de partição.

```
ProductCatalog(Id, ...)
```

É possível mapear uma classe no seu aplicativo cliente para a tabela `ProductCatalog`, conforme mostrado no código Java a seguir. Este código define um objeto Java antigo simples (POJO) chamado `CatalogItem` que usa anotações para mapear campos de objeto em nomes de atributos do DynamoDB.

Example

```
package com.amazonaws.codesamples;

import java.util.Set;

import software.amazon.dynamodb.datamodeling.DynamoDBAttribute;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIgnore;
import software.amazon.dynamodb.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) {this.id = id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN; }

    @DynamoDBAttribute(attributeName="Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
```

```
    public String getSomeProp() { return someProp; }
    public void setSomeProp(String someProp) { this.someProp = someProp; }
}
```

No código anterior, a anotação `@DynamoDBTable` mapeia a classe `CatalogItem` para a tabela `ProductCatalog`. Você pode armazenar instâncias de classes individuais como itens na tabela. Na definição de classe, a anotação `@DynamoDBHashKey` mapeia a propriedade `Id` para a chave primária.

Por padrão, as propriedades da classe são mapeadas para os atributos com o mesmo nome na tabela. As propriedades `Title` e `ISBN` são mapeadas para os atributos com o mesmo nome na tabela.

A anotação `@DynamoDBAttribute` é opcional quando o nome do atributo do DynamoDB corresponde ao nome da propriedade declarada na classe. Quando esses nomes são diferentes, use essa anotação com o parâmetro `attributeName` para especificar a qual atributo do DynamoDB essa propriedade corresponde.

No exemplo anterior, a anotação `@DynamoDBAttribute` é adicionada a cada propriedade para garantir que os nomes de propriedades correspondam exatamente às tabelas criadas em uma etapa anterior e sejam consistentes com os nomes de atributos usados em outros exemplos de código neste guia.

Sua definição de classe pode ter propriedades que não são mapeadas para atributos na tabela. Para identificar essas propriedades, adicione a anotação `@DynamoDBIgnore`. No exemplo anterior, a propriedade `SomeProp` está marcada com a anotação `@DynamoDBIgnore`. Ao carregar uma instância `CatalogItem` na tabela, sua instância `DynamoDBMapper` não inclui a propriedade `SomeProp`. Além disso, o mapeador não retorna esse atributo quando você recupera um item da tabela.

Depois de definir sua classe de mapeamento, é possível usar métodos `DynamoDBMapper` para gravar uma instância dessa classe em um item correspondente na tabela `Catalog`. O exemplo de código a seguir demonstra essa técnica.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapper mapper = new DynamoDBMapper(client);

CatalogItem item = new CatalogItem();
item.setId(102);
```

```
item.setTitle("Book 102 Title");
item.setISBN("222-2222222222");
item.setBookAuthors(new HashSet<String>(Arrays.asList("Author 1", "Author 2")));
item.setSomeProp("Test");

mapper.save(item);
```

O exemplo de código a seguir mostra como recuperar o item e acessar alguns dos seus atributos.

```
CatalogItem partitionKey = new CatalogItem();

partitionKey.setId(102);
DynamoDBQueryExpression<CatalogItem> queryExpression = new
 DynamoDBQueryExpression<CatalogItem>()
    .withHashKeyValues(partitionKey);

List<CatalogItem> itemList = mapper.query(CatalogItem.class, queryExpression);

for (int i = 0; i < itemList.size(); i++) {
    System.out.println(itemList.get(i).getTitle());
    System.out.println(itemList.get(i).getBookAuthors());
}
```

O `DynamoDBMapper` oferece uma forma intuitiva e natural de trabalhar com dados do DynamoDB no Java. Ele também fornece uma série de recursos internos, como bloqueio otimista, transações ACID, valores de chave de partição e chave de classificação gerados automaticamente e versionamento de objetos.

Classe DynamoDBMapper

A classe `DynamoDBMapper` é o ponto de entrada do Amazon DynamoDB. Ela dá acesso a um endpoint do DynamoDB e permite acessar seus dados em várias tabelas. Ela também permite realizar várias operações de criação, leitura, atualização e exclusão (CRUD) em itens e executar consultas e verificações em tabelas. Essa classe fornece os seguintes métodos para trabalhar com o DynamoDB.

Para acessar a documentação Javadoc correspondente, consulte DynamoDBMapper no Referência da API do AWS SDK para Java.

Tópicos

- save
- balanceamento
- excluir
- consulta
- queryPage
- scan
- scanPage
- parallelScan
- batchSave
- batchLoad
- batchDelete
- batchWrite
- transactionWrite
- transactionLoad
- contagem
- generateCreateTableRequest
- createS3Link
- getS3ClientCache

save

Salva o objeto especificado na tabela. O objeto que você deseja salvar é o único parâmetro necessário para esse método. É possível fornecer parâmetros de configuração opcionais usando o objeto `DynamoDBMapperConfig`.

Se um item com a mesma chave primária não existir, esse método criará um novo item na tabela. Se existir um item com a mesma chave primária, ele atualizará o item existente. Se a chave de partição e a chave de classificação forem do tipo String e estiverem anotadas com `@DynamoDBAutoGeneratedKey`, elas receberão um identificador universal exclusivo (UUID) aleatório se não forem inicializadas. Campos de versão anotados com `@DynamoDBVersionAttribute` são incrementados em um. Além disso, se um campo de versão for atualizado ou se uma chave for gerada, o objeto transmitido será atualizado como resultado da operação.

Por padrão, somente atributos correspondentes às propriedades de classe mapeadas são atualizados. Quaisquer atributos existentes em um item não são afetados. No entanto, se você especificar `SaveBehavior.CLOBBER`, poderá forçar o item a ser completamente substituído.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

Se o versionamento estiver habilitado, as versões dos itens no lado do servidor e no lado do cliente deverão corresponder. No entanto, a versão não precisará corresponder se a opção `SaveBehavior.CLOBBER` for usada. Para obter mais informações sobre versionamento, consulte DynamoDB e bloqueio positivo com número de versão.

balanceamento

Recupera um item de uma tabela. É necessário fornecer a chave primária do item que você deseja recuperar. É possível fornecer parâmetros de configuração opcionais usando o objeto `DynamoDBMapperConfig`. Por exemplo, você pode solicitar opcionalmente leituras altamente consistentes para garantir que esse método recupere apenas os valores de itens mais recentes, como mostra a seguinte instrução Java.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

Por padrão, o DynamoDB retorna o item que possui valores finais consistentes. Para obter informações sobre o modelo de consistência eventual do DynamoDB, consulte Consistência de leitura do DynamoDB.

excluir

Exclui um item da tabela. Você deve transmitir uma instância do objeto da classe mapeada.

Se o versionamento estiver habilitado, as versões dos itens no lado do servidor e no lado do cliente deverão corresponder. No entanto, a versão não precisará corresponder se a opção `SaveBehavior.CLOBBER` for usada. Para obter mais informações sobre versionamento, consulte DynamoDB e bloqueio positivo com número de versão.

consulta

Consulta uma tabela ou um índice secundário.

Vamos supor que você tenha uma tabela, `Reply`, que armazena as respostas de tópicos de fórum. Cada assunto de tópico pode ter 0 ou mais respostas. A chave primária da tabela `Reply` consiste nos campos `Id` e `ReplyDateTime`, em que `Id` é a chave de partição e `ReplyDateTime` é a chave de classificação da chave primária.

```
Reply ( Id, ReplyDateTime, ... )
```

Vamos supor que você tenha criado um mapeamento entre uma classe `Reply` e a tabela `Reply` correspondente no DynamoDB. O código Java a seguir usa `DynamoDBMapper` para localizar todas as respostas nas últimas duas semanas para um assunto de tópico específico.

Example

```
String forumName = "&DDB;";
String forumSubject = "&DDB; Thread 1";
String partitionKey = forumName + "#" + forumSubject;

long twoWeeksAgoMilli = (new Date()).getTime() - (14L*24L*60L*60L*1000L);
Date twoWeeksAgo = new Date();
twoWeeksAgo.setTime(twoWeeksAgoMilli);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
String twoWeeksAgoStr = df.format(twoWeeksAgo);

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS(partitionKey));
eav.put(":v2",new AttributeValue().withS(twoWeeksAgoStr.toString()));

DynamoDBQueryExpression<Reply> queryExpression = new DynamoDBQueryExpression<Reply>()
    .withKeyConditionExpression("Id = :v1 and ReplyDateTime > :v2")
    .withExpressionAttributeValues(eav);

List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);
```

A consulta retorna uma coleção de objetos `Reply`.

Por padrão, o método `query` retorna uma coleção de "carregamento preguiçoso". Ele inicialmente retorna apenas uma página de resultados e, em seguida, faz uma chamada de serviço para a

próxima página, se necessário. Para obter todos os itens correspondentes, faça uma iteração na coleção `latestReplies`.

Observe que chamar o método `size()` na coleção carregará todos os resultados para fornecer uma contagem precisa. Isso pode fazer com que uma grande quantidade de throughput provisionado seja consumida, e em uma tabela muito grande pode até esgotar toda a memória na JVM.

Para consultar um índice, você deve primeiro modelá-lo como uma classe de mapeador. Suponha que a tabela `Reply` tenha um índice global secundário chamado PostedBy-Message-Index. A chave de partição para esse índice é `PostedBy`, e a chave de classificação é `Message`. A definição de classe para um item no índice seria semelhante ao seguinte.

```
@DynamoDBTable(tableName="Reply")
public class PostedByMessage {
    private String postedBy;
    private String message;

    @DynamoDBIndexHashKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "PostedBy")
    public String getPostedBy() { return postedBy; }
    public void setPostedBy(String postedBy) { this.postedBy = postedBy; }

    @DynamoDBIndexRangeKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "Message")
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

   // Additional properties go here.
}
```

A anotação `@DynamoDBTable` indica que esse índice está associado à tabela `Reply`. A anotação `@DynamoDBIndexHashKey` representa a chave de partição (PostedBy) do índice, enquanto `@DynamoDBIndexRangeKey` representa a chave de classificação (Message) do índice.

Agora, você pode usar `DynamoDBMapper` para consultar o índice, recuperando um subconjunto das mensagens que foram postadas por um usuário específico. Você não precisará especificar o nome do índice se não tiver mapeamentos conflitantes entre tabelas e índices e se os mapeamentos já tiverem sido feitos no mapeador. O mapeador vai inferir com base na chave primária e na chave de classificação. O código a seguir consulta um índice secundário global. Como um índice secundário global oferece suporte para leituras finais consistentes, mas não para leituras altamente consistentes, é necessário especificar `withConsistentRead(false)`.

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1",  new AttributeValue().withS("User A"));
eav.put(":v2",  new AttributeValue().withS("DynamoDB"));

DynamoDBQueryExpression<PostedByMessage> queryExpression = new
 DynamoDBQueryExpression<PostedByMessage>()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false)
    .withKeyConditionExpression("PostedBy = :v1 and begins_with(Message, :v2)")
    .withExpressionAttributeValues(eav);

List<PostedByMessage> iList =  mapper.query(PostedByMessage.class, queryExpression);
```

A consulta retorna uma coleção de objetos `PostedByMessage`.

queryPage

Consulta uma tabela ou um índice secundário e retorna uma única página de resultados correspondentes. Como acontece com o método `query`, você deve especificar um valor de chave de partição e um filtro de consulta que seja aplicado ao atributo de chave de classificação. No entanto, `queryPage` retorna somente a primeira "página" de dados, ou seja, a quantidade de dados que caberá em 1 MB

scan

Verifica uma tabela ou um índice secundário inteiro. Você tem a opção de especificar uma `FilterExpression` para filtrar o conjunto de resultados.

Vamos supor que você tenha uma tabela, `Reply`, que armazena as respostas de tópicos de fórum. Cada assunto de tópico pode ter 0 ou mais respostas. A chave primária da tabela `Reply` consiste nos campos `Id` e `ReplyDateTime`, em que `Id` é a chave de partição e `ReplyDateTime` é a chave de classificação da chave primária.

```
Reply ( Id, ReplyDateTime, ... )
```

Se você mapeou uma classe Java para a tabela `Reply`, será possível usar `DynamoDBMapper` para verificar essa tabela. Por exemplo, o código Java a seguir verifica a tabela `Reply` inteira, retornando somente as respostas de um determinado ano.

Example

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS("2015"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("begins_with(ReplyDateTime,:v1)")
    .withExpressionAttributeValues(eav);

List<Reply> replies =  mapper.scan(Reply.class, scanExpression);
```

Por padrão, o método `scan` retorna uma coleção de "carregamento preguiçoso". Ele inicialmente retorna apenas uma página de resultados e, em seguida, faz uma chamada de serviço para a próxima página, se necessário. Para obter todos os itens correspondentes, faça uma iteração na coleção `replies`.

Observe que chamar o método `size()` na coleção carregará todos os resultados para fornecer uma contagem precisa. Isso pode fazer com que uma grande quantidade de throughput provisionado seja consumida, e em uma tabela muito grande pode até esgotar toda a memória na JVM.

Para verificar um índice, você deve primeiro modelá-lo como uma classe de mapeador. Suponha que a tabela `Reply` tenha um índice global secundário chamado `PostedBy-Message-Index`. A chave de partição para esse índice é `PostedBy`, e a chave de classificação é `Message`. Uma classe de mapeador para esse índice é mostrada na seção consulta. Ela usa as anotações `@DynamoDBIndexHashKey` e `@DynamoDBIndexRangeKey` para especificar a chave de classificação e a chave de partição do índice.

O código de exemplo a seguir verifica `PostedBy-Message-Index`. Ele não usa um filtro de verificação e, portanto, todos os itens no índice são retornados para você.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

Verifica uma tabela ou um índice secundário e retorna uma única página de resultados correspondentes. Como com o método `scan`, você pode especificar opcionalmente um

`FilterExpression` para filtrar o conjunto de resultados. No entanto, `scanPage` retorna somente a primeira “página” de dados, ou seja, a quantidade de dados que cabe em 1 MB.

parallelScan

Realiza uma verificação paralela de uma tabela ou de um índice secundário inteiro. Você especifica um número de segmentos lógicos para a tabela, juntamente com uma expressão de verificação para filtrar os resultados. O `parallelScan` divide a tarefa de verificação entre vários operadores, um para cada segmento lógico. Por sua vez, esses operadores processam os dados em paralelo e retornam os resultados.

O exemplo de código Java a seguir realiza uma verificação paralela na tabela `Product`.

```
int numberOfThreads = 4;

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":n", new AttributeValue().withN("100"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("Price <= :n")
    .withExpressionAttributeValues(eav);

List<Product> scanResult = mapper.parallelScan(Product.class, scanExpression,
 numberOfThreads);
```

batchSave

Salva os objetos em uma ou mais tabelas usando uma ou mais chamadas para o método `AmazonDynamoDB.batchWriteItem`. Esse método não fornece garantias de transação.

O código Java a seguir salva dois itens (livros) na tabela `ProductCatalog`.

```
Book book1 = new Book();
book1.setId(901);
book1.setProductCategory("Book");
book1.setTitle("Book 901 Title");

Book book2 = new Book();
book2.setId(902);
book2.setProductCategory("Book");
book2.setTitle("Book 902 Title");
```

```
mapper.batchSave(Arrays.asList(book1, book2));
```

batchLoad

Recupera vários itens de uma ou mais tabelas usando suas chaves primárias.

O seguinte código Java recupera dois itens de duas tabelas diferentes.

```
ArrayList<Object> itemsToGet = new ArrayList<Object>();

ForumItem forumItem = new ForumItem();
forumItem.setForumName("Amazon DynamoDB");
itemsToGet.add(forumItem);

ThreadItem threadItem = new ThreadItem();
threadItem.setForumName("Amazon DynamoDB");
threadItem.setSubject("Amazon DynamoDB thread 1 message text");
itemsToGet.add(threadItem);

Map<String, List<Object>> items = mapper.batchLoad(itemsToGet);
```

batchDelete

Exclui objetos de uma ou mais tabelas usando uma ou mais chamadas para o método `AmazonDynamoDB.batchWriteItem`. Esse método não fornece garantias de transação.

O código Java a seguir exclui dois itens (livros) na tabela `ProductCatalog`.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

Salva e exclui objetos em/de uma ou mais tabelas usando uma ou mais chamadas para o método `AmazonDynamoDB.batchWriteItem`. Esse método não oferece garantias de transação ou suporte para versionamento (inserções ou exclusões condicionais).

O código Java a seguir grava um novo item na tabela `Forum`, grava um novo item na tabela `Thread` e exclui um item da tabela `ProductCatalog`.

```
// Create a Forum item to save
```

```
Forum forumItem = new Forum();
forumItem.setName("Test BatchWrite Forum");

// Create a Thread item to save
Thread threadItem = new Thread();
threadItem.setForumName("AmazonDynamoDB");
threadItem.setSubject("My sample question");

// Load a ProductCatalog item to delete
Book book3 = mapper.load(Book.class, 903);

List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
List<Book> objectsToDelete = Arrays.asList(book3);

mapper.batchWrite(objectsToWrite, objectsToDelete);
```

transactionWrite

Salva e exclui objetos em/de uma ou mais tabelas usando uma chamada para o método `AmazonDynamoDB.transactWriteItems`.

Para obter uma lista de exceções específicas de transação, consulte Erros TransactWriteItems.

Para obter mais informações sobre transações do DynamoDB e as garantias de ACID (atomicidade, consistência, isolamento e durabilidade) fornecidas, consulte Amazon DynamoDB Transactions.

> **Note**
>
> Esse método não é compatível com o seguinte:
>
> - DynamoDBMapperConfig.SaveBehavior.

O código Java a seguir grava um novo item em cada uma das tabelas `Forum` e `Thread`, de forma transacional.

```
Thread s3ForumThread = new Thread();
s3ForumThread.setForumName("S3 Forum");
s3ForumThread.setSubject("Sample Subject 1");
s3ForumThread.setMessage("Sample Question 1");

Forum s3Forum = new Forum();
```

```
s3Forum.setName("S3 Forum");
s3Forum.setCategory("Amazon Web Services");
s3Forum.setThreads(1);

TransactionWriteRequest transactionWriteRequest = new TransactionWriteRequest();
transactionWriteRequest.addPut(s3Forum);
transactionWriteRequest.addPut(s3ForumThread);
mapper.transactionWrite(transactionWriteRequest);
```

transactionLoad

Carrega objetos de uma ou mais tabelas usando uma chamada para o método `AmazonDynamoDB.transactGetItems`.

Para obter uma lista de exceções específicas de transação, consulte Erros TransactGetItems.

Para obter mais informações sobre transações do DynamoDB e as garantias de ACID (atomicidade, consistência, isolamento e durabilidade) fornecidas, consulte Amazon DynamoDB Transactions.

O código Java a seguir carrega um item de cada uma das tabelas `Forum` e `Thread`, de forma transacional.

```
Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB Forum");

TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();
transactionLoadRequest.addLoad(dynamodbForum);
transactionLoadRequest.addLoad(dynamodbForumThread);
mapper.transactionLoad(transactionLoadRequest);
```

contagem

Avalia a expressão de verificação especificada e retorna a contagem de itens correspondentes. Dados de itens não são retornados.

generateCreateTableRequest

Analisa uma classe POJO que representa um tabela do DynamoDB e retorna um `CreateTableRequest` para essa tabela.

createS3Link

Cria um link para um objeto no Amazon S3. Você deve especificar um nome de bucket e um nome de chave que identifique exclusivamente o objeto no bucket.

Para usar `createS3Link`, a sua classe de mapeador deve definir métodos getter e setter. O exemplo de código a seguir ilustra isso, adicionando um novo atributo e métodos getter/setter à classe `CatalogItem`:

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    ...

    public S3Link productImage;

    ....

    @DynamoDBAttribute(attributeName = "ProductImage")
    public S3Link getProductImage() {
            return productImage;
    }

    public void setProductImage(S3Link productImage) {
        this.productImage = productImage;
    }

...
}
```

O código Java a seguir define um novo item a ser gravado na tabela `Product`. O item inclui um link para uma imagem do produto. Os dados da imagem são carregados no Amazon S3.

```
CatalogItem item = new CatalogItem();

item.setId(150);
item.setTitle("Book 150 Title");

String amzn-s3-demo-bucket = "amzn-s3-demo-bucket";
String myS3Key = "productImages/book_150_cover.jpg";
item.setProductImage(mapper.createS3Link(amzn-s3-demo-bucket, myS3Key));
```

```
item.getProductImage().uploadFrom(new File("/file/path/book_150_cover.jpg"));

mapper.save(item);
```

A classe `S3Link` fornece muitos outros métodos para manipular objetos no Amazon S3. Para obter mais informações, consulte os Javadocs para `S3Link`.

getS3ClientCache

Retorna o `S3ClientCache` subjacente para acessar o Amazon S3. Um `S3ClientCache` é um Mapa inteligente para objetos `AmazonS3Client`. Se você tem vários clientes, um `S3ClientCache` pode ajudá-lo a manter esses clientes organizados por região da AWS e pode criar novos clientes do Amazon S3 sob demanda.

Tipos de dados compatíveis com DynamoDBMapper para Java

Esta seção descreve os tipos de dados arbitrários, as coleções e os tipos de dados Java primitivos com aceitos no Amazon DynamoDB.

O Amazon DynamoDB oferece suporte aos seguintes tipos de dados Java primitivos e classes wrapper primitivas.

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`
- `Date` (como a string de precisão em milissegundos ISO_8601, modificada para UTC)
- `Calendar` (como a string de precisão em milissegundos ISO_8601, modificada para UTC)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`
- `BigInteger`

> **Note**
>
> - Para obter mais informações sobre as regras de nomenclatura do DynamoDB e os vários tipos de dados compatíveis, consulte Tipos de dados compatíveis e regras de nomenclatura no Amazon DynamoDB.
> - O DynamoDBMapper oferece suporte a valores Binary vazios.
> - Valores String vazios são compatíveis com o AWS SDK for Java 2.x.
>
> No AWS SDK para Java 1.x, o DynamoDBMapper oferece suporte à leitura de valores de atributos String vazios. No entanto, ele não gravará valores de atributos String vazios porque esses atributos são descartados da solicitação.

O DynamoDB oferece suporte aos tipos de coleção Java Set, List e Map. A tabela a seguir resume como esses tipos Java são mapeados nos tipos do DynamoDB.

Tipo Java	Tipo do DynamoDB
Todos os tipos de número	`N` (tipo Número)
Strings	`S` (tipo String)
Booleano	`BOOL` (tipo booliano), 0 ou 1.
ByteBuffer	`B` (tipo Binário)
Data	`S` (tipo String). Os valores de Date são armazenados como strings formatadas em ISO-8601.
Tipos de coleção Set	Tipo `SS` (conjunto de strings), tipo `NS` (conjunto de números) e tipo `BS` (conjunto de binários).

A interface `DynamoDBTypeConverter` permite que você mapeie seus próprios tipos de dados arbitrários em um tipo de dados com suporte nativo do DynamoDB. Para obter mais informações, consulte Mapear dados arbitrários no DynamoDB.

Anotações Java para DynamoDB

Esta seção descreve as anotações que estão disponíveis para mapear suas classes e propriedades em tabelas e atributos no Amazon DynamoDB.

Para conhecer a documentação Javadoc correspondente, consulte Resumo de tipos de anotação na Referência da API do AWS SDK para Java.

> **Note**
>
> Nas seguintes anotações, apenas `DynamoDBTable` e `DynamoDBHashKey` são necessários.

Tópicos

- DynamoDBAttribute
- DynamoDBAutoGeneratedKey
- DynamoDBAutoGeneratedTimestamp
- DynamoDBDocument
- DynamoDBHashKey
- DynamoDBIgnore
- DynamoDBIndexHashKey
- DynamoDBIndexRangeKey
- DynamoDBRangeKey
- DynamoDBTable
- DynamoDBTypeConverted
- DynamoDBTyped
- DynamoDBVersionAttribute

DynamoDBAttribute

Mapeia uma propriedade para um atributo de tabela. Por padrão, cada propriedade de classe é mapeada para um atributo de item com o mesmo nome. No entanto, se os nomes não forem os mesmos, você poderá usar essa anotação para mapear uma propriedade para o atributo. No

seguinte trecho de código Java, `DynamoDBAttribute` mapeia a propriedade `BookAuthors` para o nome de atributo `Authors` na tabela.

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

O `DynamoDBMapper` usa `Authors` como o nome do atributo ao salvar o objeto na tabela.

DynamoDBAutoGeneratedKey

Marca uma propriedade de chave de partição ou de chave de classificação como sendo gerada automaticamente. `DynamoDBMapper` gera um UUID aleatório ao salvar esses atributos. Apenas propriedades String podem ser marcadas como chaves geradas automaticamente.

O exemplo a seguir demonstra o uso de chaves geradas automaticamente.

```
@DynamoDBTable(tableName="AutoGeneratedKeysExample")
public class AutoGeneratedKeys {
    private String id;
    private String payload;

    @DynamoDBHashKey(attributeName = "Id")
    @DynamoDBAutoGeneratedKey
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    @DynamoDBAttribute(attributeName="payload")
    public String getPayload() { return this.payload; }
    public void setPayload(String payload) { this.payload = payload; }

    public static void saveItem() {
        AutoGeneratedKeys obj = new AutoGeneratedKeys();
        obj.setPayload("abc123");

        // id field is null at this point
        DynamoDBMapper mapper = new DynamoDBMapper(dynamoDBClient);
        mapper.save(obj);

        System.out.println("Object was saved with id " + obj.getId());
    }
```

```
}
```

DynamoDBAutoGeneratedTimestamp

Gera automaticamente um carimbo de data/hora.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

Opcionalmente, a estratégia de geração automática pode ser definida fornecendo um atributo de estratégia. O padrão é `ALWAYS`.

DynamoDBDocument

Indica que uma classe pode ser serializada como um documento do Amazon DynamoDB.

Por exemplo, vamos supor que você queria mapear um documento JSON em um atributo do DynamoDB do tipo Map (`M`). O exemplo de código a seguir define um item que contém um atributo aninhado (Pictures) do tipo Map.

```
public class ProductCatalogItem {

    private Integer id;  //partition key
    private Pictures pictures;
    /* ...other attributes omitted... */

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id;}
    public void setId(Integer id) {this.id = id;}

    @DynamoDBAttribute(attributeName="Pictures")
    public Pictures getPictures() { return pictures;}
    public void setPictures(Pictures pictures) {this.pictures = pictures;}

    // Additional properties go here.

    @DynamoDBDocument
    public static class Pictures {
        private String frontView;
        private String rearView;
```

```
        private String sideView;

        @DynamoDBAttribute(attributeName = "FrontView")
        public String getFrontView() { return frontView; }
        public void setFrontView(String frontView) { this.frontView = frontView; }

        @DynamoDBAttribute(attributeName = "RearView")
        public String getRearView() { return rearView; }
        public void setRearView(String rearView) { this.rearView = rearView; }

        @DynamoDBAttribute(attributeName = "SideView")
        public String getSideView() { return sideView; }
        public void setSideView(String sideView) { this.sideView = sideView; }

    }
}
```

É possível salvar um novo item `ProductCatalog`, com `Pictures`, conforme mostrado no exemplo a seguir.

```
ProductCatalogItem item = new ProductCatalogItem();

Pictures pix = new Pictures();
pix.setFrontView("http://example.com/products/123_front.jpg");
pix.setRearView("http://example.com/products/123_rear.jpg");
pix.setSideView("http://example.com/products/123_left_side.jpg");
item.setPictures(pix);

item.setId(123);

mapper.save(item);
```

O item de `ProductCatalog` resultante seria semelhante ao seguinte (no formato JSON).

```
{
  "Id" : 123
  "Pictures" : {
    "SideView" : "http://example.com/products/123_left_side.jpg",
    "RearView" : "http://example.com/products/123_rear.jpg",
    "FrontView" : "http://example.com/products/123_front.jpg"
  }
}
```

DynamoDBHashKey

Mapeia uma propriedade de classe para a chave de partição da tabela. A propriedade deve ser uma string escalar, número ou tipos binários. A propriedade não pode ser um tipo de coleção.

Vamos supor que você tenha uma tabela, `ProductCatalog`, com `Id` como chave primária. O código Java a seguir define uma classe `CatalogItem` e mapeia sua propriedade `Id` para a chave primária da tabela `ProductCatalog` usando a tag `@DynamoDBHashKey`.

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {
    private Integer Id;
   @DynamoDBHashKey(attributeName="Id")
   public Integer getId() {
        return Id;
   }
   public void setId(Integer Id) {
        this.Id = Id;
   }
   // Additional properties go here.
}
```

DynamoDBIgnore

Indica à instância de `DynamoDBMapper` que a propriedade associada deve ser ignorada. Quando salvar dados na tabela, o `DynamoDBMapper` não salva essa propriedade na tabela.

Aplicado ao método getter ou ao campo de classe de uma propriedade não modelada. Se a anotação for aplicada diretamente ao campo de classe, o getter e o setter correspondentes deverão ser declarados na mesma classe.

DynamoDBIndexHashKey

Mapeia uma propriedade de classe na chave de partição de um índice secundário global. A propriedade deve ser uma string escalar, número ou tipos binários. A propriedade não pode ser um tipo de coleção.

Use essa anotação se você precisa realizar uma `Query` em um índice secundário global. É necessário especificar o nome de índice (`globalSecondaryIndexName`). Se o nome da propriedade da classe for diferente da chave de partição do índice, também será necessário especificar o nome desse atributo de índice (`attributeName`).

DynamoDBIndexRangeKey

Mapeia uma propriedade de classe na chave de classificação de um índice secundário global ou índice secundário local. A propriedade deve ser uma string escalar, número ou tipos binários. A propriedade não pode ser um tipo de coleção.

Use essa anotação se você precisa realizar uma `Query` em um índice secundário local ou índice secundário global e deseja refinar seus resultados usando a chave de classificação do índice. É necessário especificar o nome de índice (`globalSecondaryIndexName` ou `localSecondaryIndexName`). Se o nome da propriedade da classe for diferente da chave de classificação do índice, você também deve especificar o nome desse atributo de índice (`attributeName`).

DynamoDBRangeKey

Mapeia uma propriedade de classe para a chave de classificação da tabela. A propriedade deve ser uma string escalar, número ou tipos binários. Não pode ser um tipo de coleção.

Se a chave primária for composta (chave de partição e chave de classificação), você poderá usar essa tag para mapear seu campo de classe para a chave de classificação. Por exemplo, vamos supor que você tenha uma tabela `Reply` que armazena respostas para tópicos de fórum. Cada tópico pode ter muitas respostas. Portanto, a chave primária dessa tabela é tanto `ThreadId` quanto `ReplyDateTime`. `ThreadId` é a chave de partição, e `ReplyDateTime` é a chave de classificação.

O código Java a seguir define uma classe `Reply` e a mapeia para a tabela `Reply`. Ele usa ambas as tags `@DynamoDBHashKey` e `@DynamoDBRangeKey` para identificar propriedades de classes que são mapeadas para a chave primária.

```
@DynamoDBTable(tableName="Reply")
public class Reply {
    private Integer id;
    private String replyDateTime;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    @DynamoDBRangeKey(attributeName="ReplyDateTime")
    public String getReplyDateTime() { return replyDateTime; }
    public void setReplyDateTime(String replyDateTime) { this.replyDateTime =
 replyDateTime; }
```

```
   // Additional properties go here.
}
```

DynamoDBTable

Identifica a tabela de destino no DynamoDB. Por exemplo, o seguinte código Java define uma classe `Developer` e a mapeia na tabela `People` no DynamoDB.

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

A anotação `@DynamoDBTable` pode ser herdada. Qualquer nova classe que herde da classe `Developer` também é mapeada para a tabela `People`. Por exemplo, suponha que você crie uma classe `Lead` que herde da classe `Developer`. Como você mapeou a classe `Developer` para a tabela `People`, e os objetos de classe `Lead` também são armazenados na mesma tabela.

Também é possível substituir `@DynamoDBTable`. Qualquer nova classe que herde da classe `Developer` por padrão é mapeada para a mesma tabela `People`. No entanto, você pode substituir esse mapeamento padrão. Por exemplo, se você criar uma classe que herda da classe `Developer`, será possível mapeá-la explicitamente para outra tabela adicionando a anotação `@DynamoDBTable`, conforme mostrado no seguinte exemplo de código Java.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDBTypeConverted

Uma anotação para marcar uma propriedade como usando um conversor de tipo personalizado. Pode ser anotada em uma anotação definida pelo usuário para transmitir propriedades adicionais a `DynamoDBTypeConverter`.

A interface `DynamoDBTypeConverter` permite que você mapeie seus próprios tipos de dados arbitrários em um tipo de dados com suporte nativo do DynamoDB. Para obter mais informações, consulte Mapear dados arbitrários no DynamoDB.

DynamoDBTyped

Uma anotação para substituir a associação de tipo de atributo padrão. Tipos padrão não exigem a anotação se estiverem aplicando a associação de atributo padrão para esse tipo.

DynamoDBVersionAttribute

Identifica uma propriedade de classe para armazenar um número de versão de bloqueio otimista. `DynamoDBMapper` atribui um número de versão a essa propriedade ao salvar um novo item e o incrementa cada vez que você atualizar o item. Apenas há suporte para tipos de números escalares. Para obter informações sobre tipos de dados, consulte Tipos de dados. Para obter mais informações sobre versionamento, consulte DynamoDB e bloqueio positivo com número de versão.

Definições de configuração opcionais para DynamoDBMapper

Quando você cria uma instância de `DynamoDBMapper`, ela tem certos comportamentos padrão que podem ser substituídos com o uso da classe `DynamoDBMapperConfig`.

O seguinte trecho de código cria um `DynamoDBMapper` com configurações personalizadas:

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapperConfig mapperConfig = DynamoDBMapperConfig.builder()
        .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
        .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
        .withTableNameOverride(null)

 .withPaginationLoadingStrategy(DynamoDBMapperConfig.PaginationLoadingStrategy.EAGER_LOADING)
    .build();

DynamoDBMapper mapper = new DynamoDBMapper(client, mapperConfig);
```

Para obter mais informações, consulte DynamoDBMapperConfig na Referência da API do AWS SDK para Java.

Como alternativa, você pode usar os seguintes argumentos para uma instância de `DynamoDBMapperConfig`:

- Um valor de enumeração `DynamoDBMapperConfig.ConsistentReads`:
  - `EVENTUAL`: a instância de mapeador usa uma solicitação de leitura eventualmente consistente.
  - `CONSISTENT`: a instância do mapeador usa uma solicitação de leitura fortemente consistente. É possível usar essa configuração opcional com operações `load`, `query` ou `scan`. Leituras altamente consistentes têm implicações sobre a performance e a cobrança. Consulte a página de detalhes do produto do DynamoDB para obter mais informações.

Se você não especificar uma configuração de consistência de leitura para sua instância de mapeador, o padrão será `EVENTUAL`.

> **Note**
>
> Esse valor é aplicado em operações `query`, `querypage`, `load` e `batch load` do DynamoDBMapper.

- Um valor de enumeração `DynamoDBMapperConfig.PaginationLoadingStrategy`: controla como a instância de mapeador processa uma lista paginada de dados, como os resultados de um `query` ou `scan`:
  - `LAZY_LOADING`: a instância do mapeador carrega dados quando possível e mantém todos os resultados carregados na memória.
  - `EAGER_LOADING`: a instância de mapeador carrega os dados assim que a lista é inicializada.
  - `ITERATION_ONLY`: você só pode usar um iterador para ler da lista. Durante a iteração, a lista limpará todos os resultados anteriores antes de carregar a próxima página e, portanto, ela manterá no máximo uma página dos resultados carregados na memória. Isso também significa que a lista só pode ser iterada uma vez. Essa estratégia é recomendada ao lidar com itens grandes, a fim de reduzir a sobrecarga de memória.

  Se você não especificar uma estratégia de carregamento de paginação para a sua instância de mapeador, o padrão será `LAZY_LOADING`.
- Um valor de enumeração `DynamoDBMapperConfig.SaveBehavior`: especifica como a instância de mapeador deve lidar com atributos durante operações de salvamento:
  - `UPDATE`: durante uma operação de gravação, todos os atributos modelados são atualizados, enquanto os atributos não modelados não são afetados. Tipos de números primitivos (byte, int, long) são definidos como 0. Tipos de objetos são definidos como nulos.
  - `CLOBBER`: limpa e substitui todos os atributos, incluindo os não modelados, durante uma operação de gravação. Isso é feito excluindo-se o item e o recriando. Restrições de campo com versionamento também são desconsideradas.

  Se você não especificar o comportamento de salvamento para sua instância de mapeador, o padrão será `UPDATE`.

> **Note**
>
> As operações transacionais do DynamoDBMapper não são compatíveis com a enumeração do `DynamoDBMapperConfig.SaveBehavior`.

- Um objeto `DynamoDBMapperConfig.TableNameOverride` instrui a instância do mapeador a ignorar o nome de tabela especificado pela anotação `DynamoDBTable` de uma classe e, em vez disso, usar um nome de tabela diferente que você fornece. Isso é útil ao particionar dados em várias tabelas no tempo de execução.

Você pode substituir o objeto de configuração padrão para `DynamoDBMapper` por operação, conforme necessário.

DynamoDB e bloqueio positivo com número de versão

O bloqueio positivo é uma estratégia para garantir que o item no lado do cliente que você está atualizando (ou excluindo) seja o mesmo que o item no Amazon DynamoDB. Se você usar essa estratégia, suas gravações de banco de dados serão protegidas contra substituição pelas gravações de outros e vice-versa.

Com o bloqueio positivo, cada item tem um atributo que serve como um número de versão. Se você recuperar um item de uma tabela, o aplicativo registrará o número da versão desse item. Você poderá atualizar o item somente se o número de versão no lado do servidor não tiver sido alterado. Se há uma incompatibilidade de versão, isso significa que alguém modificou o item antes de você. A tentativa de atualização falha, porque você tem uma versão obsoleta do item. Se isso acontecer, basta tentar novamente ao recuperar o item e tentar atualizá-lo. O bloqueio positivo impede que você substitua acidentalmente as alterações que foram feitas por outros. Também impede que outros substituam acidentalmente suas alterações.

Embora seja possível implementar a própria estratégia de bloqueio positivo, o AWS SDK para Java oferece a anotação `@DynamoDBVersionAttribute`. Na classe de mapeamento da sua tabela, você designa uma propriedade para armazenar o número da versão e a marca usando essa anotação. Quando um objeto é salvo, o item correspondente na tabela do DynamoDB terá um atributo que armazena o número da versão. O `DynamoDBMapper` atribui um número de versão quando você salvar o objeto pela primeira vez e incrementa automaticamente o número da versão sempre que você atualiza o item. Suas solicitações de atualização ou exclusão só

serão bem-sucedidas se a versão do objeto no lado do cliente corresponder ao número de versão correspondente do item na tabela do DynamoDB.

`ConditionalCheckFailedException` será lançada se:

- Você usar bloqueio positivo com `@DynamoDBVersionAttribute` e o valor de versão no servidor for diferente do valor no lado do cliente.
- Você especificar suas próprias restrições condicionais ao salvar dados usando `DynamoDBMapper` com `DynamoDBSaveExpression` e ocorrer falha nessas restrições.

> **Note**
>
> - As tabelas globais do DynamoDB usam uma reconciliação "último gravador ganha" entre as atualizações simultâneas. Se você usa tabelas globais, a política de último gravador ganha. Portanto, neste caso, a estratégia de bloqueio não funciona como esperado.
> - As operações de gravação transacional `DynamoDBMapper` não são compatíveis com expressões de condição e anotação `@DynamoDBVersionAttribute` no mesmo objeto. Se um objeto em uma gravação transacional for anotado com `@DynamoDBVersionAttribute` e também tiver uma expressão de condição, a SdkClientException será lançada.

Por exemplo, o código Java a seguir define uma classe `CatalogItem` que tem várias propriedades. A propriedade `Version` está marcada com a anotação `@DynamoDBVersionAttribute`.

Example

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;
    private Long version;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
```

```
    public void setId(Integer Id) { this.id = Id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN;}

    @DynamoDBAttribute(attributeName = "Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp;}
    public void setSomeProp(String someProp) {this.someProp = someProp;}

    @DynamoDBVersionAttribute
    public Long getVersion() { return version; }
    public void setVersion(Long version) { this.version = version;}
}
```

Você pode aplicar a anotação `@DynamoDBVersionAttribute` a tipos anuláveis fornecidos pelas classes wrapper primitivas que fornecem um tipo que permite valor nulo, como `Long` e `Integer`.

O bloqueio positivo tem o seguinte impacto sobre estes métodos `DynamoDBMapper`:

- `save`: para um novo item, `DynamoDBMapper` atribui um número de versão inicial 1. Se você recuperar um item, atualizar uma ou mais das suas propriedades e tentar salvar as alterações, a operação de salvamento será bem-sucedida somente se o número de versão no lado do cliente e no lado do servidor corresponderem. A classe `DynamoDBMapper` incrementa o número de versão automaticamente.
- `delete`: o método `delete` usa um objeto como parâmetro, e `DynamoDBMapper` realiza uma verificação de versão antes de excluir o item. A verificação da versão pode ser desabilitada se `DynamoDBMapperConfig.SaveBehavior.CLOBBER` for especificado na solicitação.

  A implementação interna do bloqueio positivo em `DynamoDBMapper` usa o suporte a atualizações e exclusões condicionais fornecido pelo DynamoDB.
- `transactionWrite` —

- `Put`: para um novo item, `DynamoDBMapper` atribui um número de versão inicial 1. Se você recuperar um item, atualizar uma ou mais das suas propriedades e tentar salvar as alterações, a operação Put será bem-sucedida somente se o número de versão no lado do cliente e no lado do servidor corresponder. A classe `DynamoDBMapper` incrementa o número de versão automaticamente.
- `Update`: para um novo item, `DynamoDBMapper` atribui um número de versão inicial 1. Se você recuperar um item, atualizar uma ou mais das suas propriedades e tentar salvar as alterações, a operação Update será bem-sucedida somente se o número de versão no lado do cliente e no lado do servidor corresponder. A classe `DynamoDBMapper` incrementa o número de versão automaticamente.
- `Delete`: o `DynamoDBMapper` realiza uma verificação de versão antes de excluir o item. A operação Delete só será bem-sucedida se o número de versão no lado do cliente e no lado do servidor corresponder.
- `ConditionCheck`: a anotação `@DynamoDBVersionAttribute` não é compatível com operações `ConditionCheck`. Uma SdkClientException será lançada quando um item `ConditionCheck` for anotado com `@DynamoDBVersionAttribute`.

Desabilitar o bloqueio positivo

Para desabilitar o bloqueio positivo, você pode alterar o valor de enumeração `DynamoDBMapperConfig.SaveBehavior` de `UPDATE` para `CLOBBER`. Você pode fazer isso criando uma instância de `DynamoDBMapperConfig` que ignora a verificação de versão e usando essa instância para todas as suas solicitações. Para obter informações sobre `DynamoDBMapperConfig.SaveBehavior` e outros parâmetros `DynamoDBMapper` opcionais, consulte Definições de configuração opcionais para DynamoDBMapper .

Você também pode definir um comportamento de bloqueio somente para uma operação específica. Por exemplo, o seguinte trecho de código Java usa `DynamoDBMapper` para salvar um item de catálogo. Ele especifica `DynamoDBMapperConfig.SaveBehavior` adicionando o parâmetro `DynamoDBMapperConfig` opcional ao método `save`.

> **Note**
>
> O método transactionWrite não oferece suporte à configuração DynamoDBMapperConfig.SaveBehavior. A desabilitação do bloqueio positivo para transactionWrite é incompatível.

Example

```
DynamoDBMapper mapper = new DynamoDBMapper(client);

// Load a catalog item.
CatalogItem item = mapper.load(CatalogItem.class, 101);
item.setTitle("This is a new title for the item");
...
// Save the item.
mapper.save(item,
    new DynamoDBMapperConfig(
        DynamoDBMapperConfig.SaveBehavior.CLOBBER));
```

Mapear dados arbitrários no DynamoDB

Além dos tipos de Java compatíveis (consulte Tipos de dados compatíveis com DynamoDBMapper para Java), é possível usar tipos em sua aplicação para os quais não há um mapeamento direto para os tipos do Amazon DynamoDB. Para mapear esses tipos, é necessário fornecer uma implementação que converta o tipo complexo em um tipo compatível com o DynamoDB e vice-versa, e anotar o método de acesso de tipos complexos usando a anotação `@DynamoDBTypeConverted`. O código de conversor transforma os dados quando os objetos são salvos ou carregados. Ele também é usado para todas as operações que consomem tipos complexos. Observe que, ao comparar dados durante operações de consulta e verificação, as comparações são feitas com os dados armazenados no DynamoDB.

Por exemplo, considere a seguinte classe `CatalogItem`, que define uma propriedade, `Dimension`, que é de `DimensionType`. Essa propriedade armazena as dimensões de itens, como altura, largura e espessura. Suponha que você decida armazenar essas dimensões de itens como uma string (como 8.5x11x.05) no DynamoDB. O exemplo a seguir fornece o código de conversor que converte o objeto `DimensionType` em uma string e uma string em `DimensionType`.

> **Note**
>
> Este exemplo de código pressupõe que você já carregou dados no DynamoDB para sua conta seguindo as instruções na seção Criar tabelas e carregar dados para exemplos de código no DynamoDB.
> Para obter instruções passo a passo sobre como executar o exemplo a seguir, consulte Exemplos de código Java.

Example

```
public class DynamoDBMapperExample {

    static AmazonDynamoDB client;

    public static void main(String[] args) throws IOException {

        // Set the AWS region you want to access.
        Regions usWest2 = Regions.US_WEST_2;
        client = AmazonDynamoDBClientBuilder.standard().withRegion(usWest2).build();

        DimensionType dimType = new DimensionType();
        dimType.setHeight("8.00");
        dimType.setLength("11.0");
        dimType.setThickness("1.0");

        Book book = new Book();
        book.setId(502);
        book.setTitle("Book 502");
        book.setISBN("555-5555555555");
        book.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));
        book.setDimensions(dimType);

        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(book);

        Book bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Book info: " + "\n" + bookRetrieved);

        bookRetrieved.getDimensions().setHeight("9.0");
        bookRetrieved.getDimensions().setLength("12.0");
        bookRetrieved.getDimensions().setThickness("2.0");

        mapper.save(bookRetrieved);

        bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Updated book info: " + "\n" + bookRetrieved);
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
```

```
        private String ISBN;
        private Set<String> bookAuthors;
        private DimensionType dimensionType;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Authors")
        public Set<String> getBookAuthors() {
            return bookAuthors;
        }

        public void setBookAuthors(Set<String> bookAuthors) {
            this.bookAuthors = bookAuthors;
        }

        @DynamoDBTypeConverted(converter = DimensionTypeConverter.class)
        @DynamoDBAttribute(attributeName = "Dimensions")
        public DimensionType getDimensions() {
```

```
            return dimensionType;
        }

        @DynamoDBAttribute(attributeName = "Dimensions")
        public void setDimensions(DimensionType dimensionType) {
            this.dimensionType = dimensionType;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ",
dimensionType= "
                    + dimensionType.getHeight() + " X " + dimensionType.getLength() + "
X "
                    + dimensionType.getThickness()
                    + ", Id=" + id + ", Title=" + title + "]";
        }
    }

    static public class DimensionType {

        private String length;
        private String height;
        private String thickness;

        public String getLength() {
            return length;
        }

        public void setLength(String length) {
            this.length = length;
        }

        public String getHeight() {
            return height;
        }

        public void setHeight(String height) {
            this.height = height;
        }

        public String getThickness() {
            return thickness;
        }
```

```
        public void setThickness(String thickness) {
            this.thickness = thickness;
        }
    }

    // Converts the complex type DimensionType to a string and vice-versa.
    static public class DimensionTypeConverter implements DynamoDBTypeConverter<String,
DimensionType> {

        @Override
        public String convert(DimensionType object) {
            DimensionType itemDimensions = (DimensionType) object;
            String dimension = null;
            try {
                if (itemDimensions != null) {
                    dimension = String.format("%s x %s x %s",
itemDimensions.getLength(), itemDimensions.getHeight(),
                            itemDimensions.getThickness());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return dimension;
        }

        @Override
        public DimensionType unconvert(String s) {

            DimensionType itemDimension = new DimensionType();
            try {
                if (s != null && s.length() != 0) {
                    String[] data = s.split("x");
                    itemDimension.setLength(data[0].trim());
                    itemDimension.setHeight(data[1].trim());
                    itemDimension.setThickness(data[2].trim());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return itemDimension;
        }
    }
```

```
}
```

Exemplos de DynamoDBMapper

O AWS SDK para Java fornece uma classe `DynamoDBMapper`, permitindo mapear classes do lado do cliente para tabelas do DynamoDB. Para usar `DynamoDBMapper`, defina a relação entre os itens em uma tabela do DynamoDB e as instâncias de objeto correspondentes no seu código. A classe `DynamoDBMapper` permite que você realize várias operações de criação, leitura, atualização e exclusão (CRUD) em itens e execute consultas e verificações em tabelas.

Para saber mais sobre como usar `DynamoDBMapper`, consulte DynamoDB Examples Using the AWS SDK for Java (Exemplos do DynamoDB usando o SDK para Java) no Guia do desenvolvedor do AWS SDK para Java 1.x.

Java 2.x: Cliente Aprimorado do DynamoDB

O Cliente Aprimorado do DynamoDB é uma biblioteca de alto nível que faz parte do AWS SDK para Java versão 2 (v2). Ele oferece uma maneira simples de mapear classes do lado do cliente para tabelas do DynamoDB. Basta definir as relações entre as tabelas e suas classes de modelo correspondentes no seu código. Depois que você definir essas relações, poderá executar intuitivamente várias operações de criação, leitura, atualização ou exclusão (CRUD) em tabelas ou itens do DynamoDB.

Para obter mais informações sobre como você pode usar o cliente aprimorado com o DynamoDB, consulte Usar o Cliente Aprimorado do DynamoDB no AWS SDK para Java 2.x.

Trabalhar com o modelo de documento do .NET no DynamoDB

O AWS SDK para .NET fornece classes do modelo de documento para empacotar algumas das operações de baixo nível do Amazon DynamoDB, simplificando ainda mais sua codificação. No modelo de documento, as classes primárias são `Table` e `Document`. A classe `Table` fornece métodos de operação de dados, como `PutItem`, `GetItem` e `DeleteItem`. Ela também fornece os métodos `Query` e `Scan`. A classe `Document` representa um único item em uma tabela.

As classes do modelo de documento anteriores estão disponíveis no namespace `Amazon.DynamoDBv2.DocumentModel`.

> **Note**
>
> Não é possível usar as classes do modelo de documento para criar, atualizar e excluir tabelas. Por outro lado, o modelo de documento oferece suporte à maioria das operações de dados comuns.

Tópicos

- Tipos de dados compatíveis

Tipos de dados compatíveis

O modelo de documento oferece suporte a um conjunto de tipos de dados .NET primitivos e tipos de dados de coleções. O modelo é compatível com os seguintes tipos de dados primitivos.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Guid`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

A tabela a seguir resume o mapeamento dos tipos .NET anteriores nos tipos do DynamoDB.

Tipo primitivo .NET	Tipo do DynamoDB
Todos os tipos de número	`N` (tipo Número)
Todos os tipos de string	`S` (tipo String)
MemoryStream, byte[]	`B` (tipo Binário)
bool	`N` (tipo numérico), 0 representa false 1 representa true.
DateTime	`S` (tipo String). Os valores de `DateTime` são armazenados como strings formatadas em ISO-8601.
Guid	`S` (tipo String).
Tipos de coleção (List, HashSet e array)	Tipo `BS` (conjunto binário), tipo `SS` (conjunto de strings) e tipo `NS` (conjunto de números)

O AWS SDK para .NET define tipos para mapear os tipos Boolean, Null, List e Map do DynamoDB na API do modelo de documento do .NET:

- Use `DynamoDBBool` para o tipo booliano.
- Use `DynamoDBNull` para o tipo nulo.
- Use `DynamoDBList` para o tipo lista.
- Use `Document` para o tipo mapa.

> **Note**
>
> - Valores binários vazios são compatíveis.
> - A leitura de valores string vazios é compatível. Os valores de atributos string vazios são compatíveis nos valores de atributos do tipo de conjunto string durante a gravação no DynamoDB. Os valores de atributos string vazios do tipo string e os valores string vazios contidos no tipo Lista ou Mapa são descartados das solicitações de gravação

Trabalhar com o modelo de persistência de objetos do .NET e o DynamoDB

O AWS SDK para .NET fornece um modelo de persistência de objetos que permite mapear classes do cliente em tabelas do Amazon DynamoDB. Em seguida, cada instância de objeto é mapeada para um item nas tabelas correspondentes. Para salvar objetos no lado do cliente nas tabelas, o modelo de persistência de objetos fornece a classe `DynamoDBContext`, um ponto de entrada para o DynamoDB. Esta classe fornece uma conexão ao DynamoDB e permite que você acesse tabelas, execute várias operações CRUD e realize consultas.

O modelo de persistência de objetos fornece um conjunto de atributos para mapear classes no lado do cliente para tabelas e propriedades/campos para atributos de tabela.

> **Note**
>
> O modelo de persistência de objetos não fornece uma API para criar, atualizar ou excluir tabelas. Ele fornece apenas operações de dados. É possível usar somente a API de baixo nível do AWS SDK para .NET para criar, atualizar e excluir tabelas.

O exemplo a seguir mostra como o modelo de persistência de objetos funciona. Ele começa com a tabela `ProductCatalog`. `Id` é a chave primária.

```
ProductCatalog(Id, ...)
```

Suponha que você tenha uma classe `Book` com propriedades `Title`, `ISBN` e `Authors`. É possível mapear a classe `Book` na tabela `ProductCatalog` adicionando os atributos definidos pelo modelo de persistência de objetos, conforme mostrado no trecho de código C# a seguir.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]
    public int Id { get; set; }

    public string Title { get; set; }
    public int ISBN { get; set; }

    [DynamoDBProperty("Authors")]
```

```
    public List<string> BookAuthors { get; set; }

    [DynamoDBIgnore]
    public string CoverPage { get; set; }
  }
```

No exemplo anterior, o atributo `DynamoDBTable` mapeia a classe `Book` na tabela `ProductCatalog`.

O modelo de persistência objeto oferece suporte a tipos de mapeamento explícito e padrão entre propriedades de classe e atributos de tabela.

- Mapeamento explícito: para mapear uma propriedade em uma chave primária, você deve usar os atributos `DynamoDBHashKey` e `DynamoDBRangeKey` do modelo de persistência de objetos. Além disso, para os atributos de chave não primárias, se um nome de propriedade na sua classe e o atributo de tabela correspondente ao qual você deseja mapear não forem os mesmos, será necessário definir o mapeamento adicionando explicitamente o atributo `DynamoDBProperty`.

  No exemplo anterior, a propriedade `Id` é mapeada na chave primária com o mesmo nome, e a propriedade `BookAuthors` é mapeada no atributo `Authors` na tabela `ProductCatalog`.
- Mapeamento padrão — por padrão, o modelo de persistência objeto mapeia as propriedades da classe para os atributos com o mesmo nome na tabela.

  No exemplo anterior, as propriedades `Title` e `ISBN` são mapeadas nos atributos com o mesmo nome na tabela `ProductCatalog`.

Não é necessário mapear cada propriedade de classe. Identifique essas propriedades adicionando o atributo `DynamoDBIgnore`. Quando você salva uma instância de `Book` na tabela, `DynamoDBContext` não inclui a propriedade `CoverPage`. Ele também não retorna essa propriedade quando você recupera a instância de livro.

É possível mapear propriedades de tipos primitivos do .NET, como int e string. Também é possível mapear qualquer tipo de dados arbitrário, desde que você forneça um conversor apropriado para mapear os dados arbitrários em um do tipos do DynamoDB. Para saber mais sobre o mapeamento de tipos arbitrários, consulte Mapear dados arbitrários com o DynamoDB usando o modelo de persistência de objetos do AWS SDK para .NET.

O modelo de persistência objetos oferece suporte a bloqueio otimista. Durante uma operação de atualização, isso garante que você tenha a cópia mais recente do item que está prestes a

atualizar. Para ter mais informações, consulte Bloqueio positivo usando o DynamoDB e o modelo de persistência de objetos do AWS SDK para .NET.

Para obter mais informações, consulte os tópicos abaixo:

Tópicos

- Tipos de dados compatíveis
- Atributos do DynamoDB do modelo de persistência de objetos do .NET
- Classe DynamoDBContext do modelo de persistência de objetos do .NET
- Bloqueio positivo usando o DynamoDB e o modelo de persistência de objetos do AWS SDK para .NET
- Mapear dados arbitrários com o DynamoDB usando o modelo de persistência de objetos do AWS SDK para .NET

Tipos de dados compatíveis

O modelo de persistência de objetos oferece suporte a um conjunto de tipos de dados primitivos do .NET, coleções e tipos de dados arbitrários. O modelo é compatível com os seguintes tipos de dados primitivos.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`

- `UInt64`

O modelo de persistência de objetos também oferece suporte aos tipos de coleção .NET. `DynamoDBContext` é capaz de converter tipos de coleção concretos e objetos CLR básicos (POCOs).

A tabela a seguir resume o mapeamento dos tipos .NET anteriores nos tipos do DynamoDB.

Tipo primitivo .NET	Tipo do DynamoDB
Todos os tipos de número	`N` (tipo Número)
Todos os tipos de string	`S` (tipo String)
MemoryStream, byte[]	`B` (tipo Binário)
bool	`N` (tipo numérico), 0 representa false 1 representa true.
Tipos de coleção	Tipo `BS` (conjunto binário), tipo `SS` (conjunto de strings) e tipo `NS` (conjunto de números)
DateTime	`S` (tipo String). Os valores de `DateTime` são armazenados como strings formatadas em ISO-8601.

O modelo de persistência de objetos também oferece suporte a tipos de dados arbitrários. No entanto, você deve fornecer o código de conversor para mapear os tipos complexos em tipos do DynamoDB.

> **Note**
>
> - Valores binários vazios são compatíveis.
> - A leitura de valores string vazios é compatível. Os valores de atributos string vazios são compatíveis nos valores de atributos do tipo de conjunto string durante a gravação no DynamoDB. Os valores de atributos string vazios do tipo string e os valores string vazios contidos no tipo Lista ou Mapa são descartados das solicitações de gravação

Atributos do DynamoDB do modelo de persistência de objetos do .NET

Esta seção descreve os atributos oferecidos pelo modelo de persistência objetos para que você possa mapear suas classes e propriedades em tabelas e atributos do DynamoDB.

> **Note**
>
> Nos seguintes atributos, apenas `DynamoDBTable` e `DynamoDBHashKey` são necessários.

DynamoDBGlobalSecondaryIndexHashKey

Mapeia uma propriedade de classe na chave de partição de um índice secundário global. Use esse atributo se você precisa realizar uma `Query` em um índice secundário global.

DynamoDBGlobalSecondaryIndexRangeKey

Mapeia uma propriedade de classe na chave de classificação de um índice secundário global. Use esse atributo se você precisa realizar uma operação `Query` em um índice secundário global e deseja refinar seus resultados usando a chave de classificação de índice.

DynamoDBHashKey

Mapeia uma propriedade de classe para a chave de partição da chave primária da tabela. Os atributos de chave primária não podem ser um tipo de coleção.

Os seguintes exemplos de código C# mapeiam a classe `Book` na tabela `ProductCatalog` e a propriedade `Id` na chave de partição da chave primária da tabela.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

DynamoDBIgnore

Indica que a propriedade associada deve ser ignorada. Se não desejar salvar nenhuma das suas propriedades de classe, você poderá adicionar esse atributo para instruir `DynamoDBContext` a não incluir essa propriedade ao salvar objetos na tabela.

DynamoDBLocalSecondaryIndexRangeKey

Mapeia uma propriedade de classe na chave de classificação de um índice secundário local. Use esse atributo se você precisa realizar uma operação `Query` em um índice secundário local e deseja refinar seus resultados usando a chave de classificação de índice.

DynamoDBProperty

Mapeia uma propriedade de classe em um atributo de tabela. Se a propriedade de classe for mapeada em um atributo de tabela com o mesmo nome, não será necessário especificar esse atributo. No entanto, se os nomes não forem iguais, você poderá usar essa tag para fornecer o mapeamento. Na instrução C# a seguir, `DynamoDBProperty` mapeia a propriedade `BookAuthors` no atributo `Authors` na tabela.

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

`DynamoDBContext` usa essas informações de mapeamento para criar o atributo `Authors` ao salvar dados de objetos na tabela correspondente.

DynamoDBRenamable

Especifica um nome alternativo para uma propriedade de classe. Isso é útil quando você está escrevendo um conversor personalizado para o mapeamento de dados arbitrários em uma tabela do DynamoDB na qual o nome de uma propriedade de classe é diferente de um atributo da tabela.

DynamoDBRangeKey

Mapeia uma propriedade de classe na chave de classificação da chave primária da tabela. Se a tabela tiver uma chave primária composta (chave de partição e chave de classificação), você deverá especificar os atributos `DynamoDBHashKey` e `DynamoDBRangeKey` no seu mapeamento de classes.

Por exemplo, a tabela de exemplo `Reply` tem uma chave primária composta pela chave de partição `Id` e pela chave de classificação `Replenishment`. O exemplo de código C# a seguir mapeia a classe `Reply` na tabela `Reply`. A definição de classe também indica que duas de suas propriedades são mapeadas para a chave primária.

```
[DynamoDBTable("Reply")]
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

DynamoDBTable

Identifica a tabela de destino do DynamoDB na qual a classe é mapeada. Por exemplo, o exemplo de código C# a seguir mapeia a classe `Developer` na tabela `People` no DynamoDB.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

Esse atributo pode ser herdado ou substituído.

- O atributo `DynamoDBTable` pode ser herdado. No exemplo anterior, se você adicionar uma nova classe, `Lead`, que herda da classe `Developer`, ela também será mapeada na tabela `People`. Ambos os objetos `Developer` e `Lead` são armazenados na tabela `People`.
- O atributo `DynamoDBTable` também pode ser sobrescrito. No exemplo de código C# a seguir, a classe `Manager` herda da classe `Developer`. No entanto, a adição explícita do atributo `DynamoDBTable` mapeia a classe em outra tabela (`Managers`).

  ```
  [DynamoDBTable("Managers")]
  public class Manager : Developer { ...}
  ```

É possível adicionar o parâmetro opcional, `LowerCamelCaseProperties`, para solicitar que o DynamoDB coloque em minúscula a primeira letra do nome da propriedade ao armazenar os objetos em uma tabela, conforme mostrado no exemplo de código C# a seguir.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
```

```
    ...
}
```

Ao salvar as instâncias da classe `Developer`, `DynamoDBContext` salva a propriedade `DeveloperName` como `developerName`.

DynamoDBVersion

Identifica uma propriedade de classe para armazenar o número de versão do item. Para obter mais informações sobre versionamento, consulte Bloqueio positivo usando o DynamoDB e o modelo de persistência de objetos do AWS SDK para .NET.

Classe DynamoDBContext do modelo de persistência de objetos do .NET

A classe `DynamoDBContext` é o ponto de entrada para do banco de dados do Amazon DynamoDB. Ela fornece uma conexão com o DynamoDB e permite que você acesse seus dados em várias tabelas, realize várias operações CRUD e execute consultas. A classe `DynamoDBContext` fornece os seguintes métodos:

Tópicos

- CreateMultiTableBatchGet
- CreateMultiTableBatchWrite
- CreateBatchGet
- CreateBatchWrite
- Excluir
- Descartar
- ExecuteBatchGet
- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Carregar
- Consulta
- Save (Salvar)
- Verificar

- ToDocument
- Especificar parâmetros opcionais para DynamoDBContext

CreateMultiTableBatchGet

Cria um objeto `MultiTableBatchGet` formado por vários objetos `BatchGet` individuais. Cada um desses objetos `BatchGet` pode ser usado para recuperar itens de uma única tabela do DynamoDB.

Para recuperar os itens das tabelas, use o método `ExecuteBatchGet`, passando o objeto `MultiTableBatchGet` como um parâmetro.

CreateMultiTableBatchWrite

Cria um objeto `MultiTableBatchWrite` formado por vários objetos `BatchWrite` individuais. Cada um desses objetos `BatchWrite` pode ser usado para gravar ou excluir itens em uma única tabela do DynamoDB.

Para gravar nas tabelas, use o método `ExecuteBatchWrite`, passando o objeto `MultiTableBatchWrite` como um parâmetro.

CreateBatchGet

Cria um objeto `BatchGet` que você pode usar para recuperar vários itens de uma tabela.

CreateBatchWrite

Cria um objeto `BatchWrite` que você pode usar para inserir vários itens em uma tabela ou para excluir vários itens de uma tabela.

Excluir

Exclui um item da tabela. O método requer a chave primária do item que você deseja excluir. É possível fornecer o valor da chave primária ou um objeto no lado do cliente que contém um valor de chave primária como um parâmetro para esse método.

- Se você especificar um objeto no lado do cliente como um parâmetro e tiver habilitado o bloqueio otimista, a exclusão apenas será bem-sucedida se as versões no lado do cliente e no lado do servidor desse objeto corresponderem.
- Se você especificar somente o valor da chave primária como parâmetro, a exclusão será bem-sucedida, independentemente de você ter habilitado ou não o bloqueio otimista.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `DeleteAsync` em vez disso.

Descartar

Descarta todos os recursos gerenciados e não gerenciados.

ExecuteBatchGet

Lê dados de uma ou mais tabelas, processando todos os objetos `BatchGet` em um `MultiTableBatchGet`.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `ExecuteBatchGetAsync` em vez disso.

ExecuteBatchWrite

Grava ou exclui dados em uma ou mais tabelas, processando todos os objetos `BatchWrite` em um `MultiTableBatchWrite`.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `ExecuteBatchWriteAsync` em vez disso.

FromDocument

Considerando uma instância de `Document`, o método `FromDocument` retorna uma instância de uma classe no lado do cliente.

Isso será útil se você quiser usar as classes de modelo de documento junto com o modelo de persistência de objetos para realizar qualquer operação de dados. Para obter mais informações sobre as classes do modelo de documento fornecidas pelo AWS SDK para .NET, consulte Trabalhar com o modelo de documento do .NET no DynamoDB.

Suponha que você tenha um objeto `Document` denominado `doc` que contém uma representação de um item `Forum`. (Para ver como construir esse objeto, consulte a descrição do método `ToDocument` mais adiante neste tópico.) Você pode usar `FromDocument` para recuperar o item `Forum` de `Document`, conforme mostrado no exemplo de código C# a seguir.

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> Se o objeto `Document` implementar a interface `IEnumerable`, você poderá usar o método `FromDocuments` em vez disso. Isso permite uma iteração sobre todas as instâncias da classe em `Document`.

FromQuery

Executa uma operação `Query`, com os parâmetros de consulta definidos em um objeto `QueryOperationConfig`.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `FromQueryAsync` em vez disso.

FromScan

Executa uma operação `Scan`, com os parâmetros de verificação definidos em um objeto `ScanOperationConfig`.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `FromScanAsync` em vez disso.

GetTargetTable

Recupera a tabela de destino para o tipo especificado. Isso é útil quando você está escrevendo um conversor personalizado para o mapeamento de dados arbitrários para uma tabela do DynamoDB e precisa determinar qual tabela está associada a um tipo de dados personalizado.

Carregar

Recupera um item de uma tabela. O método requer somente a chave primária do item que você deseja recuperar.

Por padrão, o DynamoDB retorna o item com valores finais consistentes. Para obter informações sobre o modelo final consistente, consulte Consistência de leitura do DynamoDB.

O método `Load` ou `LoadAsync` chama a operação GetItem, que exige que você especifique a chave primária para a tabela. Como `GetItem` ignora o parâmetro `IndexName`, você não pode carregar um item usando a partição ou a chave de classificação de um índice. Portanto, é necessário usar a chave primária da tabela para carregar um item.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `LoadAsync` em vez disso. Para ver um exemplo do uso do método `LoadAsync` para realizar operações CRUD de alto nível em uma tabela do DynamoDB, consulte o exemplo a seguir.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
```

```
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
```

```
    }
```

Consulta

Consulta uma tabela com base em parâmetros de consulta que você fornece.

Você poderá consultar uma tabela somente se ela tiver uma chave primária composta (chave de partição e chave de classificação). Ao consultar, você deve especificar uma chave de partição e uma condição que se aplique à chave de classificação.

Suponha que você tenha uma classe `Reply` no lado do cliente mapeada na tabela `Reply` no DynamoDB. O exemplo de código C# a seguir consulta a tabela `Reply` para encontrar respostas de tópicos de fórum postadas nos últimos 15 dias. A tabela `Reply` tem uma chave primária com a chave de partição `Id` e a chave de classificação `ReplyDateTime`.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

string replyId = "DynamoDB#DynamoDB Thread 1"; //Partition key
DateTime twoWeeksAgoDate = DateTime.UtcNow.Subtract(new TimeSpan(14, 0, 0, 0)); // Date
 to compare.
IEnumerable<Reply> latestReplies = context.Query<Reply>(replyId,
 QueryOperator.GreaterThan, twoWeeksAgoDate);
```

Isso retorna uma coleção de objetos `Reply`.

Por padrão, o método `Query` retorna uma coleção `IEnumerable` de "carregamento preguiçoso". Ele inicialmente retorna apenas uma página de resultados e, em seguida, faz uma chamada de serviço para a próxima página, se necessário. Para obter todos os itens correspondentes, você só precisa fazer uma iteração na coleção `IEnumerable`.

Se a sua tabela tiver uma chave primária simples (chave de partição), você não poderá usar o método `Query`. Em vez disso, poderá usar o método `Load` e fornecer a chave de partição para recuperar o item.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `QueryAsync` em vez disso.

Save (Salvar)

Salva o objeto especificado na tabela. Se a chave primária especificada no objeto de entrada não existir na tabela, o método adicionará um novo item à tabela. Se a chave primária existir, o método atualizará o item existente.

Se você tiver o bloqueio otimista configurado, a atualização será bem-sucedida apenas se as versões do item no lado do cliente e no lado do servidor corresponderem. Para ter mais informações, consulte Bloqueio positivo usando o DynamoDB e o modelo de persistência de objetos do AWS SDK para .NET.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `SaveAsync` em vez disso.

Verificar

Realiza uma verificação de tabela inteira.

Você pode filtrar o resultado da verificação especificando uma condição de verificação. A condição pode ser avaliada em qualquer atributo da tabela. Suponha que você tenha uma classe `Book` no lado do cliente mapeada na tabela `ProductCatalog` no DynamoDB. O exemplo de código C# a seguir verifica a tabela e retorna apenas os itens de livro com preços inferiores a 0.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

Por padrão, o método `Scan` retorna uma coleção `IEnumerable` de "carregamento preguiçoso". Ele inicialmente retorna apenas uma página de resultados e, em seguida, faz uma chamada de serviço para a próxima página, se necessário. Para obter todos os itens correspondentes, basta fazer uma iteração na coleção `IEnumerable`.

Por motivos de performance, você deve consultar suas tabelas e evitar uma verificação de tabela.

> **Note**
>
> Para realizar essa operação em segundo plano, use o método `ScanAsync` em vez disso.

ToDocument

Retorna uma instância da classe de modelo de documento `Document` da sua instância de classe.

Isso será útil se você quiser usar as classes de modelo de documento junto com o modelo de persistência de objetos para realizar qualquer operação de dados. Para obter mais informações sobre as classes do modelo de documento fornecidas pelo AWS SDK para .NET, consulte Trabalhar com o modelo de documento do .NET no DynamoDB.

Suponha que você tenha uma classe de cliente mapeada na tabela de exemplo `Forum`. É possível usar um `DynamoDBContext` para obter um item, como um objeto `Document` da tabela `Forum`, conforme mostrado no exemplo de código C# a seguir.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

Especificar parâmetros opcionais para DynamoDBContext

Ao usar o modelo de persistência de objeto, você pode especificar os seguintes parâmetros opcionais para `DynamoDBContext`.

- **`ConsistentRead`**: ao recuperar dados usando as operações `Load`, `Query` ou `Scan`, você pode opcionalmente adicionar esse parâmetro para solicitar os valores mais recentes dos dados.
- **`IgnoreNullValues`**: esse parâmetro instrui `DynamoDBContext` a ignorar valores nulos em atributos durante uma operação `Save`. Se esse parâmetro for false (ou se não estiver definido), um valor nulo será interpretado como uma diretiva para excluir o atributo específico.
- **`SkipVersionCheck`**: esse parâmetro instrui o `DynamoDBContext` a não comparar versões ao salvar ou excluir um item. Para obter mais informações sobre versionamento, consulte Bloqueio positivo usando o DynamoDB e o modelo de persistência de objetos do AWS SDK para .NET.

- **`TableNamePrefix`**: prefixa todos os nomes de tabelas com uma string específica. Se esse parâmetro for nulo (ou se não estiver definido), nenhum prefixo será usado.
- `DynamoDBEntryConversion`: especifica o esquema de conversão usado pelo cliente. Você pode definir esse parâmetro para a versão V1 ou V2. A versão padrão é V1.

  O comportamento desse parâmetro pode mudar com base na versão definida. Por exemplo:

  - Na V1, o tipo de dados `bool` é convertido no tipo numérico `N`, em que 0 representa falso e 1 representa verdadeiro. Na V2, `bool` é convertido em `BOOL`.
  - Na V2, listas e matrizes não são agrupadas com HashSets. Listas e matrizes de números, tipos baseados em strings e tipos baseados em binários são convertidos no tipo `L` (Lista), que pode ser enviado vazio para atualizar uma lista. Isso é diferente da V1, em que uma lista vazia não é enviada pela rede.

    Na V1, os tipos de coleção, como lista, HashSet e matrizes, são tratados da mesma forma. Lista, HashSet e matriz de números são convertidos no tipo `NS` (conjunto de números).

  O exemplo a seguir define a versão do esquema de conversão como V2, o que altera o comportamento de conversão entre os tipos .NET e os tipos de dados do DynamoDB.

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

O exemplo de código C# a seguir cria um `DynamoDBContext` especificando dois dos parâmetros opcionais anteriores, `ConsistentRead` e `SkipVersionCheck`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
       new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

`DynamoDBContext` inclui esses parâmetros opcionais com cada solicitação enviada usando esse contexto.

Em vez de definir esses parâmetros no nível de `DynamoDBContext`, é possível especificá-los para operações individuais que você executa usando `DynamoDBContext`, conforme mostrado no exemplo de código C# a seguir. O exemplo carrega um item de livro específico. O método `Load` de `DynamoDBContext` especifica os parâmetros `ConsistentRead` e `SkipVersionCheck` opcionais.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

Nesse caso, `DynamoDBContext` inclui esses parâmetros somente ao enviar a solicitação `Get`.

Bloqueio positivo usando o DynamoDB e o modelo de persistência de objetos do AWS SDK para .NET

O suporte para bloqueio otimista no modelo de persistência de objetos garante que a versão do item para a sua aplicação seja igual à versão do item no lado do servidor antes que esse item seja atualizado ou excluído. Suponha que você recupere um item para atualização. No entanto, antes de você retornar suas atualizações, outra aplicação atualiza o mesmo item. Agora, a aplicação tem uma cópia obsoleta do item. Sem o bloqueio otimista, qualquer atualização que você realizar substituirá a atualização feita pelo outro aplicativo.

O recurso de bloqueio otimista do modelo de persistência de objetos fornece a tag `DynamoDBVersion` que você pode usar para habilitar o bloqueio otimista. Para usar esse recurso, adicione uma propriedade à sua classe para armazenar o número de versão. Você adiciona o atributo `DynamoDBVersion` à propriedade. Quando o objeto for salvo pela primeira vez, `DynamoDBContext` atribuirá um número de versão e incrementará esse valor cada vez que você atualizar o item.

Sua solicitação de atualização ou exclusão só será bem-sucedida se a versão do objeto no lado do cliente corresponder ao número de versão correspondente do item no lado do servidor. Se a sua aplicação tiver uma cópia obsoleta, ela deverá obter a versão mais recente do servidor antes de poder atualizar ou excluir o item.

O exemplo de código C# a seguir define uma classe `Book` com atributos de persistência de objetos mapeando-a na tabela `ProductCatalog`. A propriedade `VersionNumber` na classe decorada com o atributo `DynamoDBVersion` armazena o valor do número de versão.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]   //Partition key
    public int Id { get; set; }
    [DynamoDBProperty]
    public string Title { get; set; }
    [DynamoDBProperty]
    public string ISBN { get; set; }
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }
    [DynamoDBVersion]
    public int? VersionNumber { get; set; }
  }
```

> **Note**
>
> Você pode aplicar o atributo `DynamoDBVersion` apenas a um tipo primitivo numérico anulável (como `int?`).

O bloqueio otimista tem o seguinte impacto sobre operações `DynamoDBContext`:

- Para um novo item, `DynamoDBContext` atribui o número de versão inicial 0. Se você recuperar um item existente, atualizar uma ou mais das suas propriedades e tentar salvar as alterações, a operação de salvamento será bem-sucedida somente se o número de versão no lado do cliente e no lado do servidor corresponderem. `DynamoDBContext` incrementa o número de versão. Você não precisa definir o número de versão.
- O método `Delete` fornece sobrecargas que podem usar um valor de chave primária ou um objeto como parâmetro, conforme mostrado no exemplo de código C# a seguir.

  Example

  ```
  DynamoDBContext context = new DynamoDBContext(client);
  ...
  // Load a book.
  Book book = context.Load<ProductCatalog>(111);
  // Do other operations.
  ```

```
// Delete 1 - Pass in the book object.
context.Delete<ProductCatalog>(book);

// Delete 2 - Pass in the Id (primary key)
context.Delete<ProductCatalog>(222);
```

Se você fornecer um objeto como parâmetro, a exclusão apenas será bem-sucedida se a versão do objeto corresponder à versão de item no lado do servidor correspondente. No entanto, se você fornecer um valor de chave primária como parâmetro, `DynamoDBContext` desconhecerá qualquer número de versão e excluirá o item sem fazer a verificação de versão.

Observe que a implementação interna do bloqueio otimista no código do modelo de persistência de objetos usa as ações de API de atualização condicional e exclusão condicional no DynamoDB.

Desabilitar o bloqueio positivo

Para desabilitar o bloqueio otimista, use a propriedade de configuração `SkipVersionCheck`. Você pode definir essa propriedade ao criar `DynamoDBContext`. Nesse caso, o bloqueio otimista está desabilitado para solicitações feitas usando o contexto. Para ter mais informações, consulte Especificar parâmetros opcionais para DynamoDBContext .

Em vez de definir a propriedade no nível do contexto, você pode desabilitar o bloqueio otimista para uma operação específica, conforme mostrado no exemplo de código C# a seguir. O exemplo de código usa o contexto para excluir um item de livro. O método `Delete` define a propriedade `SkipVersionCheck` opcional como true, desabilitando a verificação de versão.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Load a book.
Book book = context.Load<ProductCatalog>(111);
...
// Delete the book.
context.Delete<Book>(book, new DynamoDBContextConfig { SkipVersionCheck = true });
```

Mapear dados arbitrários com o DynamoDB usando o modelo de persistência de objetos do AWS SDK para .NET

Além dos tipos de .NET compatíveis (consulte Tipos de dados compatíveis), é possível usar tipos em sua aplicação para os quais não há um mapeamento direto para os tipos do Amazon DynamoDB. O

modelo de persistência de objetos oferece suporte ao armazenamento de dados de tipos arbitrários, desde que você forneça o conversor para converter dados do tipo arbitrário no tipo do DynamoDB e vice-versa. O código de conversor transforma os dados durante os processos de salvar e carregar os objetos.

É possível criar qualquer tipo no lado do cliente. No entanto, os dados armazenados nas tabelas são um dos tipos do DynamoDB e, durante a consulta e a verificação, qualquer comparação de dados feita baseia-se nos dados armazenados no DynamoDB.

O exemplo de código C# a seguir define uma classe `Book` com as propriedades `Id`, `Title`, `ISBN` e `Dimension`. A propriedade `Dimension` é do `DimensionType`, que descreve as propriedades `Height`, `Width` e `Thickness`. O código de exemplo fornece os métodos de conversor `ToEntry` e `FromEntry` para converter dados entre o `DimensionType` e os tipos de string do DynamoDB. Por exemplo, ao salvar uma instância `Book`, o conversor cria uma string `Dimension` de livro como “8.5x11x.05". Quando você recupera um livro, ele converte a string em uma instância `DimensionType`.

O exemplo mapeia o tipo `Book` na tabela `ProductCatalog`. Ele salva uma instância `Book` de exemplo, recupera essa instância, atualiza suas dimensões e salva novamente o `Book` atualizado.

Para obter instruções passo a passo sobre como testar o exemplo a seguir, consulte Exemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelMappingArbitraryData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
```

```
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);

                // 1. Create a book.
                DimensionType myBookDimensions = new DimensionType()
                {
                    Length = 8M,
                    Height = 11M,
                    Thickness = 0.5M
                };

                Book myBook = new Book
                {
                    Id = 501,
                    Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                    ISBN = "999-9999999999",
                    BookAuthors = new List<string> { "Author 1", "Author 2" },
                    Dimensions = myBookDimensions
                };

                context.Save(myBook);

                // 2. Retrieve the book.
                Book bookRetrieved = context.Load<Book>(501);

                // 3. Update property (book dimensions).
                bookRetrieved.Dimensions.Height += 1;
                bookRetrieved.Dimensions.Length += 1;
                bookRetrieved.Dimensions.Thickness += 0.2M;
                // Update the book.
                context.Save(bookRetrieved);

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }
    }
    [DynamoDBTable("ProductCatalog")]
    public class Book
```

```
    {
        [DynamoDBHashKey] //Partition key
        public int Id
        {
            get; set;
        }
        [DynamoDBProperty]
        public string Title
        {
            get; set;
        }
        [DynamoDBProperty]
        public string ISBN
        {
            get; set;
        }
        // Multi-valued (set type) attribute.
        [DynamoDBProperty("Authors")]
        public List<string> BookAuthors
        {
            get; set;
        }
        // Arbitrary type, with a converter to map it to DynamoDB type.
        [DynamoDBProperty(typeof(DimensionTypeConverter))]
        public DimensionType Dimensions
        {
            get; set;
        }
    }

    public class DimensionType
    {
        public decimal Length
        {
            get; set;
        }
        public decimal Height
        {
            get; set;
        }
        public decimal Thickness
        {
            get; set;
        }
```

```
    }

    // Converts the complex type DimensionType to string and vice-versa.
    public class DimensionTypeConverter : IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            DimensionType bookDimensions = value as DimensionType;
            if (bookDimensions == null) throw new ArgumentOutOfRangeException();

            string data = string.Format("{1}{0}{2}{0}{3}", " x ",
                              bookDimensions.Length, bookDimensions.Height,
 bookDimensions.Thickness);

            DynamoDBEntry entry = new Primitive
            {
                Value = data
            };
            return entry;
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            Primitive primitive = entry as Primitive;
            if (primitive == null || !(primitive.Value is String) ||
 string.IsNullOrEmpty((string)primitive.Value))
                throw new ArgumentOutOfRangeException();

            string[] data = ((string)(primitive.Value)).Split(new string[] { " x " },
 StringSplitOptions.None);
            if (data.Length != 3) throw new ArgumentOutOfRangeException();

            DimensionType complexData = new DimensionType
            {
                Length = Convert.ToDecimal(data[0]),
                Height = Convert.ToDecimal(data[1]),
                Thickness = Convert.ToDecimal(data[2])
            };
            return complexData;
        }
    }
}
```

Executar os exemplos de código neste Guia do desenvolvedor

Os AWS SDKs fornecem um amplo suporte ao Amazon DynamoDB nas seguintes linguagens:

- Java
- JavaScript no navegador
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

Os exemplos de código neste guia do desenvolvedor fornecem uma cobertura mais detalhada das operações do DynamoDB usando as seguintes linguagens de programação:

- Exemplos de código Java
- Exemplos de código .NET

Antes de iniciar este exercício, será necessário criar uma conta da AWS, obter a chave de acesso e a chave secreta e configurar a AWS Command Line Interface (AWS CLI) no seu computador. Para obter mais informações, consulte Configurar o DynamoDB (serviço da Web) .

> **Note**
>
> Se estiver usando a versão para download do DynamoDB, você precisará usar a AWS CLI para criar as tabelas e os dados de exemplo. Também precisará especificar o parâmetro `--endpoint-url` com cada comando da AWS CLI. Para obter mais informações, consulte Definir o endpoint local .

Criar tabelas e carregar dados para exemplos de código no DynamoDB

Veja abaixo as noções básicas sobre como criar tabelas no DynamoDB, carregar um conjunto de dados de exemplo, consultar os dados e atualizá-los.

- Etapa 1: crie uma tabela no DynamoDB
- Etapa 2: grave dados em uma tabela do DynamoDB
- Etapa 3: leia os dados de uma tabela do DynamoDB
- Etapa 4: atualize os dados em uma tabela do DynamoDB

Exemplos de código Java

Tópicos

- Java: configurar suas credenciais da AWS
- Java: configurar a região e o endpoint da AWS

Este Guia do Desenvolvedor contém trechos de código Java e programas prontos para serem executados. Você encontrará esses exemplos de código nas seguintes seções:

- Trabalhar com itens e atributos no DynamoDB
- Trabalhar com tabelas e dados no DynamoDB
- Consultar tabelas no DynamoDB
- Verificar tabelas no DynamoDB
- Melhorar o acesso aos dados com índices secundários no DynamoDB
- Java 1.x: DynamoDBMapper
- Capturar dados de alterações para o DynamoDB Streams

É possível começar rapidamente usando o Eclipse com o AWS Toolkit for Eclipse. Além de um IDE completo, você também terá o AWS SDK para Java com atualizações automáticas e modelos pré-configurados para a compilação de aplicações da AWS.

Como executar os exemplos de código Java (usando o Eclipse)

1. Baixe e instale o IDE do Eclipse.
2. Faça download e instale o AWS Toolkit for Eclipse.

3. Inicie o Eclipse e, no menu Eclipse, escolha File (Arquivo), New (Novo) e Other (Outro).
4. Em Selecionar um assistente, escolha AWS, Projeto AWS Java e Próximo.
5. Em Criar um AWS Java, faça o seguinte:

   a. Em Nome do projeto, digite um nome para o seu projeto.

   b. Em Selecionar conta (Selecionar conta), escolha o perfil de suas credenciais na lista.

      Se essa for a primeira vez que você usa o AWS Toolkit for Eclipse, escolha Configurar contas da AWS para configurar suas credenciais da AWS.
6. Escolha Finish (Concluir) para criar o projeto.
7. No menu do Eclipse, escolha File (Arquivo), New (Novo) e, em seguida, Class (Classe).
8. Em Java Class (Classe Java), digite um nome para a sua classe em Name (Nome) (use o mesmo nome que o exemplo de código que você deseja executar) e escolha Finish (Concluir) para criar a classe.
9. Copie o exemplo de código da página de documentação no editor do Eclipse.
10. Para executar o código, escolha Run (Executar) no menu do Eclipse.

O SDK for Java fornece clientes thread-safe para trabalhar com o DynamoDB. De acordo com as melhores práticas, seus aplicativos devem criar um único cliente e reutilizá-lo entre os threads.

Para obter mais informações, consulte AWS SDK para Java.

> **Note**
>
> Os exemplos de código neste guia são destinados para uso com a versão mais recente do AWS SDK para Java.
> Se estiver usando o AWS Toolkit for Eclipse, você poderá configurar atualizações automáticas para o SDK for Java. Para fazer isso no Eclipse, vá para Preferences (Preferências) e escolha AWS Toolkit, AWS SDK para Java e Download new SDKs automatically (Fazer download automático de novos SDKs).

Java: configurar suas credenciais da AWS

O SDK for Java exige que você forneça credenciais da AWS para a sua aplicação em tempo de execução. Os exemplos de código neste guia pressupõem que você esteja usando um arquivo

de credenciais da AWS, conforme descrito em Configurar suas credenciais da AWS no Guia do desenvolvedor da AWS SDK para Java.

Veja a seguir um exemplo de um arquivo de credenciais da AWS chamado `~/.aws/credentials` em que o caractere de til (~) representa seu diretório inicial.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: configurar a região e o endpoint da AWS

Por padrão, os exemplos de código acessam o DynamoDB na região Oeste dos EUA (Oregon). É possível alterar a região ao modificar as propriedades `AmazonDynamoDB`.

O exemplo de código a seguir instancia uma nova `AmazonDynamoDB`.

```
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

É possível usar o método `withRegion` para executar seu código no DynamoDB em qualquer região em que ele está disponível. Para obter uma lista completa, consulte Regiões e endpoints da AWS, na Referência geral da Amazon Web Services.

Se desejar executar os exemplos de código usando o DynamoDB localmente no seu computador, defina o endpoint conforme descrito a seguir.

AWS SDK V1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

AWS SDK V2

```
DynamoDbClient client = DynamoDbClient.builder()
```

```
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
validation
    .region(Region.US_EAST_1)
    .credentialsProvider(StaticCredentialsProvider.create(
    AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

Exemplos de código .NET

Tópicos

- .NET: configurar as suas credenciais da AWS
- .NET: configurar a região e o endpoint da AWS

Este guia contém trechos de código .NET e programas prontos para serem executados. Você encontrará esses exemplos de código nas seguintes seções:

- Trabalhar com itens e atributos no DynamoDB
- Trabalhar com tabelas e dados no DynamoDB
- Consultar tabelas no DynamoDB
- Verificar tabelas no DynamoDB
- Melhorar o acesso aos dados com índices secundários no DynamoDB
- Trabalhar com o modelo de documento do .NET no DynamoDB
- Trabalhar com o modelo de persistência de objetos do .NET e o DynamoDB
- Capturar dados de alterações para o DynamoDB Streams

Você pode começar rapidamente usando o AWS SDK para .NET com o Toolkit for Visual Studio.

Como executar os exemplos de código .NET (usando o Visual Studio)

1. Baixe e instale o Microsoft Visual Studio.
2. Baixe e instale o Toolkit for Visual Studio.
3. Inicie o Visual Studio. Escolha File (Arquivo), New (Novo), Project (Projeto).
4. Em New Project (Novo projeto), escolha AWS Empty Project (Projeto vazio da) e depois OK.
5. Em Credenciais de acesso da AWS, escolha Usar perfil existente, escolha seu perfil de credenciais na lista e depois OK.

Se esta for a primeira vez que você usa o Toolkit for Visual Studio, escolha Use a new profile (Usar um novo perfil) para configurar suas credenciais da AWS.

6. No projeto do Visual Studio, escolha a guia para o código-fonte do seu programa (`Program.cs`). Copie o exemplo de código da página da documentação no editor do Visual Studio, substituindo qualquer outro código que você visualize no editor.
7. Se você receber mensagens de erro da forma The type or namespace name...could not be found (O tipo ou nome do namespace...não pôde ser encontrado), será necessário instalar o conjunto do AWS SDK para o DynamoDB da seguinte forma:

   a. No Solution Explorer, abra o menu de contexto (clique com o botão direito do mouse) do seu projeto e escolha Manage NuGet Packages (Gerenciar pacotes do NuGet).

   b. No Gerenciador de pacotes NuGet, escolha Browse (Procurar).

   c. Na caixa de pesquisa, digite **`AWSSDK.DynamoDBv2`** e aguarde a conclusão da pesquisa.

   d. Escolha AWSSDK.DynamoDBv2 e Install (Instalar).

   e. Quando a instalação estiver concluída, escolha a guia Program.cs para retornar ao seu programa.
8. Para executar o código, escolha Start (Iniciar) na barra de ferramentas do Visual Studio.

O SDK para .NET fornece clientes thread-safe para trabalhar com o DynamoDB. De acordo com as melhores práticas, seus aplicativos devem criar um único cliente e reutilizá-lo entre os threads.

Para obter mais informações, consulte AWS SDK for .NET.

> **Note**
>
> Os exemplos de código neste guia são destinados para uso com a versão mais recente do AWS SDK para .NET.

.NET: configurar as suas credenciais da AWS

O SDK para .NET exige que você forneça credenciais da AWS para a sua aplicação em tempo de execução. Os exemplos de código neste guia pressupõem que você esteja usando o SDK Store para gerenciar seu arquivo de credenciais da AWS, conforme descrito em Usar o SDK Store no Guia do desenvolvedor do AWS SDK para .NET.

O Toolkit for Visual Studio oferece suporte a vários conjuntos de credenciais de qualquer número de contas. Cada conjunto é chamado de perfil. O Visual Studio adiciona entradas ao arquivo `App.config` do projeto para que a sua aplicação possa encontrar as credenciais da AWS em tempo de execução.

O exemplo a seguir mostra o arquivo `App.config` padrão que é gerado ao criar um novo projeto usando o Toolkit for Visual Studio.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

Em tempo de execução, o programa usa o conjunto `default` de credenciais da AWS, conforme especificado pela entrada `AWSProfileName`. As credenciais da AWS são mantidas no SDK Store em formato criptografado. O Toolkit for Visual Studio fornece uma interface gráfica do usuário para gerenciar suas credenciais no Visual Studio. Para obter mais informações, consulte Specifying credentials (Especificar credenciais) no AWS Toolkit for Visual Studio User Guide (Guia do usuário do).

> **Note**
>
> Por padrão, os exemplos de código acessam o DynamoDB na região Oeste dos EUA (Oregon). É possível alterar a região ao modificar a entrada `AWSRegion` no arquivo App.config. É possível definir `AWSRegion` como qualquer região em que o DynamoDB está disponível. Para obter uma lista completa, consulte Regiões e endpoints da AWS, na Referência geral da Amazon Web Services.

.NET: configurar a região e o endpoint da AWS

Por padrão, os exemplos de código acessam o DynamoDB na região Oeste dos EUA (Oregon). É possível alterar a região ao modificar a entrada `AWSRegion` no arquivo `App.config`. É possível alterar a região ao modificar as propriedades `AmazonDynamoDBClient`.

O exemplo de código a seguir instancia uma nova `AmazonDynamoDBClient`. O cliente é modificado de forma que o código seja executado no DynamoDB em uma região diferente.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Para obter uma lista completa de regiões, consulte Regiões e endpoints da AWS na Referência geral da Amazon Web Services.

Se desejar executar os exemplos de código usando o DynamoDB localmente no seu computador, defina o endpoint conforme descrito a seguir.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

API de baixo nível do DynamoDB

A API de baixo nível do Amazon DynamoDB é a interface de nível de protocolo para DynamoDB. Nesse nível, cada solicitação de HTTP(S) deve ser corretamente formatada e ter uma assinatura digital válida.

Os AWS SDKs criam solicitações da API do DynamoDB de baixo nível em seu nome e processam as respostas no DynamoDB. Isso permite que você se concentre na lógica do seu aplicativo, em vez de detalhes de baixo nível. No entanto, você ainda pode se beneficiar de um conhecimento básico de como a API de baixo nível do DynamoDB funciona.

Para obter mais informações sobre a API de baixo nível do DynamoDB, consulte a Referência da API do Amazon DynamoDB.

> **Note**
>
> O DynamoDB Streams tem sua própria API de baixo nível, que é separada do DynamoDB e tem suporte total dos AWS SDKs.
> Para obter mais informações, consulte Capturar dados de alterações para o DynamoDB Streams. Para obter a API de baixo nível do DynamoDB Streams, consulte a Referência da API do Amazon DynamoDB Streams.

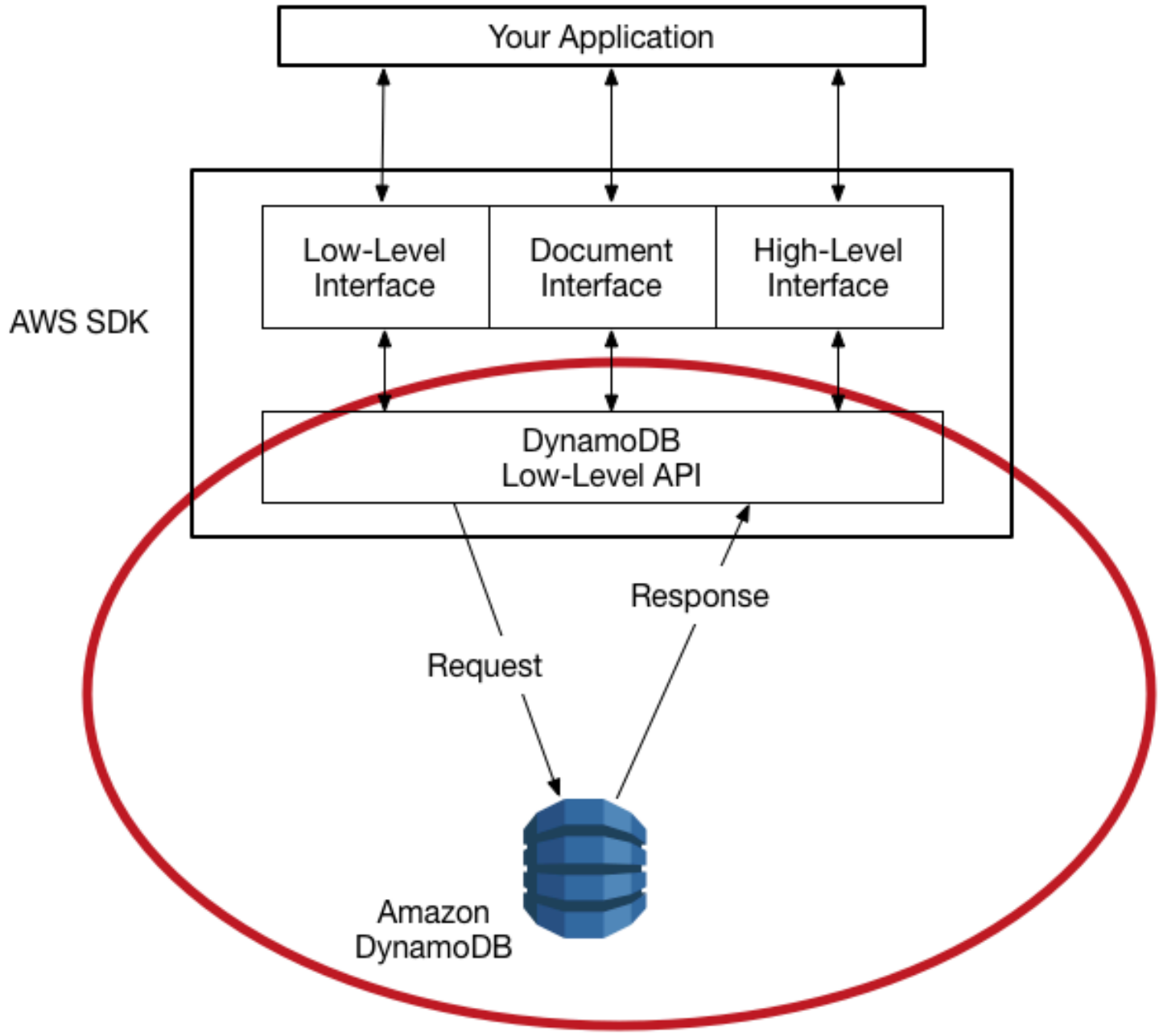
A API de baixo nível do DynamoDB usa a JavaScript Object Notation (JSON) como um formato de protocolo de conexão. O JSON apresenta dados em uma hierarquia de forma que os valores e a estrutura dos dados sejam transmitidos simultaneamente. O pares de nome-valor são definidos no formato `name:value`. A hierarquia de dados é definida por colchetes aninhados de pares de nome-valor.

O DynamoDB usa JSON somente como um protocolo de transporte, não como um formato de armazenamento. Os AWS SDKs usam JSON para enviar dados ao DynamoDB, e o DynamoDB responde com JSON. O DynamoDB não armazena dados persistentemente no formato JSON.

> **Note**
>
> Para obter mais informações sobre JSON Introdução ao JSON site `JSON.org`.

Tópicos

- Formato de solicitação
- Formato de resposta
- Descritores de tipo de dados
- Dados numéricos
- Dados binários



Formato de solicitação

A API de baixo nível do DynamoDB aceita solicitações `POST` HTTP(S) como entrada. Os AWS SDKs criam essas solicitações para você.

Vamos supor que você tenha uma tabela chamada `Pets`, com um esquema de chaves que consiste em `AnimalType` (chave de partição) e `Name` (chave de classificação). Ambos os atributos são do tipo `string`. Para recuperar um item de `Pets`, o AWS SDK cria a solicitação a seguir.

```
POST / HTTP/1.1
```

```
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Pets",
    "Key": {
        "AnimalType": {"S": "Dog"},
        "Name": {"S": "Fido"}
    }
}
```

Observe o seguinte sobre essa solicitação:

- O cabeçalho `Authorization` contém as informações necessárias para o DynamoDB autenticar a solicitação. Para obter mais informações, consulte Assinar solicitações de API da AWS e Processo de assinatura do Signature versão 4 na Referência geral da Amazon Web Services.
- O cabeçalho `X-Amz-Target` contém o nome de uma operação do DynamoDB: `GetItem`. (Isso também é acompanhado pela versão da API de baixo nível, neste caso `20120810`.)
- A carga útil (corpo) da solicitação contém os parâmetros da operação, no formato JSON. Para a operação `GetItem`, os parâmetros são `TableName` e `Key`.

Formato de resposta

Após o recebimento da solicitação, o DynamoDB a processa e retorna uma resposta. Para a solicitação mostrada anteriormente, a carga de resposta HTTP(S) contém os resultados da operação, conforme mostrado no exemplo a seguir.

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

```
{
    "Item": {
        "Age": {"N": "8"},
        "Colors": {
            "L": [
                {"S": "White"},
                {"S": "Brown"},
                {"S": "Black"}
            ]
        },
        "Name": {"S": "Fido"},
        "Vaccinations": {
            "M": {
                "Rabies": {
                    "L": [
                        {"S": "2009-03-17"},
                        {"S": "2011-09-21"},
                        {"S": "2014-07-08"}
                    ]
                },
                "Distemper": {"S": "2015-10-13"}
            }
        },
        "Breed": {"S": "Beagle"},
        "AnimalType": {"S": "Dog"}
    }
}
```

Neste momento, o AWS SDK retorna os dados da resposta para sua aplicação para processamento adicional.

> **Note**
>
> Se o DynamoDB não puder processar uma solicitação, ele retornará uma mensagem e um código de erro HTTP. O AWS SDK propaga esses elementos em sua aplicação na forma de exceções. Para obter mais informações, consulte Tratamento de erros com o DynamoDB.

Descritores de tipo de dados

O protocolo da API de baixo nível do DynamoDB exige que cada atributo seja acompanhado por um descritor de tipo de dados. Descritores de tipos de dados são tokens que informam ao DynamoDB como interpretar cada atributo.

Os exemplos em Formato de solicitação e Formato de resposta mostram exemplos de como os descritores de tipo de dados são usados. A solicitação `GetItem` especifica `S` para os atributos de esquema de chaves de `Pets` (`AnimalType` e `Name`), que são do tipo `string`. A resposta `GetItem` contém o item Pets com atributos do tipo `string` (`S`), `number` (`N`), `map` (`M`) e `list` (`L`).

Veja a seguir uma lista completa dos descritores de tipos de dados do DynamoDB:

- **`S`** – String
- **`N`** – Number
- **`B`** – Binary
- **`BOOL`** – Boolean
- **`NULL`** – Null
- **`M`** – Map
- **`L`** – List
- **`SS`** – String Set
- **`NS`** – Number Set
- **`BS`** – Binary Set

> **Note**
>
> Para obter descrições detalhadas dos tipos de dados do DynamoDB, consulte Tipos de dados.

Dados numéricos

As diferentes linguagens de programação oferecem diferentes níveis de suporte para JSON. Em alguns casos, é possível decidir usar uma biblioteca de terceiros para validar e analisar documentos JSON.

Algumas bibliotecas de terceiros se desenvolvem com base no tipo número do JSON, fornecendo seus próprios tipos, como `int`, `long` ou `double`. No entanto, o tipo de dados de número nativo no DynamoDB não é mapeado exatamente nesses outros tipos de dados, portanto, essas distinções de tipo podem causar conflitos. Além disso, muitas bibliotecas do JSON não manipulam valores numéricos de precisão fixa, e elas inferem automaticamente um tipo de dados duplo para sequências de dígitos que contêm um separador decimal.

Para solucionar esses problemas, o DynamoDB fornece um único tipo numérico sem perda de dados. Para evitar conversões implícitas indesejadas para um valor duplo, o DynamoDB usa strings para a transferência de dados de valores numéricos. Essa abordagem fornece flexibilidade para atualizar valores de atributo, sem deixar de manter a semântica de classificação adequada, como colocar os valores "01", "2" e "03" na sequência apropriada.

Se a precisão numérica for importante para sua aplicação, você deverá converter valores numéricos em strings antes de passá-los para o DynamoDB.

Dados binários

O DynamoDB oferece suporte a atributos binários. No entanto, o JSON não é originalmente compatível com a codificação de dados binários. Para enviar dados binários em uma solicitação, será necessário codificá-los em formato base64. Ao receber a solicitação, o DynamoDB decodifica os dados em base64 de volta para binário.

O esquema de codificação base64 usado pelo DynamoDB é descrito na RFC 4648 no site da Internet Engineering Task Force (IETF).

Programar o Amazon DynamoDB com Python e Boto3

Este guia fornece uma orientação para programadores que desejam usar o Amazon DynamoDB com Python. Saiba mais sobre as diferentes camadas de abstração, gerenciamento de configuração, tratamento de erros, controle de políticas de novas tentativas, gerenciamento de keep-alive e muito mais.

Tópicos

- Sobre o Boto
- Usar a documentação do Boto
- Noções básicas sobre as camadas de abstração do cliente e de recursos
- Usar o recurso de tabela batch_writer

- Exemplos de código adicionais que exploram as camadas do cliente e de recursos
- Noções básicas sobre como os objetos Client e Resource interagem com sessões e threads
- Personalizar o objeto Config
- Tratamento de erros
- Registro em log
- Hooks de eventos
- Paginação e o paginador
- Waiters

Sobre o Boto

É possível acessar o DynamoDB por meio do Python usando o SDK da AWS oficial para Python, geralmente chamado de Boto3. O nome Boto se origina de um golfinho de água doce nativo do Rio Amazonas. A biblioteca Boto3 é a terceira versão principal da biblioteca, lançada pela primeira vez em 2015. A biblioteca Boto3 é bem grande, pois comporta todos os serviços da AWS, não apenas o DynamoDB. Essa orientação visa somente às partes do Boto3 relevantes para o DynamoDB.

O Boto é mantido pela AWS como um projeto de código aberto hospedado no GitHub. Ele é dividido em dois pacotes: Botocore e Boto3.

- O Botocore oferece a funcionalidade de nível inferior. No Botocore, você encontrará o cliente, a sessão, as credenciais, a configuração e as classes de exceção.
- O Boto3 se baseia no Botocore. Ele oferece uma interface de nível superior, mais relacionada ao Python. Especificamente, ele expõe uma tabela do DynamoDB como um recurso e oferece uma interface mais simples e elegante em comparação à interface de cliente orientada a serviços, de nível inferior.

Como esses projetos estão hospedados no GitHub, é possível visualizar o código-fonte, rastrear problemas abertos ou enviar seus próprios problemas.

Usar a documentação do Boto

Comece a usar a documentação do Boto com os seguintes recursos:

- Comece com a seção Início rápido, que fornece um ponto de partida sólido para a instalação do pacote. Acesse-a para receber instruções sobre como instalar o Boto3, caso ele ainda não esteja

instalado (em geral, o Boto3 é disponibilizado automaticamente nos serviços da AWS, como o AWS Lambda.

- Depois disso, concentre-se no guia do DynamoDB da documentação. Ele mostra como realizar as atividades básicas do DynamoDB: criar e excluir tabelas, manipular itens, além de realizar operações em lote, consultas e verificações. Seus exemplos usam a interface de recursos. A exibição de `boto3.resource('dynamodb')` indica que você está usando a interface de recursos de nível superior.
- Depois do guia, você pode analisar a referência do DynamoDB. Essa página de pouso fornece uma lista completa das classes e dos métodos disponíveis. Na parte superior, será exibida a classe `DynamoDB.Client`. Ela concede acesso de nível inferior a todas as operações do ambiente de gerenciamento e do plano de dados. Na parte inferior, é exibida a classe `DynamoDB.ServiceResource`. Trata-se da interface relacionada ao Python de nível superior. Com ela, é possível criar tabelas, realizar operações em lote entre tabelas ou ter acesso a uma instância `DynamoDB.ServiceResource.Table` para ações específicas de tabelas.

Noções básicas sobre as camadas de abstração do cliente e de recursos

As duas interfaces com as quais você trabalhará são a interface do cliente e a interface de recursos.

- A interface do cliente de nível inferior oferece um mapeamento de um para um para a API de serviço subjacente. Todas as APIs oferecidas pelo DynamoDB estão disponíveis por meio do cliente. Isso significa que a interface do cliente pode fornecer funcionalidade completa, mas geralmente é mais detalhada e complexa de usar.
- A interface de recursos de nível superior não fornece um mapeamento de um para um da API de serviço subjacente. No entanto, ela oferece métodos que tornam mais conveniente o acesso ao serviço, como `batch_writer`.

Veja a seguir um exemplo de inserção de um item usando a interface do cliente. Observe como todos os valores são transmitidos como um mapa com a chave indicando o tipo (“S” para string, “N” para número) e o valor como string. Isso é conhecido como formato JSON do DynamoDB.

```
import boto3

dynamodb = boto3.client('dynamodb')

dynamodb.put_item(
    TableName='YourTableName',
```

```
    Item={
        'pk': {'S': 'id#1'},
        'sk': {'S': 'cart#123'},
        'name': {'S': 'SomeName'},
        'inventory': {'N': '500'},
        # ... more attributes ...
    }
)
```

Aqui está a mesma operação `PutItem` usando a interface de recursos. A digitação dos dados está implícita:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

table.put_item(
    Item={
        'pk': 'id#1',
        'sk': 'cart#123',
        'name': 'SomeName',
        'inventory': 500,
        # ... more attributes ...
    }
)
```

Se necessário, será possível converter entre JSON normal e JSON do DynamoDB usando as classes `TypeSerializer` e `TypeDeserializer` fornecidas com o boto3:

```
def dynamo_to_python(dynamo_object: dict) -> dict:
    deserializer = TypeDeserializer()
    return {
        k: deserializer.deserialize(v)
        for k, v in dynamo_object.items()
    }

def python_to_dynamo(python_object: dict) -> dict:
    serializer = TypeSerializer()
    return {
```

```
        k: serializer.serialize(v)
        for k, v in python_object.items()
    }
```

Veja como realizar uma consulta usando a interface do cliente. Ela expressa a consulta como uma estrutura JSON. Ela usa uma string `KeyConditionExpression` que exige a substituição de variáveis para lidar com possíveis conflitos de palavras-chave:

```
import boto3

client = boto3.client('dynamodb')

# Construct the query
response = client.query(
    TableName='YourTableName',
    KeyConditionExpression='pk = :pk_val AND begins_with(sk, :sk_val)',
    FilterExpression='#name = :name_val',
    ExpressionAttributeValues={
        ':pk_val': {'S': 'id#1'},
        ':sk_val': {'S': 'cart#'},
        ':name_val': {'S': 'SomeName'},
    },
    ExpressionAttributeNames={
        '#name': 'name',
    }
)
```

A mesma operação de consulta usando a interface de recursos pode ser reduzida e simplificada:

```
import boto3
from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

response = table.query(
    KeyConditionExpression=Key('pk').eq('id#1') & Key('sk').begins_with('cart#'),
    FilterExpression=Attr('name').eq('SomeName')
)
```

Como exemplo final, imagine que você queira ter o tamanho aproximado de uma tabela (que são metadados mantidos na tabela que são atualizados a cada seis horas). Com a interface do cliente,

você precisa realizar uma operação `describe_table()` e extrair a resposta da estrutura JSON exibida:

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

Com a interface de recursos, a tabela realiza a operação describe implicitamente e apresenta os dados diretamente como um atributo:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> Ao pensar na possibilidade de desenvolver usando a interface do cliente ou de recursos, esteja ciente de que novos recursos não serão adicionados à interface de recursos de acordo com a documentação de recursos: “A equipe do AWS SDK para Python não pretende adicionar novos recursos à interface de recursos no boto3. As interfaces existentes continuarão funcionando durante o ciclo de vida do boto3. Os clientes podem encontrar acesso a novos recursos de serviço por meio da interface do cliente”.

Usar o recurso de tabela batch_writer

Uma praticidade disponível somente com o recurso de tabela de nível superior é o `batch_writer`. O DynamoDB comporta operações de gravação em lote, permitindo até 25 operações put ou delete em uma solicitação de rede. O agrupamento em lotes como esse melhora a eficiência ao minimizar as viagens de ida e volta da rede.

Com a biblioteca de cliente de nível inferior, você deve usar a operação `client.batch_write_item()` para executar lotes. É necessário dividir manualmente o trabalho

em lotes de 25. Depois de cada operação, você também precisa solicitar o recebimento de uma lista de itens não processados (algumas das operações de gravação podem ser bem-sucedidas, enquanto outras podem falhar). Depois, é necessário transmitir esses itens não processados novamente para uma operação `batch_write_item()` posterior. Há uma quantidade significativa de código clichê.

O método Table.batch_writer cria um gerenciador de contexto para gravar objetos em um lote. Ele apresenta uma interface em que parece que você está escrevendo itens um de cada vez, mas internamente está armazenando em buffer e enviando os itens em lotes. Ele também lida com novas tentativas de itens não processados implicitamente.

```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

movies = # long list of movies in {'pk': 'val', 'sk': 'val', etc} format
with table.batch_writer() as writer:
    for movie in movies:
        writer.put_item(Item=movie)
```

Exemplos de código adicionais que exploram as camadas do cliente e de recursos

Também é possível consultar os seguintes repositórios de exemplos de código que exploram o uso das várias funções, usando o cliente e o recurso:

- Exemplos oficiais de código de ação única da AWS.
- Exemplos oficiais de código orientado por cenários da AWS.
- Exemplos de códigos de ação única mantidos pela comunidade.

Noções básicas sobre como os objetos Client e Resource interagem com sessões e threads

O objeto Resource não é seguro para threads e não deve ser compartilhado entre threads ou processos. Consulte o guia sobre recursos para ter mais detalhes.

O objeto Client, por outro lado, geralmente é seguro para threads, exceto para recursos avançados específicos. Consulte o guia sobre clientes para ter mais detalhes.

O objeto Session não é seguro para threads. Portanto, toda vez que você criar um cliente ou um recurso em um ambiente de vários threads, primeiro será necessário criar uma sessão e, depois, o cliente ou o recurso por meio da sessão. Consulte o guia sobre sessões para ter mais detalhes.

Ao chamar o `boto3.resource()`, você está usando implicitamente a sessão padrão. Isso é conveniente para escrever código de thread único. Ao escrever código de vários threads, primeiro é necessário criar uma sessão para cada thread e, depois, recuperar o recurso dessa sessão:

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Personalizar o objeto Config

Ao criar um objeto Client ou Resource, é possível transmitir parâmetros nomeados opcionais para personalizar o comportamento. O parâmetro chamado `config` disponibiliza uma série de funcionalidades. É uma instância de `botocore.client.Config` e a documentação de referência do Config mostra todos os itens expostos a serem controlados. O guia de configuração fornece uma visão geral útil.

> **Note**
>
> É possível modificar muitas dessas configurações comportamentais em nível de sessão, no arquivo de configuração AWS ou como variáveis de ambiente.

Configuração de tempos limite

Um dos usos de uma configuração personalizada é ajustar os comportamentos de rede:

- connect_timeout (float ou int): o tempo em segundos até que uma exceção de tempo limite seja lançada ao tentar fazer uma conexão. O padrão é 60 segundos.
- connect_timeout (float ou int): o tempo em segundos até que uma exceção de tempo limite seja lançada ao tentar estabelecer uma conexão. O padrão é 60 segundos.

Tempos limite de sessenta segundos são excessivos para o DynamoDB. Isso significa que uma falha transitória na rede causará um minuto de atraso para o cliente antes que ele possa tentar novamente. O código a seguir reduz os tempos limite para um segundo:

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

Para ler mais discussões sobre tempos limite, consulte Tuning AWS Java SDK HTTP request settings for latency-aware DynamoDB applications. Observe que o SDK do Java tem mais configurações de tempo limite do que o Python.

Configuração de keep-alive

Se estiver usando o botocore 1.27.84 ou posterior, você também poderá controlar o TCP Keep-Alive:

- tcp_keepalive (bool): habilita a opção de soquete TCP Keep-Alive usada ao criar conexões, se definida como `True` (o padrão é `False`). O recurso está disponível apenas a partir do botocore 1.27.84.

Configurar o TCP Keep-Alive como `True` pode reduzir as latências médias. Veja um exemplo de código que define condicionalmente o TCP Keep-Alive como verdadeiro quando você tem a versão correta do botocore:

```
import botocore
import boto3
from botocore.config import Config
from distutils.version import LooseVersion

required_version = "1.27.84"
current_version = botocore.__version__

my_config = Config(
   connect_timeout = 0.5,
   read_timeout = 0.5
)
if LooseVersion(current_version) > LooseVersion(required_version):
    my_config = my_config.merge(Config(tcp_keepalive = True))

dynamodb = boto3.resource('dynamodb', config=my_config)
```

> **Note**
>
> O TCP Keep-Alive é diferente do HTTP Keep-Alive. Com o TCP Keep-Alive, pacotes pequenos são enviados pelo sistema operacional subjacente pela conexão do soquete para manter a conexão ativa e detectar imediatamente qualquer queda. Com o HTTP Keep-Alive, a conexão da web baseada no soquete subjacente é reutilizada. O HTTP Keep-Alive está sempre habilitado com o boto3.

Há um limite de quanto tempo uma conexão inativa pode ser mantida ativa. Pense em enviar solicitações periódicas (digamos a cada minuto) se tiver uma conexão inativa, mas quiser que a próxima solicitação use uma conexão já estabelecida.

Configuração de novas tentativas

A configuração também aceita um dicionário chamado novas tentativas, no qual é possível especificar o comportamento de novas tentativas desejado. As novas tentativas acontecem no SDK quando o SDK recebe um erro e o erro é de um tipo transitório. Se um erro for repetido internamente (e uma nova tentativa, por fim, produzir uma resposta bem-sucedida), do ponto de vista do código de chamada, não haverá erro, apenas uma latência ligeiramente elevada. Veja os valores que você pode especificar:

- max_attempts: um número inteiro que representa o número máximo de novas tentativas que serão feitas em uma única solicitação. Por exemplo, definir esse valor como dois fará com que a solicitação seja repetida no máximo duas vezes após a solicitação inicial. Definir esse valor como zero não vai ocasionar nenhuma nova tentativa após a solicitação inicial.
- total_max_attempts: um número inteiro que representa o número máximo total de tentativas que serão feitas em uma única solicitação. Isso inclui a solicitação inicial, portanto, um valor de um indica que nenhuma solicitação será repetida. Se `total_max_attempts` e `max_attempts` forem fornecidos, `total_max_attempts` terá precedência. `total_max_attempts` é preferível a `max_attempts` porque é associado à variável de ambiente `AWS_MAX_ATTEMPTS` e ao valor do arquivo de configuração `max_attempts`.
- mode: uma string que representa o tipo de modo de nova tentativa que o botocore deve usar. Os valores válidos são:
  - legacy: o modo padrão. Espera 50 ms na primeira tentativa e, depois, usa recuo exponencial com um fator de base dois. Em relação ao DynamoDB, ele executa até dez tentativas no total, (a menos que seja substituído pelo valor acima).

> **Note**
>
> Com um recuo exponencial, a última tentativa aguardará quase 13 segundos.

- standard: padrão nomeado porque é mais consistente com outros SDKs da AWS. Espera por um período aleatório que varia de 0 ms a 1.000 ms pela primeira nova tentativa. Se outra nova tentativa for necessária, ele escolherá outro período aleatório de 0 ms a 1.000 ms e o multiplicará por 2. Se for necessária uma nova tentativa, ele fará a mesma escolha aleatória multiplicada por quatro e assim por diante. Cada espera é limitada a 20 segundos. Esse modo executará novas tentativas em mais condições de falha detectadas do que o modo `legacy`. Em relação ao DynamoDB, ele executa até três tentativas no total, (a menos que seja substituído pelo valor acima).
- adaptável: modo de novas tentativas experimental que inclui toda a funcionalidade do modo padrão, mas inclui controle de utilização automática do lado do cliente. Com a limitação de taxa adaptável, os SDKs podem diminuir a taxa na qual as solicitações são enviadas para acomodar melhor a capacidade dos serviços da AWS. Esse é um modo provisório cujo comportamento pode mudar.

Uma definição expandida desses modos de novas tentativas pode ser encontrada no guia de novas tentativas, bem como no tópico Retry behavior na referência do SDK.

Veja um exemplo que usa explicitamente a política de novas tentativas `legacy` com, no máximo, três 3 solicitações no total (duas novas tentativas):

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0,
   retries = {
     'mode': 'legacy',
     'total_max_attempts': 3
   }
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

Como o DynamoDB é um sistema de alta disponibilidade e baixa latência, convém adotar uma postura mais incisiva em relação à velocidade das novas tentativas do que as respectivas políticas permitem. É possível implementar sua própria política de novas tentativas definindo o máximo de tentativas como zero, detectando por conta própria as exceções e tentando novamente, conforme apropriado, com o próprio código, em vez de recorrer ao boto3 para fazer novas tentativas implícitas.

Se você gerencia sua própria política de novas tentativas, convém diferenciar entre controles de utilização e erros:

- Um controle de utilização (designado por um `ProvisionedThroughputExceededException` ou `ThrottlingException`) indica um serviço íntegro que está informando que você excedeu sua capacidade de leitura ou gravação em uma tabela ou partição do DynamoDB. A cada milissegundo que se passa, um pouco mais de capacidade de leitura ou gravação é disponibilizada e, portanto, é possível realizar novas tentativas com rapidez (por exemplo, a cada 50 ms) para tentar acessar essa capacidade recém-liberada. Com os controles de utilização, não é especialmente necessário um recuo exponencial porque os controles de utilização são leves para o DynamoDB exibir e não cobram por solicitação. O recuo exponencial atribui atrasos maiores aos threads do cliente que já esperaram por mais tempo, o que estende estatisticamente o p50 e o p99.
- Um erro (designado por um `InternalServerError` ou um `ServiceUnavailable`, entre outros) indica um problema transitório com o serviço. Isso pode se relacionar à toda a tabela ou possivelmente apenas à partição na qual você está lendo ou gravando. Com erros, é possível pausar por mais tempo antes de novas tentativas, (como 250 ms ou 500 ms), e usar a instabilidade para escalonar as novas tentativas.

Configuração de conexões máximas de grupo

Por fim, a configuração permite controlar o tamanho do grupo de conexões:

- max_pool_connections (int): o número máximo de conexões a serem mantidas em um grupo de conexões. Se esse valor não for definido, será usado o valor padrão dez.

Essa opção controla o número máximo de conexões HTTP a serem mantidas agrupadas para reutilização. Um grupo diferente é mantido por sessão. Ao prever que mais de dez threads serão direcionados para clientes ou recursos criados na mesma sessão, pense em aumentar isso, para que os threads não precisem esperar por outros threads usando uma conexão em grupo.

```
import boto3
from botocore.config import Config

my_config = Config(
   max_pool_connections = 20
)

# Setup a single session holding up to 20 pooled connections
session = boto3.Session(my_config)

# Create up to 20 resources against that session for handing to threads
# Notice the single-threaded access to the Session and each Resource
resource1 = session.resource('dynamodb')
resource2 = session.resource('dynamodb')
# etc
```

Tratamento de erros

Nem todas as exceções de serviço da AWS são definidas estaticamente no Boto3. Isso ocorre porque os erros e as exceções dos serviços da AWS variam muito e estão sujeitos a alterações. O Boto3 agrupa todas as exceções de serviço como um `ClientError` e expõe os detalhes como JSON estruturado. Por exemplo, uma resposta de erro pode ser estruturada assim:

```
{
    'Error': {
        'Code': 'SomeServiceException',
        'Message': 'Details/context around the exception or error'
    },
    'ResponseMetadata': {
        'RequestId': '1234567890ABCDEF',
        'HostId': 'host ID data will appear here as a hash',
        'HTTPStatusCode': 400,
        'HTTPHeaders': {'header metadata key/values will appear here'},
        'RetryAttempts': 0
    }
}
```

O código a seguir captura todas as exceções `ClientError` e examina o valor da string do `Code` no `Error` para determinar qual ação realizar:

```
import botocore
```

```
import boto3

dynamodb = boto3.client('dynamodb')

try:
    response = dynamodb.put_item(...)

except botocore.exceptions.ClientError as err:
    print('Error Code: {}'.format(err.response['Error']['Code']))
    print('Error Message: {}'.format(err.response['Error']['Message']))
    print('Http Code: {}'.format(err.response['ResponseMetadata']['HTTPStatusCode']))
    print('Request ID: {}'.format(err.response['ResponseMetadata']['RequestId']))

    if err.response['Error']['Code'] in ('ProvisionedThroughputExceededException',
 'ThrottlingException'):
        print("Received a throttle")
    elif err.response['Error']['Code'] == 'InternalServerError':
        print("Received a server error")
    else:
        raise err
```

Alguns códigos de exceção (mas não todos) foram materializados como classes de nível superior. É possível optar por lidar com eles diretamente. Ao usar a interface do cliente, essas exceções são preenchidas dinamicamente no cliente e você captura essas exceções usando sua instância cliente, da seguinte maneira:

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Ao usar a interface de recursos, é necessário usar `.meta.client` para fazer o percurso entre o recurso e o cliente subjacente a fim de acessar as exceções, da seguinte maneira:

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

Para analisar a lista de tipos de exceções materializadas, é possível gerar a lista dinamicamente:

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

Ao realizar uma operação de gravação com uma expressão condicional, é possível solicitar que, se a expressão falhar, o valor do item seja exibido na resposta de erro.

```
try:
    response = table.put_item(
        Item=item,
        ConditionExpression='attribute_not_exists(pk)',
        ReturnValuesOnConditionCheckFailure='ALL_OLD'
    )
except table.meta.client.exceptions.ConditionalCheckFailedException as e:
    print('Item already exists:', e.response['Item'])
```

Para ler mais sobre tratamento de erros e exceções:

- O guia do boto3 sobre tratamento de erros tem mais informações sobre técnicas de tratamento de erros.
- A seção do Guia do desenvolvedor do DynamoDB sobre erros de programação lista os erros que você pode encontrar.
- A seção Common Errors na referência da API.
- A documentação sobre cada operação de API lista quais erros essa chamada pode gerar (por exemplo, BatchWriteItem).

Registro em log

A biblioteca do boto3 se integra ao módulo de registro em log integrado do Python para monitorar o que acontece durante uma sessão. Para controlar os níveis de registro em log, é possível configurar o módulo de registro em log:

```
import logging

logging.basicConfig(level=logging.INFO)
```

Isso configura o logger raiz para registrar em log `INFO` e mensagens de nível superior. Mensagens de registro em log que forem menos rígidas do que o nível serão ignoradas. Os níveis de registro em log incluem `DEBUG`, `INFO`, `WARNING`, `ERROR` e `CRITICAL`. O padrão é `WARNING`.

Os loggers no boto3 são hierárquicos. A biblioteca usa alguns loggers diferentes, cada um correspondendo a diferentes partes da biblioteca. É possível controlar separadamente o comportamento de cada um:

- boto3: o logger principal do módulo boto3.

- botocore: o logger principal do pacote do botocore.
- botocore.auth: usado para registrar em log a criação de assinaturas da AWS para solicitações.
- botocore.credentials: usado para registrar em log o processo de busca e atualização de credenciais.
- botocore.endpoint: usado para registrar em log a criação da solicitação antes de ser enviada pela rede.
- botocore.hooks: usado para registrar em log eventos acionados na biblioteca.
- botocore.loaders: usado para registrar em log quando partes dos modelos de serviço da AWS são carregadas.
- botocore.parsers: usado para registrar em log as respostas do serviço da AWS antes de serem analisadas.
- botocore.retryhandler: usado para registrar em log o processamento de novas tentativas de solicitação do serviço da AWS (modo herdado).
- botocore.retries.standard: usado para registrar em log o processamento de novas tentativas de solicitação do serviço da AWS (modo padrão ou adaptável).
- botocore.utils: usado para registrar em log atividades diversas na biblioteca.
- botocore.waiter: usado para registrar em log a funcionalidade dos waiters, que pesquisam um serviço da AWS até que determinado estado seja atingido.

Outras bibliotecas também realizam o registro em log. Internamente, o boto3 usa o urllib3 de terceiros para lidar com a conexão HTTP. Quando a latência é importante, é possível observar os logs para garantir que o grupo esteja sendo bem utilizado, vendo quando o urllib3 estabelece uma nova conexão ou fecha uma ociosa.

- urllib3.connectionpool: use para registrar em log eventos de tratamento de eventos do grupo de conexões.

O trecho de código a seguir define a maioria dos registros em log como `INFO` com o registro em log de `DEBUG` da atividade do endpoint e do grupo de conexões:

```
import logging

logging.getLogger('boto3').setLevel(logging.INFO)
logging.getLogger('botocore').setLevel(logging.INFO)
logging.getLogger('botocore.endpoint').setLevel(logging.DEBUG)
```

```
logging.getLogger('urllib3.connectionpool').setLevel(logging.DEBUG)
```

Hooks de eventos

O Botocore emite eventos durante várias partes de sua execução. É possível registrar manipuladores para esses eventos para que, sempre que um evento for emitido, seu manipulador seja chamado. Isso permite que você estenda o comportamento do botocore sem precisar modificar os componentes internos.

Por exemplo, digamos que você queira acompanhar cada vez que uma operação `PutItem` for chamada em qualquer tabela do DynamoDB na aplicação. É possível se inscrever no evento `'provide-client-params.dynamodb.PutItem'` para capturar e registrar toda vez que uma operação `PutItem` for invocada na sessão associada. Veja um exemplo abaixo:

```
import boto3
import botocore
import logging

def log_put_params(params, **kwargs):
    if 'TableName' in params and 'Item' in params:
        logging.info(f"PutItem on table {params['TableName']}: {params['Item']}")

logging.basicConfig(level=logging.INFO)

session = boto3.Session()
event_system = session.events

# Register our interest in hooking in when the parameters are provided to PutItem
event_system.register('provide-client-params.dynamodb.PutItem', log_put_params)

# Now, every time you use this session to put an item in DynamoDB,
# it will log the table name and item data.
dynamodb = session.resource('dynamodb')
table = dynamodb.Table('YourTableName')
table.put_item(
    Item={
        'pk': '123',
        'sk': 'cart#123',
        'item_data': 'YourItemData',
        # ... more attributes ...
    }
)
```

No manipulador, é possível até mesmo manipular os parâmetros programaticamente para alterar o comportamento:

```
params['TableName'] = "NewTableName"
```

Para ter mais informações sobre eventos, consulte a documentação do botocore sobre eventos e a documentação do boto3 sobre eventos.

Paginação e o paginador

Algumas solicitações, como Query e Scan, limitam o tamanho dos dados exibidos em uma única solicitação e exigem que você faça solicitações repetidas para extrair as páginas subsequentes.

É possível controlar o número máximo de itens a serem lidos em cada página com o parâmetro `limit`. Por exemplo, se você quiser os dez últimos itens, poderá usar `limit` para recuperar somente os últimos dez itens. Observe que esse limite é quantos itens devem ser lidos da tabela antes que qualquer filtragem seja aplicada. Não há como especificar que você deseja exatamente dez após a filtragem; só é possível controlar a contagem pré-filtrada e conferir o lado do cliente quando realmente tiver recuperado dez itens. Independentemente do limite, cada resposta sempre tem um tamanho máximo de 1 MB.

Se a resposta incluir uma `LastEvaluatedKey`, isso indica que a resposta foi encerrada porque atingiu um limite de contagem ou tamanho. A chave é a última avaliada para a resposta. É possível recuperar essa `LastEvaluatedKey` e transmiti-la para uma chamada de acompanhamento como `ExclusiveStartKey` para ler a próxima parte desse ponto de partida. Quando não há `LastEvaluatedKey` exibida, isso significa que não há mais itens que correspondam a Query ou Scan.

Veja um exemplo simples (usando a interface de recursos, mas a interface do cliente tem o mesmo padrão) que lê no máximo cem itens por página e se repete até que todos os itens tenham sido lidos.

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

query_params = {
    'KeyConditionExpression': Key('pk').eq('123') & Key('sk').gt(1000),
    'Limit': 100
}
```

```
while True:
    response = table.query(**query_params)

    # Process the items however you like
    for item in response['Items']:
        print(item)

    # No LastEvaluatedKey means no more items to retrieve
    if 'LastEvaluatedKey' not in response:
        break

    # If there are possibly more items, update the start key for the next page
    query_params['ExclusiveStartKey'] = response['LastEvaluatedKey']
```

Por conveniência, o boto3 pode fazer isso por você com paginadores. No entanto, ele só funciona com a interface do cliente. Veja o código reformulado para usar paginadores:

```
import boto3

dynamodb = boto3.client('dynamodb')

paginator = dynamodb.get_paginator('query')

query_params = {
    'TableName': 'YourTableName',
    'KeyConditionExpression': 'pk = :pk_val AND sk > :sk_val',
    'ExpressionAttributeValues': {
        ':pk_val': {'S': '123'},
        ':sk_val': {'N': '1000'},
    },
    'Limit': 100
}

page_iterator = paginator.paginate(**query_params)

for page in page_iterator:
    # Process the items however you like
    for item in page['Items']:
        print(item)
```

Para ter mais informações, consulte o Guia sobre paginadores e a Referência da API para DynamoDB.Paginator.Query.

> **Note**
>
> Os paginadores também têm suas próprias configurações chamadas `MaxItems`, `StartingToken` e `PageSize`. Para paginar com o DynamoDB, você deve ignorar essas configurações.

Waiters

Os waiters oferecem a capacidade de esperar que algo seja concluído antes de continuar. No momento, eles comportam apenas a espera pela criação ou a exclusão de uma tabela. Em segundo plano, a operação waiter faz uma verificação para você a cada 20 segundos até 25 vezes. É possível fazer isso por conta própria, mas o uso de um waiter é uma solução mais sofisticada ao elaborar automações.

Esse código mostra como esperar que uma tabela específica seja criada:

```
# Create a table, wait until it exists, and print its ARN
response = client.create_table(...)
waiter = client.get_waiter('table_exists')
waiter.wait(TableName='YourTableName')
print('Table created:', response['TableDescription']['TableArn']
```

Para ter mais informações, consulte o Guia de waiters e a Referência sobre waiters.

Programar o Amazon DynamoDB com JavaScript

Este guia fornece uma orientação para programadores que desejam usar o Amazon DynamoDB com JavaScript. Saiba mais sobre o AWS SDK para JavaScript, camadas de abstração disponíveis, configuração de conexões, tratamento de erros, definição de políticas de novas tentativas, gerenciamento de keep alive e muito mais.

Tópicos

- Sobre o AWS SDK para JavaScript
- Usar o AWS SDK para JavaScript V3
- Acessar a documentação do JavaScript
- Camadas de abstração

- Usar a função de utilitário de ordenação
- Ler itens
- Gravações condicionais
- Paginação
- Especificar a configuração
- Waiters
- Gerenciamento de erros
- Registro em log
- Considerações

Sobre o AWS SDK para JavaScript

O AWS SDK para JavaScript concede acesso a Serviços da AWS usando scripts do navegador ou Node.js. Esta documentação concentra-se na versão mais recente do SDK (V3). O AWS SDK para JavaScript V3 é mantido pela AWS como um projeto de código aberto hospedado no GitHub. Os problemas e as solicitações de recursos são públicos e você pode acessá-los na página de problemas do repositório do GitHub.

O JavaScript V2 é semelhante ao V3, mas contém diferenças de sintaxe. A V3 é mais modular, facilitando o envio de dependências menores e tem suporte de primeira classe para TypeScript. Recomendamos o uso da versão mais recente do SDK.

Usar o AWS SDK para JavaScript V3

É possível adicionar o SDK ao à aplicação Node.js usando o Node Package Manager. Os exemplos abaixo mostram como adicionar os pacotes de SDK mais comuns para trabalhar com o DynamoDB.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

A instalação de pacotes adiciona referências à seção de dependências do arquivo de projeto package.json. Você tem a opção de usar a sintaxe mais recente do módulo ECMAScript. Para ter mais detalhes sobre essas duas abordagens, consulte a seção Considerações.

Acessar a documentação do JavaScript

Comece a usar a documentação do JavaScript com os seguintes recursos:

- Acesse o Guia do desenvolvedor para ver a documentação básica do JavaScript. As instruções de instalação estão localizadas na seção Configuração.
- Acesse a documentação de referência da API para examinar todas as classes e os métodos disponíveis.
- O SDK para JavaScript é compatível com muitos outros Serviços da AWS além do DynamoDB. Use o seguinte procedimento para localizar uma cobertura de API específica para o DynamoDB:

  1. Em Serviços, selecione DynamoDB e bibliotecas. Isso documenta o cliente de nível baixo.
  2. Selecione lib-dynamodb. Isso documenta o cliente de alto nível. Os dois clientes representam duas camadas de abstração diferentes que você tem a opção de usar. Consulte a seção a seguir para ter mais informações sobre camadas de abstração.

Camadas de abstração

O SDK para JavaScript V3 tem um cliente de nível inferior (`DynamoDBClient`) e um cliente de alto nível (`DynamoDBDocumentClient`).

Tópicos

- Cliente de nível inferior (DynamoDBClient)
- Cliente de alto nível (DynamoDBDocumentClient)

Cliente de nível inferior (**`DynamoDBClient`**)

O cliente de nível inferior não fornece abstrações extras sobre o protocolo de conexão subjacente. Ele oferece controle total sobre todos os aspectos da comunicação, mas como não há abstrações, é necessário realizar tarefas, como fornecer definições de itens usando o formato JSON do DynamoDB.

Como mostra o exemplo abaixo, com esse formato, os tipos de dados devem ser declarados explicitamente. Um S indica um valor de string e um N indica um valor numérico. Os números na rede são sempre enviados como strings marcadas como tipos de números para garantir que não haja perda de precisão. As chamadas de API de nível inferior têm um padrão de nomenclatura, como `PutItemCommand` e `GetItemCommand`.

O exemplo a seguir está usando um cliente de nível inferior com um `Item` definido usando JSON do DynamoDB:

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      "id": { S: "Product01" },
      "description": { S: "Hiking Boots" },
      "category": { S: "footwear" },
      "sku": { S: "hiking-sku-01" },
      "size": { N: "9" }
    }
  };

  try {
    const data = await client.send(new PutItemCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

Cliente de alto nível (**`DynamoDBDocumentClient`**)

O cliente de documentos de alto nível do DynamoDB oferece recursos de conveniência integrados, como eliminar a necessidade de organizar dados manualmente e permitir leituras e gravações diretas usando objetos JavaScript padrão. A documentação do `lib-dynamodb` fornece a lista de vantagens.

Para instanciar o `DynamoDBDocumentClient`, crie um `DynamoDBClient` de nível inferior e, depois, envolva-o com um `DynamoDBDocumentClient`. A convenção de nomenclatura da função difere um pouco entre os dois pacotes. Por exemplo, o nível inferior usa `PutItemCommand` enquanto o alto nível usa `PutCommand`. Os nomes distintos permitem que os dois conjuntos de funções coexistam no mesmo contexto, permitindo que você misture os dois no mesmo script.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
```

```
const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    },
  };

  try {
    const data = await docClient.send(new PutCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

addProduct();
```

O padrão de uso é consistente ao ler itens usando operações de API, como `GetItem`, `Query` ou `Scan`.

Usar a função de utilitário de ordenação

É possível usar o cliente de nível inferior e ordenar ou desordenar os tipos de dados por conta própria. O pacote de utilitários, util-dynamodb, tem uma função de utilitário `marshall()` que aceita JSON e produz JSON do DynamoDB, além de uma função un`marshall()` que faz o contrário. O exemplo a seguir usa o cliente de nível inferior com a ordenação de dados gerenciada pela chamada `marshall()`.

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
const { marshall } = require("@aws-sdk/util-dynamodb");
```

```
const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: marshall({
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    }),
  };

  try {
    const data = await client.send(new PutItemCommand(params));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

Ler itens

Para ler um único item do DynamoDB use a operação de API `GetItem`. Semelhante ao comando `PutItem`, você tem a opção de usar o cliente de nível inferior ou o cliente Document de alto nível. O exemplo abaixo demonstra o uso do cliente Document de alto nível para recuperar um item.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, GetCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function getProduct() {
  const params = {
    TableName: "products",
    Key: {
      id: "Product01",
    },
  };
```

```
  try {
    const data = await docClient.send(new GetCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

getProduct();
```

Use a operação de API `Query` para ler vários itens. É possível usar o cliente de nível inferior ou o cliente Document. O exemplo abaixo usa o cliente Document de alto nível.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  QueryCommand,
} = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function productSearch() {
  const params = {
    TableName: "products",
    IndexName: "GSI1",
    KeyConditionExpression: "#category = :category and begins_with(#sku, :sku)",
    ExpressionAttributeNames: {
      "#category": "category",
      "#sku": "sku",
    },
    ExpressionAttributeValues: {
      ":category": "footwear",
      ":sku": "hiking",
    },
  };

  try {
    const data = await docClient.send(new QueryCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
```

```
  }
}

productSearch();
```

Gravações condicionais

As operações de gravação do DynamoDB podem especificar uma expressão de condição lógica que deve ser avaliada como verdadeira para que a gravação continue. Se a condição não for avaliada como verdadeira, a operação de gravação gerará uma exceção. A expressão da condição pode conferir se o item já existe ou se seus atributos correspondem a determinadas restrições.

```
ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"
```

Quando a expressão condicional falha, é possível usar `ReturnValuesOnConditionCheckFailure` para solicitar que a resposta de erro inclua o item que não atendeu às condições para deduzir qual era o problema. Para ter mais detalhes, consulte Handle conditional write errors in high concurrency scenarios with Amazon DynamoDB.

```
try {
      const response = await client.send(new PutCommand({
          TableName: "YourTableName",
          Item: item,
          ConditionExpression: "attribute_not_exists(pk)",
          ReturnValuesOnConditionCheckFailure: "ALL_OLD"
      }));
  } catch (e) {
      if (e.name === 'ConditionalCheckFailedException') {
          console.log('Item already exists:', e.Item);
      } else {
          throw e;
      }
  }
```

Exemplos de código adicionais mostrando outros aspectos do uso do SDK V3 do JavaScript estão disponíveis na documentação do SDK V3 do JavaScript e no repositório DynamoDB-SDK-Examples do GitHub.

Paginação

Tópicos

- Usar o método de conveniência paginateScan

Solicitações de leitura, como `Scan` ou `Query`, provavelmente exibirão vários itens em um conjunto de dados. Se você executar `Scan` ou `Query` com um parâmetro `Limit`, depois que o sistema ler tantos itens, uma resposta parcial será enviada e você precisará paginar para recuperar itens adicionais.

O sistema só lerá no máximo 1 megabyte de dados por solicitação. Se você incluir uma expressão `Filter`, o sistema ainda lerá um megabyte, no máximo, de dados do disco, mas exibirá os itens desse megabyte que corresponderem ao filtro. A operação de filtro pode exibir 0 itens para uma página, mas ainda exigir mais paginação antes que a pesquisa seja concluída.

Você deve procurar `LastEvaluatedKey` na resposta e usá-la como o parâmetro `ExclusiveStartKey` em uma solicitação subsequente para continuar a recuperação de dados. Isso serve como um marcador, conforme observado no exemplo a seguir.

> **Note**
>
> A amostra transmite um `lastEvaluatedKey` nulo como `ExclusiveStartKey` na primeira iteração e isso é permitido.

Exemplo: usando a `LastEvaluatedKey`:

```
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScan() {
  let lastEvaluatedKey;
  let pageCount = 0;

  do {
    const params = {
      TableName: "products",
      ExclusiveStartKey: lastEvaluatedKey,
    };

    const response = await client.send(new ScanCommand(params));
    pageCount++;
```

```
    console.log(`Page ${pageCount}, Items:`, response.Items);
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);
}

paginatedScan().catch((err) => {
  console.error(err);
});
```

Usar o método de conveniência **`paginateScan`**

O SDK fornece métodos de conveniência chamados `paginateScan` e `paginateQuery` que fazem esse trabalho para você e faz as solicitações repetidas nos bastidores. Especifique o número máximo de itens a serem lidos por solicitação usando o parâmetro `Limit` padrão.

```
const { DynamoDBClient, paginateScan } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScanUsingPaginator() {
  const params = {
    TableName: "products",
    Limit: 100
  };

  const paginator = paginateScan({client}, params);

  let pageCount = 0;

  for await (const page of paginator) {
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, page.Items);
  }
}

paginatedScanUsingPaginator().catch((err) => {
  console.error(err);
});
```

> **Note**
>
> Realizar verificações de tabela completas regularmente não é um padrão de acesso recomendado, a menos que a tabela seja pequena.

Especificar a configuração

Tópicos

- Configuração de tempos limite
- Configuração de keep-alive
- Configuração de novas tentativas

Ao configurar o `DynamoDBClient`, é possível especificar várias substituições de configuração transmitindo um objeto de configuração para o construtor. Por exemplo, é possível especificar a região à qual se conectar se ela ainda não for conhecida pelo contexto da chamada ou pelo URL do endpoint a ser usado. Isso será útil se você quiser visar a uma instância do DynamoDB Local para fins de desenvolvimento.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

Configuração de tempos limite

O DynamoDB usa HTTPS para a comunicação cliente-servidor. É possível controlar alguns aspectos da camada HTTP fornecendo um objeto `NodeHttpHandler`. Por exemplo, é possível ajustar os valores de tempo limite principais `connectionTimeout` e `requestTimeout`. O `connectionTimeout` é a duração máxima, em milissegundos, que o cliente aguardará ao tentar estabelecer uma conexão antes de desistir.

O `requestTimeout` define quanto tempo o cliente aguardará por uma resposta após o envio de uma solicitação, também em milissegundos. Os padrões para ambos são zero, o que significa que o tempo limite está desabilitado e não há limite de quanto tempo o cliente esperará se a resposta não chegar. É necessário definir os tempos limite para algo razoável para que, no caso de um problema

de rede, a solicitação seja encerrada com um erro e uma nova solicitação possa ser iniciada. Por exemplo:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";

const requestHandler = new NodeHttpHandler({
  connectionTimeout: 2000,
  requestTimeout: 2000,
});

const client = new DynamoDBClient({
  requestHandler
});
```

> **Note**
>
> O exemplo fornecido usa a importação Smithy. Smithy é uma linguagem que define serviços e SDKs, é de código aberto e mantida pela AWS.

Além de configurar valores de tempo limite, você pode definir o número máximo de soquetes, o que permite um maior número de conexões simultâneas por origem. O guia do desenvolvedor inclui detalhes sobre a configuração do parâmetro `maxSockets`.

Configuração de keep-alive

Ao usar HTTPS, a primeira solicitação sempre exige alguma comunicação de ida e volta para estabelecer uma conexão segura. O HTTP Keep-Alive permite que solicitações subsequentes reutilizem a conexão já estabelecida, tornando as solicitações mais eficientes e reduzindo a latência. O HTTP Keep-Alive está habilitado por padrão com o JavaScript V3.

Há um limite de quanto tempo uma conexão inativa pode ser mantida ativa. Pense em enviar solicitações periódicas, talvez a cada minuto, se tiver uma conexão ociosa, mas quiser que a próxima solicitação use uma conexão já estabelecida.

> **Note**
>
> Observe que na V2 antiga do SDK, o keep-alive estava desativado por padrão, o que significa que cada conexão seria fechada imediatamente após o uso. Se estiver usando a V2, poderá substituir essa configuração.

Configuração de novas tentativas

Quando o SDK recebe uma resposta de erro e o erro pode ser exibido conforme determinado pelo SDK, como uma exceção de controle de utilização ou uma exceção de serviço temporário, ele tentará novamente. Isso acontece de forma invisível para você como chamador, exceto que você note que a solicitação demorou mais para ser bem-sucedida.

O SDK para JavaScript V3 fará três solicitações no total, por padrão, antes de desistir e transmitir o erro para o contexto de chamada. É possível ajustar o número e a frequência dessas novas tentativas.

O construtor `DynamoDBClient` aceita uma configuração `maxAttempts` que limita quantas tentativas acontecerão. O exemplo abaixo eleva o valor do padrão de três para um total de cinco. Se você defini-lo como 0 ou 1, isso indica que você não quer nenhuma nova tentativa automática e quer lidar com quaisquer erros retomáveis no bloco Catch.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

Também é possível controlar o tempo das novas tentativas com uma estratégia de repetição personalizada. Para fazer isso, importe o pacote de utilitários `util-retry` e crie uma função de recuo personalizada que calcule o tempo de espera entre as novas tentativas com base na contagem atual delas.

O exemplo abaixo determina no máximo cinco tentativas com atrasos de 15, 30, 90 e 360 milissegundos caso a primeira tentativa falhe. A função de recuo personalizada, `calculateRetryBackoff`, calcula os atrasos aceitando o número de novas tentativas (começa com 1 na primeira tentativa) e exibe quantos milissegundos esperar por essa solicitação.

```
const { ConfiguredRetryStrategy } = require("@aws-sdk/util-retry");
```

```
const calculateRetryBackoff = (attempt) => {
  const backoffTimes = [15, 30, 90, 360];
  return backoffTimes[attempt - 1] || 0;
};

const client = new DynamoDBClient({
  retryStrategy: new ConfiguredRetryStrategy(
    5, // max attempts.
    calculateRetryBackoff // backoff function.
  ),
});
```

Waiters

O cliente do DynamoDB inclui duas funções waiter úteis que podem ser usadas ao criar, modificar ou excluir tabelas quando você quiser que o código espere até que a modificação da tabela seja concluída. Por exemplo, é possível implantar uma tabela, chamar a função `waitUntilTableExists` e o código será bloqueado até que a tabela se torne ATIVA. O waiter pesquisa internamente o serviço do DynamoDB com uma `describe-table` a cada vinte segundos.

```
import {waitUntilTableExists, waitUntilTableNotExists} from "@aws-sdk/client-dynamodb";

… <create table details>

const results = await waitUntilTableExists({client: client, maxWaitTime: 180},
 {TableName: "products"});
if (results.state == 'SUCCESS') {
  return results.reason.Table
}
console.error(`${results.state} ${results.reason}`);
```

O recurso `waitUntilTableExists` retorna o controle somente quando pode executar um comando `describe-table` que mostra o status da tabela ATIVO. Isso garante que você possa usar `waitUntilTableExists` para aguardar a conclusão da criação, bem como modificações, como adicionar um índice GSI, que pode levar algum tempo para ser aplicado antes que a tabela retorne ao status ATIVO.

Gerenciamento de erros

Nos primeiros exemplos aqui, detectamos todos os erros de forma ampla. No entanto, em aplicações práticas, é importante discernir entre vários tipos de erros e implementar um tratamento de erros mais preciso.

As respostas de erro do DynamoDB contêm metadados, incluindo o nome do erro. É possível capturar erros e, depois, comparar os possíveis nomes de string das condições de erro para determinar como proceder. Sobre erros do lado do servidor, é possível utilizar o operador `instanceof` com os tipos de erro exportados pelo pacote `@aws-sdk/client-dynamodb` para gerenciar o tratamento de erros com eficiência.

É importante observar que esses erros só se manifestam após o esgotamento de todas as novas tentativas. Se um erro for repetido e, por fim, seguido por uma chamada bem-sucedida, do ponto de vista do código, não haverá erro, apenas uma latência ligeiramente elevada. As novas tentativas aparecerão nos gráficos do Amazon CloudWatch como solicitações malsucedidas, como solicitações de controle de utilização ou erro. Se o cliente atingir a contagem máxima de novas tentativas, ele desistirá e gerará uma exceção. Essa é a maneira de o cliente dizer que não vai tentar novamente.

Veja abaixo um trecho que captura o erro e age com base no tipo de erro que foi exibido.

```
import {
  ResourceNotFoundException
  ProvisionedThroughputExceededException,
  DynamoDBServiceException,
} from "@aws-sdk/client-dynamodb";

try {
  await client.send(someCommand);
} catch (e) {
    if (e instanceof ResourceNotFoundException) {
      // Handle ResourceNotFoundException
    } else if (e instanceof ProvisionedThroughputExceededException) {
      // Handle ProvisionedThroughputExceededException
    } else if (e instanceof DynamoDBServiceException) {
      // Handle DynamoDBServiceException
    } else {
      // Other errors such as those from the SDK
      if (e.name === "TimeoutError") {
        // Handle SDK TimeoutError.
      } else {
```

```
        // Handle other errors.
      }
    }
}
```

Consulte the section called “Tratamento de erros” para ver as strings de erro comuns no Guia do desenvolvedor do DynamoDB. Os erros exatos possíveis com qualquer chamada de API específica podem ser encontrados na documentação dessa chamada de API, como os documentos da API Query.

Os metadados dos erros incluem propriedades adicionais, dependendo do erro. Para `TimeoutError`, os metadados incluem o número de tentativas que foram feitas e o `totalRetryDelay`, conforme mostrado abaixo.

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

Se você gerencia sua própria política de novas tentativas, convém diferenciar entre controles de utilização e erros:

- Um controle de utilização (designado por um `ProvisionedThroughputExceededException` ou `ThrottlingException`) indica um serviço íntegro que está informando que você excedeu sua capacidade de leitura ou gravação em uma tabela ou partição do DynamoDB. A cada milissegundo que se passa, um pouco mais de capacidade de leitura ou gravação é disponibilizada e, portanto, é possível tentar acessar essa capacidade recém-liberada rapidamente; por exemplo, a cada 50 ms.

  Com os controles de utilização, não é especialmente necessário um recuo exponencial porque os controles de utilização são leves para o DynamoDB exibir e não cobram por solicitação. O recuo exponencial atribui atrasos maiores aos threads do cliente que já esperaram por mais tempo, o que estende estatisticamente o p50 e o p99.
- Um erro (designado por um `InternalServerError` ou `ServiceUnavailable`, entre outros) indica um problema transitório com o serviço, possivelmente com a tabela inteira ou apenas com a partição que você está lendo ou na qual está gravando. Com erros, é possível pausar por mais

tempo antes de novas tentativas, como 250 ms ou 500 ms, e usar a instabilidade para escalonar as novas tentativas.

Registro em log

Ative o registro em log para ter mais detalhes sobre o que o SDK está fazendo. É possível definir um parâmetro no `DynamoDBClient` conforme exibido no exemplo abaixo. Mais informações de log aparecerão no console e incluirão metadados, como o código de status e a capacidade consumida. Se você executar o código localmente em uma janela de terminal, os logs aparecerão lá. Se você executar o código no AWS Lambda e tiver o Amazon CloudWatch Logs configurado, a saída do console será gravada lá.

```
const client = new DynamoDBClient({
  logger: console
});
```

Também é possível se conectar às atividades internas do SDK e realizar o registro em log personalizado à medida que determinados eventos acontecem. O exemplo abaixo usa a `middlewareStack` do cliente para interceptar cada solicitação à medida que ela é enviada do SDK e a registra em log enquanto ela está acontecendo.

```
const client = new DynamoDBClient({});

client.middlewareStack.add(
  (next) => async (args) => {
    console.log("Sending request from AWS SDK", { request: args.request });
    return next(args);
  },
  {
    step: "build",
    name: "log-ddb-calls",
  }
);
```

A `MiddlewareStack` fornece um hook avançado para observar e controlar o comportamento do SDK. Consulte o blog Introducing Middleware Stack in Modular AWS SDK para JavaScript para ter mais informações.

Considerações

Ao implementar o AWS SDK para JavaScript em seu projeto, veja a seguir alguns outros fatores a serem considerados.

Sistemas de módulos

O SDK comporta dois sistemas de módulos, CommonJS e ES (ECMAScript). O CommonJS usa a função `require`, enquanto o ES usa a palavra-chave `import`.

1. Common JS: `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES (ECMAScript: `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

O tipo de projeto determina o sistema de módulos a ser usado e é especificado na seção de tipos do arquivo package.json. O padrão é CommonJS. Use `"type": "module"` para indicar um projeto ES. Se você tiver um projeto Node.js que use o formato de pacote CommonJS, ainda poderá adicionar funções com a sintaxe mais moderna do SDK V3 Import nomeando seus arquivos de função com a extensão .mjs. Isso permitirá que o arquivo de código seja tratado como ES (ECMAScript).

Operações assíncronas

Você verá muitos exemplos de código usando retornos de chamada e promessas para lidar com o resultado das operações do DynamoDB. Com o JavaScript moderno, essa complexidade não é mais necessária e os desenvolvedores podem aproveitar a sintaxe async/await mais sucinta e legível para operações assíncronas.

Runtime de navegador da web

Desenvolvedores web e para dispositivos móveis que criam com React ou React Native podem usar o SDK para JavaScript nos projetos. Com a V2 anterior do SDK, os desenvolvedores web precisariam carregar o SDK completo no navegador, fazendo referência a uma imagem do SDK hospedada em https://sdk.amazonaws.com/js/.

Com a V3, é possível empacotar apenas os módulos de cliente V3 necessários e todas as funções do JavaScript necessárias em um único arquivo JavaScript que usa Webpack e adicioná-lo a uma tag do script no `<head>` de suas páginas HTML, conforme explicado na seção Conceitos básicos de um script de navegador da documentação do SDK.

Operações do plano de dados do DAX

O SDK para JavaScript V3 agora comporta operações de plano de dados do Amazon DynamoDB Streams Accelerator (DAX).

Programar o DynamoDB com o AWS SDK for Java 2.x

Este guia de programação fornece uma orientação para programadores que desejam usar o Amazon DynamoDB com Java. O guia aborda diversos conceitos, incluindo camadas de abstração, gerenciamento de configurações, tratamento de erros, controle de políticas de novas tentativas e gerenciamento de keep-alive.

Tópicos

- Sobre o AWS SDK for Java 2.x
- Conceitos básicos da AWS SDK for Java 2.x
- Analisar a documentação do AWS SDK for Java 2.x
- Interfaces compatíveis
- Exemplos de código adicionais
- Programação síncrona e assíncrona
- Clientes HTTP
- Configurar um cliente HTTP
- Gerenciamento de erros
- ID da solicitação da AWS
- Registro em log
- Paginação
- Anotações de classes de dados

Sobre o AWS SDK for Java 2.x

É possível acessar o DynamoDB pelo Java usando o AWS SDK para Java oficial. O SDK para Java tem duas versões: 1.x e 2.x. O fim do suporte para a versão 1.x foi anunciado em 12 de janeiro de 2024. Ele entrará no modo de manutenção em 31 de julho de 2024 e o fim do suporte está previsto para 31 de dezembro de 2025. Para novos desenvolvimentos, é altamente recomendável que você

use 2.x, que foi lançado pela primeira vez em 2018. Este guia é voltado exclusivamente para a versão 2.x e se concentra somente nas partes do SDK relevantes para o DynamoDB.

Para obter informações sobre manutenção e suporte para os SDKs da AWS, consulte AWS SDK and Tools maintenance policy and AWS SDKs and Tools version support matrix no Guia de referência de SDKs e ferramentas da AWS.

O AWS SDK for Java 2.x é uma importante reescrita do código base da versão 1.x. O SDK para Java 2.x oferece suporte aos recursos Java modernos, como a E/S sem bloqueio introduzida no Java 8. O SDK para Java 2.x também adiciona suporte a implementações de clientes HTTP conectáveis para oferecer maior flexibilidade de conexão de rede e opções de configuração.

Uma alteração notável entre o SDK para Java 1.x e o SDK para Java 2.x é o uso de um novo nome de pacote. O SDK do Java 1.x usa o nome do pacote `com.amazonaws`, enquanto o SDK do Java 2.x usa `software.amazon.awssdk`. Da mesma forma, os artefatos do Maven do SDK para Java 1.x usam o `groupId` `com.amazonaws`, enquanto os artefatos do SDK para Java 2.x usam o `groupId` `software.amazon.awssdk`.

> **Important**
>
> O AWS SDK para Java 1.x tem um pacote do DynamoDB chamado `com.amazonaws.dynamodbv2`. A descrição “v2" no nome do pacote não indica que é destinado ao Java 2 (J2SE). Em vez disso, “v2" indica que o pacote é compatível com a segunda versão da API de baixo nível do DynamoDB em vez da versão original da API de baixo nível.

Compatibilidade com versões do Java

O AWS SDK for Java 2.x é totalmente compatível com versões do Java de suporte de longo prazo (LTS).

Conceitos básicos da AWS SDK for Java 2.x

O tutorial a seguir mostra como usar o Apache Maven para definir dependências do SDK para Java 2.x. Este tutorial também mostra como escrever o código que se conecta ao DynamoDB para listar as tabelas disponíveis do DynamoDB. O tutorial deste guia é baseado no tutorial Get started with the AWS SDK for Java 2.x do Guia do desenvolvedor do AWS SDK for Java 2.x. Editamos esse tutorial para fazer chamadas para o DynamoDB e não para o Amazon S3.

Para concluir este tutorial, faça o seguinte:

- Etapa 1: configurar para este tutorial
- Etapa 2: criar o projeto
- Etapa 3: escrever o código
- Etapa 4: compilar e executar o aplicativo

Etapa 1: configurar para este tutorial

Antes de iniciar este tutorial, é necessário instalar o seguinte:

- Permissão para acessar o DynamoDB.
- Um ambiente de desenvolvimento Java configurado com acesso de autenticação única aos Serviços da AWS usando o Portal de acesso da AWS.

Para se preparar para este tutorial, siga as instruções em Setup overview no Guia do desenvolvedor do AWS SDK for Java 2.x. Depois de configurar seu ambiente de desenvolvimento com acesso de autenticação única para o SDK do Java e ter uma sessão ativa do portal de acesso da AWS, continue com a Etapa 2 deste tutorial.

Etapa 2: criar o projeto

Para criar o projeto para este tutorial, você executa um comando do Maven que solicita informações sobre como configurar o projeto. Depois que todas as informações forem inseridas e confirmadas, o Maven concluirá a construção do projeto criando um arquivo `pom.xml` e criará arquivos Java stub.

1. Abra uma janela de terminal ou prompt de comando e navegue até um diretório de sua escolha, por exemplo, sua pasta `Desktop` ou `Home`.
2. Insira o comando a seguir no terminal e pressione Enter.

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. Para cada prompt, insira o valor listado na segunda coluna.

Prompt	Valor a informar
`Define value for property 'service':`	`dynamodb`
`Define value for property 'httpClient' :`	`apache-client`
`Define value for property 'nativeImage' :`	`false`
`Define value for property 'credentialProvider'`	`identity-center`
`Define value for property 'groupId':`	`org.example`
`Define value for property 'artifactId':`	`getstarted`
`Define value for property 'version' 1.0-SNAPSHOT:`	*<Enter>*
`Define value for property 'package' org.example:`	*<Enter>*

4. Depois de inserir o último valor, o Maven lista as escolhas que você fez. Para confirmar, insira Y. Ou insira N e suas opções novamente.

O Maven cria uma pasta de projeto chamada `getstarted` com base no valor de `artifactId` que você informou. Dentro da pasta `getstarted`, encontre um arquivo chamado `README.md` que você possa revisar, um arquivo `pom.xml` e um diretório `src`.

O Maven cria a árvore de diretórios a seguir.

```
getstarted
 ### README.md
 ### pom.xml
 ### src
```

```
    ### main
    #   ### java
    #   #   ### org
    #   #       ### example
    #   #           ### App.java
    #   #           ### DependencyFactory.java
    #   #           ### Handler.java
    #   ### resources
    #       ### simplelogger.properties
    ### test
        ### java
            ### org
                ### example
                    ### HandlerTest.java

 10 directories, 7 files
```

O exemplo a seguir mostra o conteúdo do arquivo de projeto `pom.xml`.

`pom.xml`

A seção `dependencyManagement` contém uma dependência do AWS SDK for Java 2.x e a seção `dependencies` tem uma dependência do DynamoDB. A especificação dessas dependências força o Maven a incluir os arquivos `.jar` relevantes no caminho da classe Java. Por padrão, o SDK da AWS não inclui todas as classes para todos os Serviços da AWS. Para o DynamoDB, se você usar a interface de baixo nível, deverá ter uma dependência no artefato `dynamodb`. Se você usar a interface de alto nível, a dependência deverá estar no artefato `dynamodb-enhanced`. Se você não incluir as dependências relevantes, o código não será compilado. O projeto usa o Java 1.8 devido ao valor `1.8` nas propriedades `maven.compiler.source` e `maven.compiler.target`.

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>getstarted</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.shade.plugin.version>3.2.1</maven.shade.plugin.version>
        <maven.compiler.plugin.version>3.6.1</maven.compiler.plugin.version>
        <exec-maven-plugin.version>1.6.0</exec-maven-plugin.version>
        <aws.java.sdk.version>2.22.0</aws.java.sdk.version> <-------- SDK version
picked up from archetype version.
        <slf4j.version>1.7.28</slf4j.version>
        <junit5.version>5.8.1</junit5.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>bom</artifactId>
                <version>${aws.java.sdk.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>dynamodb</artifactId>  <-------- DynamoDB dependency
            <exclusions>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>netty-nio-client</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>apache-client</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sso</artifactId> <-------- Required for identity center
authentication.
        </dependency>
```

```
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>ssooidc</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>apache-client</artifactId> <-------- HTTP client specified.
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Needed to adapt Apache Commons Logging used by Apache HTTP Client to
Slf4j to avoid
        ClassNotFoundException: org.apache.commons.logging.impl.LogFactoryImpl during
runtime -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Test Dependencies -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
```

```
            <version>${junit5.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven.compiler.plugin.version}</version>
            </plugin>
        </plugins>
    </build>

</project>
```

Etapa 3: escrever o código

O código a seguir mostra a classe App criada pelo Maven. O método `main` é o ponto de entrada no aplicativo, que cria uma instância da classe `Handler` e, em seguida, chama seu método `sendRequest`.

Classe **App**

```
package org.example;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;

 public class App {
     private static final Logger logger = LoggerFactory.getLogger(App.class);

     public static void main(String... args) {
         logger.info("Application starts");

         Handler handler = new Handler();
         handler.sendRequest();

         logger.info("Application ends");
     }
 }
```

A classe `DependencyFactory` criada pelo Maven contém o método de fábrica `dynamoDbClient`, que cria e exibe uma instância de `DynamoDbClient`. A instância do `DynamoDbClient` usa uma instância do cliente HTTP baseado em Apache. Isso ocorre porque você especificou `apache-client` quando o Maven solicitou o cliente HTTP a ser usado.

O código a seguir mostra a classe `DependencyFactory`.

Classe DependencyFactory

```
package org.example;

 import software.amazon.awssdk.http.apache.ApacheHttpClient;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

 /**
  * The module containing all dependencies required by the {@link Handler}.
  */
 public class DependencyFactory {

     private DependencyFactory() {}

     /**
      * @return an instance of DynamoDbClient
      */
     public static DynamoDbClient dynamoDbClient() {
         return DynamoDbClient.builder()
                         .httpClientBuilder(ApacheHttpClient.builder())
                         .build();
     }
 }
```

A classe `Handler` contém a lógica principal do seu programa. Quando uma instância de `Handler` é criada na classe `App`, a classe `DependencyFactory` fornece o cliente de serviço do `DynamoDbClient`. O código usa a instância de `DynamoDbClient` para chamar o DynamoDB.

O Maven gera a seguinte classe `Handler` com um comentário *`TODO`*. A próxima etapa do tutorial substitui o comentário *`TODO`* pelo código.

Classe **`Handler`**, gerada pelo Maven

```
package org.example;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;


public class Handler {
    private final DynamoDbClient dynamoDbClient;

    public Handler() {
        dynamoDbClient = DependencyFactory.dynamoDbClient();
    }

    public void sendRequest() {
        // TODO: invoking the API calls using dynamoDbClient.
    }
}
```

Para preencher a lógica, substitua todo o conteúdo da classe `Handler` pelo código a seguir. O método `sendRequest` é preenchido e as importações necessárias são adicionadas.

Classe **`Handler`**, implementada

O código a seguir usa a instância `DynamoDbClient` para recuperar uma lista das tabelas existentes. Se existirem tabelas para uma conta e Região da AWS, o código usará a instância de `Logger` para registrar em log os nomes dessas tabelas.

```
package org.example;

 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
 import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;


 public class Handler {
     private final DynamoDbClient dynamoDbClient;

     public Handler() {
         dynamoDbClient = DependencyFactory.dynamoDbClient();
     }

     public void sendRequest() {
         Logger logger = LoggerFactory.getLogger(Handler.class);
```

```
        logger.info("calling the DynamoDB API to get a list of existing tables");
        ListTablesResponse response = dynamoDbClient.listTables();

        if (!response.hasTableNames()) {
            logger.info("No existing tables found for the configured account &
region");
        } else {
            response.tableNames().forEach(tableName -> logger.info("Table: " +
tableName));
        }
    }
}
```

Etapa 4: compilar e executar o aplicativo

Depois que o projeto for criado e contiver a classe `Handler` completa, crie e execute a aplicação.

1. Garanta que você tenha uma sessão ativa do AWS IAM Identity Center. Para confirmar, execute o comando `aws sts get-caller-identity` da AWS Command Line Interface (AWS CLI) e verifique a resposta. Se você não tiver uma sessão ativa, consulte Sign in using the AWS CLI para obter instruções.
2. Abra uma janela de terminal ou prompt de comando e navegue até o diretório `getstarted` do seu projeto.
3. Para compilar seu projeto, execute o seguinte comando:

   ```
   mvn clean package
   ```

4. Para executar a aplicação, execute o seguinte comando:

   ```
   mvn exec:java -Dexec.mainClass="org.example.App"
   ```

Depois de visualizar o arquivo, exclua o objeto e, em seguida, exclua o bucket.

Bem-sucedida

Se seu projeto Maven foi criado e executado sem erros, parabéns! Você compilou com sucesso sua primeira aplicação Java usando o SDK para Java 2.x.

Limpeza

Para limpar os recursos que foram criados durante este tutorial, exclua a pasta de projeto `getstarted`.

Analisar a documentação do AWS SDK for Java 2.x

O Guia do desenvolvedor do AWS SDK for Java 2.x abrange todos os aspectos do SDK em todos os Serviços da AWS. Recomendamos que você analise os seguintes tópicos:

- Migrate from version 1.x to 2.x: inclui uma explicação detalhada das diferenças entre as versões 1.x e 2.x. Esse tópico também contém instruções sobre como usar as duas versões principais lado a lado.
- DynamoDB guide for Java 2.x SDK: mostra como realizar operações básicas do DynamoDB, incluindo criação de tabelas e tratamento e recuperação de itens. Esses exemplos usam a interface de nível inferior. O Java tem várias interfaces, conforme explicado na seguinte seção: Interfaces compatíveis.

> **Tip**
>
> Depois de analisar esses tópicos, adicione a Referência de API do AWS SDK for Java 2.x aos favoritos. Ela abrange todos os Serviços da AWS e recomendamos que você utilize-a como principal referência de API.

Interfaces compatíveis

O AWS SDK for Java 2.x oferece suporte às interfaces a seguir, dependendo do nível de abstração desejado.

Tópicos nesta seção

- Interface de nível inferior
- Interfaces de alto nível
- Interface Document
- Comparar interfaces com um exemplo de Query

Interface de nível inferior

A interface de nível inferior fornece um mapeamento individual para a API de serviço subjacente. Cada API do DynamoDB está disponível por meio dessa interface. Isso significa que a interface de baixo nível pode fornecer funcionalidade completa, mas geralmente é mais detalhada e complexa de usar. Por exemplo, você precisa usar as funções `.s()` para armazenar strings e as funções `.n()` para armazenar números. O exemplo a seguir de PutItem insere um item usando a interface de nível inferior.

```
import org.slf4j.*;
import software.amazon.awssdk.http.crt.AwsCrtHttpClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class PutItem {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT = DynamoDbClient.create();
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemResponse response = DYNAMODB_CLIENT.putItem(PutItemRequest.builder()
                .item(Map.of(
                        "pk", AttributeValue.builder().s("123").build(),
                        "sk", AttributeValue.builder().s("cart#123").build(),
                        "item_data",
 AttributeValue.builder().s("YourItemData").build(),
                        "inventory", AttributeValue.builder().n("500").build()
                        // ... more attributes ...
                ))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .tableName("YourTableName")
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

Interfaces de alto nível

A interface de alto nível no AWS SDK for Java 2.x é chamada de cliente aprimorado do DynamoDB. Essa interface oferece uma experiência de criação de código mais idiomática.

O cliente aprimorado oferece uma forma de associar classes de dados do lado do cliente e tabelas do DynamoDB projetadas para armazenar esses dados. Basta definir as relações entre as tabelas e suas classes de modelo correspondentes no seu código. Depois, você pode recorrer ao SDK para gerenciar o tratamento do tipo de dados. Para obter mais informações sobre o cliente aprimorado, consulte DynamoDB enhanced client API no Guia do desenvolvedor do AWS SDK for Java 2.x.

O exemplo a seguir de PutItem usa a interface de alto nível. Neste exemplo, o `DynamoDbBean` denominado `YourItem` cria um `TableSchema` que permite seu uso direto como entrada para a chamada `putItem()`.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName", TableSchema.fromBean(YourItem.class));
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourItem.class)
                .item(new YourItem("123", "cart#123", "YourItemData", 500))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}
```

```
        public YourItem(String pk, String sk, String itemData, int inventory) {
            this.pk = pk;
            this.sk = sk;
            this.itemData = itemData;
            this.inventory = inventory;
        }

        private String pk;
        private String sk;
        private String itemData;

        private int inventory;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setItemData(String itemData) {
            this.itemData = itemData;
        }

        public String getItemData() {
            return itemData;
        }

        public void setInventory(int inventory) {
            this.inventory = inventory;
        }
```

```
        public int getInventory() {
            return inventory;
        }
    }
}
```

O AWS SDK para Java 1.x tem sua própria interface de alto nível, que geralmente é chamada pela sua classe principal `DynamoDBMapper`. O AWS SDK for Java 2.x é publicado em um pacote separado (e artefato do Maven) chamado `software.amazon.awssdk.enhanced.dynamodb`. O SDK do Java 2.x geralmente é chamado por sua classe principal `DynamoDbEnhancedClient`.

Interface de alto nível usando classes de dados imutáveis

O recurso de mapeamento da API do cliente aprimorado do DynamoDB funciona com classes de dados imutáveis. Uma classe imutável tem apenas getters e requer uma classe construtora que o SDK usa para criar instâncias da classe. A imutabilidade em Java é um estilo comumente utilizado que os desenvolvedores podem usar para criar classes sem efeitos secundários. Essas classes apresentam comportamento mais previsível em aplicações multithread complexas. Em vez de usar a anotação `@DynamoDbBean` conforme mostrado na High-level interface example, as classes imutáveis usam a anotação `@DynamoDbImmutable`, que utiliza a classe de construtor como sua entrada.

O exemplo a seguir usa a classe do construtor `DynamoDbEnhancedClientImmutablePutItem` como entrada para criar um esquema de tabela. Depois, o exemplo fornece o esquema como entrada para a chamada de API PutItem.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutablePutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutablePutItem.class);

    private void putItem() {
```

```
        PutItemEnhancedResponse<YourImmutableItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableItem.class)
                .item(YourImmutableItem.builder()
                                       .pk("123")
                                       .sk("cart#123")
                                       .itemData("YourItemData")
                                       .inventory(500)
                                       .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

O exemplo a seguir mostra a classe de dados imutáveis.

```
@DynamoDbImmutable(builder = YourImmutableItem.YourImmutableItemBuilder.class)
class YourImmutableItem {
    private final String pk;
    private final String sk;
    private final String itemData;
    private final int inventory;
    public YourImmutableItem(YourImmutableItemBuilder builder) {
        this.pk = builder.pk;
        this.sk = builder.sk;
        this.itemData = builder.itemData;
        this.inventory = builder.inventory;
    }

    public static YourImmutableItemBuilder builder() { return new
 YourImmutableItemBuilder(); }

    @DynamoDbPartitionKey
    public String getPk() {
        return pk;
    }

    @DynamoDbSortKey
    public String getSk() {
        return sk;
    }
```

```
    public String getItemData() {
        return itemData;
    }

    public int getInventory() {
        return inventory;
    }

    static final class YourImmutableItemBuilder {
        private String pk;
        private String sk;
        private String itemData;
        private int inventory;

        private YourImmutableItemBuilder() {}

        public YourImmutableItemBuilder pk(String pk) { this.pk = pk; return this; }
        public YourImmutableItemBuilder sk(String sk) { this.sk = sk; return this; }
        public YourImmutableItemBuilder itemData(String itemData) { this.itemData =
 itemData; return this; }
        public YourImmutableItemBuilder inventory(int inventory) { this.inventory =
 inventory; return this; }

        public YourImmutableItem build() { return new YourImmutableItem(this); }
    }
}
```

Interface de alto nível usando classes de dados imutáveis e bibliotecas de geração de código clichê de terceiros

As classes de dados imutáveis (mostradas no exemplo anterior) exigem código clichê. Por exemplo, as lógicas getter e setter nas classes de dados, além das classes `Builder`. Bibliotecas de terceiros, como Project Lombok, podem ajudar você a gerar esse tipo de código clichê. Reduzir a maior parte do código clichê ajuda a limitar a quantidade de código necessária para trabalhar com classes de dados imutáveis e com o SDK da AWS. Isso ocasiona ainda maior produtividade e legibilidade do código. Para obter mais informações, consulte Use third-party libraries, such as Lombok no Guia do desenvolvedor do AWS SDK for Java 2.x.

O exemplo a seguir demonstra como o Project Lombok simplifica o código necessário para usar a API do cliente aprimorado do DynamoDB.

```
import org.slf4j.*;
```

```
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutableLombokPutItem {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableLombokItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableLombokItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutableLombokPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableLombokItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableLombokItem.clas
                .item(YourImmutableLombokItem.builder()
                        .pk("123")
                        .sk("cart#123")
                        .itemData("YourItemData")
                        .inventory(500)
                        .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

O exemplo a seguir mostra o objeto de dados imutáveis da classe de dados imutáveis.

```
import lombok.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;

@Builder
@DynamoDbImmutable(builder =
 YourImmutableLombokItem.YourImmutableLombokItemBuilder.class)
@Value
public class YourImmutableLombokItem {

    @Getter(onMethod_=@DynamoDbPartitionKey)
    String pk;
```

```
    @Getter(onMethod_=@DynamoDbSortKey)
    String sk;
    String itemData;
    int inventory;
}
```

A classe `YourImmutableLombokItem` usa as seguintes anotações fornecidas pelo Project Lombok e pelo SDK da AWS:

- @Builder: produz APIs de construtor complexas para classes de dados fornecidas pelo Project Lombok.
- @DynamoDbImmutable: identifica a classe `DynamoDbImmutable` como uma anotação de entidade mapeável do DynamoDB fornecida pelo SDK da AWS.
- @Value: a variante imutável de `@Data`. Todos os campos são transformados em privados e definitivos por padrão, e os setters não são gerados. O Projeto Lombok fornece essa anotação.

Interface Document

A interface Document do AWS SDK for Java 2.x evita a necessidade de especificar descritores de tipo de dados. Os tipos de dados estão implícitos pela semântica dos próprios dados. Essa interface Document é semelhante à interface Document do AWS SDK para Java 1.x, mas foi reformulada.

O Document interface example a seguir mostra a chamada `PutItem` expressa usando a interface Document. O exemplo também usa EnhancedDocument. Para executar comandos em uma tabela do DynamoDB usando a API de documentos aprimorada, primeiro é necessário associar a tabela ao esquema da tabela de documentos para criar um objeto de recurso `DynamoDBTable`. O construtor de esquema de tabela Document requer uma chave de índice primária e provedores de conversão de atributos.

É possível usar `AttributeConverterProvider.defaultProvider()` para converter atributos de documentos de tipos padrão. Você pode alterar o comportamento padrão geral com uma implementação `AttributeConverterProvider` personalizada. Você também pode alterar o conversor para um único atributo. O Guia de referência de SDKs e ferramentas da AWS fornece mais detalhes e exemplos sobre como usar conversores personalizados. O uso principal é para atributos de suas classes de domínio que não têm um conversor padrão disponível. Usando um conversor personalizado, é possível fornecer ao SDK as informações necessárias para gravação ou leitura no DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedDocumentClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()

 .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk", AttributeValueType.S)
                            .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                            .build());

    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<EnhancedDocument> response =
 DYNAMODB_TABLE.putItemWithResponse(
                        PutItemEnhancedRequest.builder(EnhancedDocument.class)
                                .item(
                                    EnhancedDocument.builder()

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                                            .putString("pk", "123")
                                            .putString("sk", "cart#123")
                                            .putString("item_data", "YourItemData")
                                            .putNumber("inventory", 500)
                                            .build())
                                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

}
```

Para converter documentos JSON de/em tipos de dados nativos do Amazon DynamoDB, você pode usar os seguintes métodos utilitários:

- `EnhancedDocument.fromJson(String json)`: cria uma instância EnhancedDocument com base em uma string JSON.
- `EnhancedDocument.toJson()`: cria uma representação de string JSON do documento que pode ser usada na aplicação como qualquer outro objeto JSON.

Comparar interfaces com um exemplo de **`Query`**

Esta seção mostra a mesma chamada `Query` expressa usando as várias interfaces. Para ajustar os resultados dessas consultas, observe o seguinte:

- O DynamoDB aponta para um valor específico de chave de partição, portanto você deve especificar a chave de partição inteira.
- Para que a consulta aponte apenas para os itens do carrinho, a chave de classificação tem uma expressão de condição de chave que usa `begins_with`.
- Usamos `limit()` para limitar a consulta a um máximo de cem itens devolvidos.
- Definimos o `scanIndexForward` como falso. Os resultados são exibidos na ordem de bytes UTF-8, o que geralmente significa que o item de carrinho com o menor número é exibido primeiro. Ao definir `scanIndexForward` como falso, revertemos a ordem, e o item do carrinho com o maior número é exibido primeiro.
- Aplicamos um filtro para remover qualquer resultado que não corresponda aos critérios. Os dados que estão sendo filtrados consomem capacidade de leitura, independentemente de o item corresponder ou não ao filtro.

Example **`Query`** usando a interface de baixo nível

O exemplo a seguir consulta uma tabela chamada `YourTableName` usando um `keyConditionExpression`. Isso limita a consulta a um valor de chave de partição específico e a um valor de chave de classificação que começa com um valor de prefixo específico. Essas condições de chave limitam a quantidade de dados lidos do DynamoDB. Por fim, a consulta aplica um filtro aos dados recuperados do DynamoDB usando uma `filterExpression`.

```
import org.slf4j.*;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;
```

```
import java.util.Map;

public class Query {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT =
 DynamoDbClient.builder().build();
    private static final Logger LOGGER = LoggerFactory.getLogger(Query.class);

    private static void query() {
        QueryResponse response = DYNAMODB_CLIENT.query(QueryRequest.builder()
                .expressionAttributeNames(Map.of("#name", "name"))
                .expressionAttributeValues(Map.of(
                    ":pk_val", AttributeValue.fromS("id#1"),
                    ":sk_val", AttributeValue.fromS("cart#"),
                    ":name_val", AttributeValue.fromS("SomeName")))
                .filterExpression("#name = :name_val")
                .keyConditionExpression("pk = :pk_val AND begins_with(sk, :sk_val)")
                .limit(100)
                .scanIndexForward(false)
                .tableName("YourTableName")
                .build());

        LOGGER.info("nr of items: " + response.count());
        LOGGER.info("First item pk: " + response.items().get(0).get("pk"));
        LOGGER.info("First item sk: " + response.items().get(0).get("sk"));
    }
}
```

Example **`Query`** usando a interface Document

O exemplo a seguir consulta uma tabela chamada `YourTableName` usando a interface Document.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;

import java.util.Map;

public class DynamoDbEnhancedDocumentClientQuery {
```

```
    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()
                    .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk",
 AttributeValueType.S)
                    .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                    .build());
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientQuery.class);

    private void query() {
        PageIterable<EnhancedDocument> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " +
 response.items().iterator().next().getString("pk"));
        LOGGER.info("First item sk: " +
 response.items().iterator().next().getString("sk"));

    }
}
```

Example **`Query`** usando a interface de alto nível

O exemplo a seguir consulta uma tabela chamada `YourTableName` usando a API do cliente aprimorado do DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import java.util.Map;

public class DynamoDbEnhancedClientQuery {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromBean(DynamoDbEnhancedClientQuery.YourItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientQuery.class);

    private void query() {
        PageIterable<YourItem> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " + response.items().iterator().next().getPk());
        LOGGER.info("First item sk: " + response.items().iterator().next().getSk());
    }
```

```
    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String name) {
            this.pk = pk;
            this.sk = sk;
            this.name = name;
        }

        private String pk;
        private String sk;
        private String name;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setName(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }
    }
}
```

Interface de alto nível usando classes de dados imutáveis

Quando você executa uma `Query` com as classes de dados imutáveis de alto nível, o código é o mesmo do exemplo de interface de alto nível, exceto pela construção da classe de entidade `YourItem` ou `YourImmutableItem`. Para ter mais informações, consulte o exemplo PutItem.

Interface de alto nível usando classes de dados imutáveis e bibliotecas de geração de código clichê de terceiros

Quando você executa uma `Query` com as classes de dados imutáveis de alto nível, o código é o mesmo do exemplo de interface de alto nível, exceto pela construção da classe de entidade `YourItem` ou `YourImmutableLombokItem`. Para ter mais informações, consulte o exemplo PutItem.

Exemplos de código adicionais

Para conferir exemplos adicionais de como usar o DynamoDB com o SDK para Java 2.x, consulte os seguintes repositórios de exemplos de código:

- Exemplos oficiais de código de ação única da AWS
- Exemplos de códigos de ação única mantidos pela comunidade
- Exemplos oficiais de código orientado por cenários da AWS

Programação síncrona e assíncrona

O AWS SDK for Java 2.x fornece clientes síncronos e assíncronos para Serviços da AWS, como o DynamoDB.

As classes `DynamoDbClient` e `DynamoDbEnhancedClient` fornecem métodos síncronos que bloqueiam a execução do seu thread até o cliente receber uma resposta do serviço. Esse cliente é a maneira mais simples de interagir com o DynamoDB se você não precisar de operações assíncronas.

As classes `DynamoDbAsyncClient` e `DynamoDbEnhancedAsyncClient` fornecem métodos assíncronos que são exibidos imediatamente e devolvem o controle ao thread de chamada sem aguardar uma resposta. O cliente sem bloqueio tem a vantagem de permitir alta simultaneidade em alguns segmentos, o que oferece tratamento eficiente de solicitações de E/S com recursos computacionais mínimos. Isso melhora o throughput e a capacidade de resposta.

O AWS SDK for Java 2.x usa o suporte nativo para E/S sem bloqueio. O AWS SDK para Java 1.x precisou simular E/S sem bloqueio.

Os métodos síncronos são exibidos antes que uma resposta esteja disponível, portanto você precisará de uma maneira de receber a resposta quando ela estiver pronta. Os métodos assíncronos no AWS SDK para Java exibem um objeto `CompletableFuture` que contém os resultados da operação assíncrona no futuro. Ao chamar `get()` ou `join()` nesses objetos `CompletableFuture`, o código fica bloqueado até que o resultado esteja disponível. Se você fizer essas duas chamadas ao mesmo tempo em que faz a solicitação, o comportamento será semelhante a uma chamada síncrona simples.

Para obter mais informações sobre programação assíncrona, consulte Use asynchronous programming no Guia do desenvolvedor do AWS SDK for Java 2.x.

Clientes HTTP

Para comportar cada cliente, existe um cliente HTTP que lida com a comunicação com os Serviços da AWS. É possível conectar clientes HTTP alternativos, escolhendo um que tenha as características mais adequadas à aplicação. Alguns são mais leves; alguns têm mais opções de configuração.

Alguns clientes HTTP comportam somente o uso síncrono, enquanto outros aceitam apenas o uso assíncrono. Para conferir um fluxograma que pode ajudar você a selecionar o cliente HTTP ideal para sua workload, consulte HTTP client recommendations no Guia do desenvolvedor do AWS SDK for Java 2.x.

A lista a seguir apresenta alguns dos possíveis clientes HTTP:

Tópicos

- Cliente HTTP baseado em Apache
- Cliente HTTP baseado em URLConnection
- Cliente HTTP baseado em Netty
- Cliente HTTP baseado em AWS CRT

Cliente HTTP baseado em Apache

A classe `ApacheHttpClient` oferece suporte a clientes de serviço síncronos. É o cliente HTTP padrão para uso síncrono. Para obter informações sobre como configurar a classe

`ApacheHttpClient`, consulte Configure the Apache-based HTTP client no Guia do desenvolvedor do AWS SDK for Java 2.x.

Cliente HTTP baseado em **`URLConnection`**

A classe `UrlConnectionHttpClient` é outra opção para clientes síncronos. Ela é carregada mais rapidamente do que o cliente HTTP baseado em Apache, mas tem menos recursos. Para obter informações sobre como configurar a classe `UrlConnectionHttpClient`, consulte Configure the URLConnection-based HTTP client no Guia do desenvolvedor do AWS SDK for Java 2.x.

Cliente HTTP baseado em Netty

A classe `NettyNioAsyncHttpClient` oferece suporte a clientes assíncronos. É a opção padrão para uso assíncrono. Para obter informações sobre como configurar a classe `NettyNioAsyncHttpClient`, consulte Configure the Netty-based HTTP client no Guia do desenvolvedor do AWS SDK for Java 2.x.

Cliente HTTP baseado em AWS CRT

As classes `AwsCrtHttpClient` e `AwsCrtAsyncHttpClient` mais recentes das bibliotecas AWS Common Runtime (CRT) são outra opção compatível com clientes síncronos e assíncronos. Em comparação com outros clientes HTTP, o AWS CRT oferece:

- Menor tempo de inicialização do SDK
- Menor espaço ocupado na memória
- Tempo de latência reduzido
- Gerenciamento de integridade da conexão
- balanceamento de carga do DNS

Para obter informações sobre como configurar as classes `AwsCrtHttpClient` e `AwsCrtAsyncHttpClient`, consulte Configure the AWS CRT-based HTTP clients no Guia do desenvolvedor do AWS SDK for Java 2.x.

O cliente HTTP baseado em AWS CRT não é o padrão porque isso romperia a compatibilidade com versões anteriores das aplicações existentes. No entanto, para o DynamoDB, recomendamos que você use o cliente HTTP baseado em AWS CRT para uso sincronizado e assíncrono.

Para conferir uma introdução sobre o cliente HTTP baseado em AWS CRT, consulte Announcing availability of the AWS CRT HTTP Client in the AWS SDK for Java 2.x no blog de ferramentas do desenvolvedor da AWS.

Configurar um cliente HTTP

Ao configurar um cliente, é possível fornecer várias opções de configuração, incluindo:

- Definir tempos limite para diferentes aspectos das chamadas de API.
- Habilitar keep-alive de TCP.
- Controlar a política de novas tentativas ao encontrar erros.
- Especificar atributos de execução que as instâncias do interceptor de execução podem modificar. Os interceptores de execução podem escrever código que intercepte a execução de suas solicitações e respostas de API. Isso possibilita executar tarefas, como publicar métricas e modificar solicitações em andamento.
- Adicionar ou manipular cabeçalhos HTTP.
- Permitir o rastreamento das métricas de performance do lado do cliente. Usar esse recurso ajuda você a coletar métricas sobre os clientes de serviço em sua aplicação e analisar a saída no Amazon CloudWatch.
- Especificar um serviço executor alternativo a ser usado para agendar tarefas, como novas tentativas assíncronas e tarefas de tempo limite.

Você controla a configuração fornecendo um objeto `ClientOverrideConfiguration` para a classe `Builder` do cliente de serviço. Você verá isso em alguns exemplos de código nas seções a seguir.

A `ClientOverrideConfiguration` fornece opções de configuração padrão. Os diferentes clientes HTTP conectáveis também têm possibilidades de configuração específicas de implementação.

Tópicos nesta seção

- Configuração de tempo limite
- RetryMode
- DefaultsMode
- Configuração de keep-alive

Configuração de tempo limite

É possível ajustar a configuração do cliente para controlar vários tempos limite relacionados às chamadas de serviço. O DynamoDB fornece latências mais baixas em comparação com outros Serviços da AWS. Portanto, talvez você queira ajustar essas propriedades a fim de reduzir os valores de tempo limite para que possa antecipar-se à falha em caso de problema na rede.

É possível personalizar o comportamento relacionado à latência usando `ClientOverrideConfiguration` no cliente do DynamoDB ou alterando as opções de configuração detalhadas na implementação do cliente HTTP subjacente.

É possível configurar as seguintes propriedades impactantes usando `ClientOverrideConfiguration`:

- `apiCallAttemptTimeout`: o tempo de espera até que uma única tentativa de solicitação HTTP seja concluída antes de desistir e atingir o tempo limite.
- `apiCallTimeout`: o tempo que o cliente tem para executar completamente uma chamada de API. Isso inclui a execução do manipulador de solicitações, que consiste em todas as solicitações HTTP, incluindo novas tentativas.

O AWS SDK for Java 2.x fornece valores padrão para algumas opções de tempo limite, como tempo limite de conexão e tempo limite de soquete. O SDK não fornece valores padrão para tempo limite de chamadas de API nem tempos limite de chamadas de API individuais. Se esses tempos limite não estiverem definidos em `ClientOverrideConfiguration`, o SDK usará o valor do tempo limite do soquete para o tempo limite geral da chamada de API. O tempo limite do soquete tem um valor padrão de trinta minutos.

RetryMode

Outra configuração relacionada à configuração de tempo limite que deve ser considerada é o objeto de configuração `RetryMode`. Esse objeto de configuração contém uma coleção de comportamentos de novas tentativas.

O SDK para Java 2.x é compatível com os seguintes modos de novas tentativas:

- `legacy`: o modo de novas tentativas padrão, se você não o alterar explicitamente. Esse modo de novas tentativas é específico do SDK para Java. Ele é caracterizado por até três novas tentativas, ou mais para serviços, como o DynamoDB que tem até oito novas tentativas.

- `standard`: é chamado de “standard” porque é mais consistente com outros SDKs da AWS. Esse modo espera por um período aleatório que varia de 0 ms a 1.000 ms pela primeira tentativa. Se outra tentativa for necessária, esse modo escolherá outro tempo aleatório entre 0 ms e 1.000 ms, e o multiplicará por dois. Se for necessária uma nova tentativa, ele fará a mesma escolha aleatória multiplicada por quatro e assim por diante. Cada espera é limitada a 20 segundos. Esse modo executa novas tentativas em mais condições de falha detectadas do que o modo `legacy`. Para o DynamoDB, ele executa até três tentativas no total, a menos que você substitua por numRetries.
- `adaptive`: baseia-se no modo `standard` e limita dinamicamente a taxa de solicitações da AWS para maximizar a taxa de êxito. Isso pode ocorrer em detrimento da latência da solicitação. Não recomendamos o modo de novas tentativas adaptável quando a latência previsível for importante.

É possível encontrar uma definição expandida desses modos de novas tentativas no tópico Retry behavior no Guia de referência de SDKs e ferramentas da AWS.

Política de novas tentativas

Todas as configurações de `RetryMode` têm uma `RetryPolicy`, que é criada com base em uma ou mais configurações de `RetryCondition`. `TokenBucketRetryCondition` é especialmente importante para o comportamento de novas tentativas da implementação do cliente do SDK do DynamoDB. Essa condição limita o número de novas tentativas que o SDK faz usando um algoritmo de bucket de token. Dependendo do modo de novas tentativas selecionado, as exceções de controle de utilização podem ou não subtrair tokens do `TokenBucket`.

Quando um cliente encontra um erro que pode ser repetido, como uma exceção de controle de utilização ou um erro temporário do servidor, o SDK repete automaticamente a solicitação. É possível controlar quantas vezes e com que rapidez essas novas tentativas acontecem.

Ao configurar um cliente, é possível fornecer uma `RetryPolicy` compatível com os seguintes parâmetros:

- `numRetries`: o número máximo de novas tentativas que devem ser aplicadas antes que uma solicitação seja considerada com falha. O valor padrão é 8, independentemente do modo de novas tentativas usado.

> **Warning**
>
> Certifique-se de alterar esse valor padrão após a devida consideração.

- `backoffStrategy`: a `BackoffStrategy` a ser aplicada às novas tentativas, em que `FullJitterBackoffStrategy` é a estratégia padrão. Essa estratégia realiza um atraso exponencial entre novas tentativas com base no número ou nas novas tentativas atuais, um atraso base e um tempo máximo de recuo. Depois, adiciona instabilidade para fornecer um pouco de aleatoriedade. O atraso básico usado no atraso exponencial é de 25 ms, independentemente do modo de novas tentativas.
- `retryCondition`: a `RetryCondition` determina se uma solicitação deve ou não ser repetida. Por padrão, ela tenta novamente um conjunto específico de códigos de status HTTP e exceções que acredita serem passíveis de nova tentativa. Para a maioria das situações, a configuração padrão deve ser suficiente.

O código a seguir fornece uma política de novas tentativas alternativa. Ele especifica um total de cinco novas tentativas (seis solicitações no total). A primeira nova tentativa deve ocorrer após um atraso de aproximadamente 100 ms, com cada nova tentativa adicional dobrando esse tempo exponencialmente, até, no máximo, um atraso de 1 segundo.

```
DynamoDbClient client = DynamoDbClient.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .retryPolicy(RetryPolicy.builder()
            .backoffStrategy(FullJitterBackoffStrategy.builder()
                .baseDelay(Duration.ofMillis(100))
                .maxBackoffTime(Duration.ofSeconds(1))
                .build())
            .numRetries(5)
            .build())
        .build())
    .build();
```

DefaultsMode

As propriedades de tempo limite que `ClientOverrideConfiguration` e `RetryMode` não gerenciam são normalmente configuradas implicitamente especificando um `DefaultsMode`.

O AWS SDK for Java 2.x (versão 2.17.102 ou posterior) introduziu suporte a `DefaultsMode`. Esse recurso oferece um conjunto de valores padrão para configurações comuns, como configurações de comunicação HTTP, comportamento de novas tentativas, configurações de endpoint regional do serviço e, potencialmente, qualquer configuração relacionada ao SDK. Os clientes que usam esse recurso podem receber novos padrões de configuração personalizados para cenários de uso comuns.

Os modos padrão são padronizados em todos os SDKs da AWS. O SDK para Java 2.x é compatível com os seguintes modos padrão:

- `legacy`: fornece configurações padrão que variam de acordo com o SDK da AWS e que existiam antes do estabelecimento de `DefaultsMode`.
- `standard`: fornece configurações padrão não otimizadas para a maioria dos cenários.
- `in-region`: baseia-se no modo padrão e inclui configurações personalizadas para aplicações que chamam Serviços da AWS de dentro da mesma Região da AWS.
- `cross-region`: baseia-se no modo padrão e inclui configurações com altos tempos limite para aplicações que chamam Serviços da AWS em uma região diferente.
- `mobile`: baseia-se no modo padrão e inclui configurações com tempo limite alto, personalizadas para aplicações para dispositivos móveis com latências mais altas.
- `auto`: baseia-se no modo padrão e inclui atributos experimentais. O SDK tenta descobrir o ambiente de runtime para determinar automaticamente as configurações apropriadas. A detecção automática é baseada em heurísticas e não fornece 100% de precisão. Se o ambiente de runtime não puder ser determinado, o modo padrão será usado. A detecção automática pode consultar metadados da instância e dados do usuário, o que pode introduzir latência. Se a latência de inicialização for fundamental para seu aplicativo, recomendamos escolher um `DefaultsMode` explícito.

É possível configurar o modo padrão das seguintes maneiras:

- Diretamente em um cliente por meio de `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`.
- Em um perfil de configuração por meio da propriedade do arquivo de perfil `defaults_mode`.
- Globalmente por meio da propriedade do sistema `aws.defaultsMode`.
- Globalmente por meio da variável de ambiente `AWS_DEFAULTS_MODE`.

> **Note**
>
> Para qualquer modo que não seja `legacy`, os valores padrão fornecidos podem mudar à medida que as práticas recomendadas evoluem. Portanto, se estiver usando um modo diferente de `legacy`, recomendamos que você realize testes ao atualizar o SDK.

A seção Smart configuration defaults no Guia de referência de SDKs e ferramentas da AWS fornece uma lista de propriedades de configuração e seus valores padrão nos diferentes modos padrão.

Escolha o valor de modo padrão com base nas características da aplicação e no AWS service (Serviço da AWS) com o qual a aplicação interage.

Esses valores são configurados com uma ampla seleção de Serviços da AWS em mente. Para uma implantação típica do DynamoDB em que a aplicação e as tabelas do DynamoDB são implantadas na mesma região, o modo `in-region` padrão é mais relevante entre os modos padrão `standard`.

Example Configuração do cliente do SDK do DynamoDB ajustada para chamadas de baixa latência

O exemplo a seguir ajusta os tempos limite para valores mais baixos para uma chamada esperada do DynamoDB de baixa latência.

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

A implementação do cliente HTTP individual pode fornecer a você um controle ainda mais granular sobre o tempo limite e o comportamento de uso da conexão. Por exemplo, para o cliente baseado em AWS CRT, é possível habilitar `ConnectionHealthConfiguration`, que permite que o cliente monitore ativamente a integridade das conexões usadas. Para obter mais informações, consulte Advanced configuration of AWS CRT-based HTTP clients no Guia do desenvolvedor do AWS SDK for Java 2.x.

Configuração de keep-alive

Habilitar o keep-alive pode reduzir as latências ao reutilizar conexões. Há dois tipos diferentes de keep-alive: HTTP Keep-Alive e TCP Keep-Alive.

- O HTTP Keep-Alive tenta manter a conexão HTTPS entre o cliente e o servidor para que solicitações posteriores possam reutilizar essa conexão. Isso ignora a pesada autenticação HTTPS em solicitações posteriores. O HTTP Keep-Alive está habilitado por padrão em todos os clientes.
- O keep-alive de TCP solicita que o sistema operacional subjacente envie pequenos pacotes pela conexão do soquete a fim de fornecer garantia extra de que o soquete seja mantido ativo e para

detectar imediatamente qualquer queda. Isso garante que não haja perda de tempo em uma solicitação posterior com a tentativa de usar um soquete descartado. Por padrão, o keep-alive de TCP está desabilitado em todos os clientes. Os exemplos de código a seguir mostram como habilitar essa opção em cada cliente HTTP. Quando habilitado para todos os clientes HTTP não baseados em CRT, o mecanismo real de keep-alive depende do sistema operacional. Portanto, é necessário configurar valores adicionais de TCP Keep-Alive, como tempo limite e número de pacotes, por meio do sistema operacional. É possível fazer isso usando `sysctl` no Linux ou macOS, ou usando valores de registro no Windows.

Example para habilitar o TCP Keep-Alive em um cliente HTTP baseado em Apache

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Cliente HTTP baseado em **`URLConnection`**

Nenhum cliente síncrono que usa o cliente HTTP baseado em `URLConnection` `HttpURLConnection` tem um mecanismo para habilitar o keep-alive.

Example para habilitar o TCP Keep-Alive em um cliente HTTP baseado em Apache

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example para habilitar o TCP Keep-Alive em um cliente HTTP baseado em AWS CRT

Com o cliente HTTP baseado em AWS CRT, é possível habilitar o TCP Keep-Alive e controlar a duração.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Ao usar o cliente assíncrono do DynamoDB, é possível habilitar o TCP Keep-Alive conforme mostrado no código a seguir.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Gerenciamento de erros

No que se refere ao tratamento de exceções, o AWS SDK for Java 2.x usa exceções de tempo de execução (não conferidas).

A exceção básica, abrangendo todas as exceções do SDK, é `SdkServiceException`, que se estende da `RuntimeException` não conferida do Java. Se você captar isso, captará todas as exceções geradas pelo SDK.

`SdkServiceException` tem uma subclasse chamada `AwsServiceException`. Essa subclasse indica qualquer problema na comunicação com o AWS service (Serviço da AWS). Ela tem uma subclasse chamada `DynamoDbException`, que indica um problema na comunicação com o DynamoDB. Se você captar isso, captará todas as exceções relacionadas ao DynamoDB, mas nenhuma outra exceção do SDK.

Há tipos de exceção mais específicos em `DynamoDbException`. Alguns desses tipos de exceção se aplicam às operações do ambiente de gerenciamento, como `TableAlreadyExistsException`. Outros se aplicam às operações do plano de dados. Veja a seguir um exemplo de exceção comum do plano de dados:

- `ConditionalCheckFailedException`: você especificou uma condição na solicitação que foi avaliada como falsa. Por exemplo, você pode ter tentado realizar uma atualização condicional em um item, mas o valor real do atributo não correspondeu ao valor esperado na condição. Uma solicitação que falhe dessa maneira não será repetida.

Outras situações não têm uma exceção específica definida. Por exemplo, quando suas solicitações têm controle de utilização, a `ProvisionedThroughputExceededException` específica pode ser gerada, enquanto, em outros casos, `DynamoDbException`mais genérica é lançada. Nos dois

casos, é possível determinar se a exceção foi causada pelo controle de utilização, conferindo se `isThrottlingException()` retorna `true`.

Dependendo das necessidades da aplicação, é possível capturar todas as instâncias `AwsServiceException` ou `DynamoDbException`. No entanto, muitas vezes você precisa de um comportamento diferente em outras situações. A lógica para lidar com uma falha na verificação de condições é diferente quando se trata de lidar com o controle de utilização. Defina com quais caminhos excepcionais você deseja lidar e não se esqueça de testar os caminhos alternativos. Isso ajuda a garantir que você seja capaz de lidar com todos os cenários relevantes.

Para obter uma lista de erros comuns que você pode encontrar, consulte Tratamento de erros com o DynamoDB. Consulte também Common Errors na Referência de API do Amazon DynamoDB. A Referência de API também apresenta os erros exatos possíveis para cada operação de API, como a operação `Query`. Para obter informações sobre como lidar com exceções, consulte Exception handling for the AWS SDK for Java 2.x no Guia do desenvolvedor do AWS SDK for Java 2.x.

ID da solicitação da AWS

Cada solicitação inclui um ID de solicitação cuja extração pode ser útil se você estiver trabalhando com AWS Support para diagnosticar um problema. Cada exceção derivada de `SdkServiceException` tem um método `requestId()` disponível para recuperar o ID da solicitação.

Registro em log

Usar o registro em log fornecido pelo SDK pode ser útil tanto para capturar mensagens importantes das bibliotecas do cliente quanto para fins de depuração mais detalhada. Os loggers são hierárquicos e o SDK usa `software.amazon.awssdk` como logger raiz. Você pode configurar o nível com um dos seguintes: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL` ou `OFF`. O nível configurado é aplicado a esse logger e à hierarquia de loggers.

Para o registro em log, o AWS SDK for Java 2.x usa o Simple Logging Façade for Java (SLF4J). Isso atua como uma camada de abstração em torno de outros loggers, que pode ser usada para conectar o logger de sua preferência. Para ter instruções sobre como conectar loggers, consulte o Manual do usuário do SLF4J.

Cada logger tem um comportamento específico. Por padrão, o logger Log4j 2.x cria um `ConsoleAppender`, que anexa eventos de log a `System.out` e assume o nível de log `ERROR` como padrão.

O logger SimpleLogger incluído no SLF4J, por padrão, emite por padrão `System.err` e utiliza como padrão o nível de log `INFO`.

Recomendamos definir o nível como `WARN` para `software.amazon.awssdk` para que todas as implantações em produção capturem todas as mensagens importantes das bibliotecas de cliente do SDK e limitem a quantidade de saída.

Se o SLF4J não conseguir encontrar um logger compatível no caminho da classe (sem vinculação ao SLF4J), ele assumirá como padrão uma implementação sem operação. Essa implementação ocasiona o registro em log de mensagens em `System.err` explicando que o SLF4J não conseguiu encontrar uma implementação de logger no caminho de classe. Para evitar essa situação, é necessário adicionar uma implementação de logger. Para fazer isso, é possível incluir uma dependência no Apache Maven `pom.xml` em artefatos, como `org.slf4j.slf4j-simple` ou `org.apache.logging.log4j.log4j-slf4j2-imp`.

Para obter informações sobre como configurar o registro em log no SDK, incluindo a adição de dependências de registro em log à configuração da aplicação, consulte Logging with the SDK for Java 2.x no Guia do desenvolvedor do AWS SDK para Java.

A configuração a seguir no arquivo `Log4j2.xml` mostra como ajustar o comportamento de registro em log se você usar o logger Apache Log4j 2. Essa configuração define o nível do logger raiz como `WARN`. Todos os loggers na hierarquia herdam esse nível de registro, incluindo o logger `software.amazon.awssdk`.

Por padrão, a saída é enviada para `System.out`. No exemplo a seguir, ainda substituímos o anexador Log4j de saída padrão para aplicar um Log4j personalizado `PatternLayout`.

Exemplo de arquivo de configuração do **`Log4j2.xml`**

A configuração a seguir registra mensagens em log nos níveis `ERROR` e `WARN` no console para todas as hierarquias de logger.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
```

```
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

Registro em log de ID de solicitação da AWS

Quando algo dá errado, você pode encontrar os IDs de solicitação nas exceções. No entanto, se você quiser os IDs das solicitações que não estão gerando exceções, poderá usar o registro em log.

O logger `software.amazon.awssdk.request` exibe os IDs de solicitação no nível `DEBUG`. O exemplo a seguir estende o configuration example anterior para manter o nível do logger raiz em `ERROR`, o `software.amazon.awssdk` em nível `WARN` e o `software.amazon.awssdk.request` em nível `DEBUG`. A definição desses níveis ajuda a capturar os IDs de solicitação e outros detalhes relacionados à solicitação, como o endpoint e o código de status.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="ERROR">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

Aqui está um exemplo da saída do log:

```
2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Sending Request:
 DefaultSdkHttpFullRequest(httpMethod=POST, protocol=https, host=dynamodb.us-
east-1.amazonaws.com, encodedPath=/, headers=[amz-sdk-invocation-id, Content-Length,
 Content-Type, User-Agent, X-Amz-Target], queryParameters=[])
 2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Received
 successful response: 200, Request ID:
```

```
QS9DUMME2NHEDH8TGT9N5V53OJVV4KQNSO5AEMVJF66Q9ASUAAJG, Extended Request ID: not
available
```

Paginação

Algumas solicitações, como `Query` e `Scan`, limitam o tamanho dos dados retornadas em uma única solicitação e exigem que você faça solicitações repetidas para extrair as páginas subsequentes.

É possível controlar o número máximo de itens a serem lidos em cada página com o parâmetro `Limit`. Por exemplo, é possível usar o parâmetro `Limit` para recuperar somente os últimos dez itens. Esse limite especifica quantos itens devem ser lidos da tabela antes da aplicação de qualquer filtragem. Se você quiser exatamente dez itens após a filtragem, não há como especificar isso. Só é possível controlar a contagem pré-filtrada e verificar no lado do cliente quando tiver realmente recuperado dez itens. Independentemente do limite, as respostas sempre têm um tamanho máximo de 1 MB.

Uma `LastEvaluatedKey` pode ser incluída na resposta de API. Isso indica que a resposta terminou porque atingiu um limite de contagem ou um limite de tamanho. Essa chave é a última avaliada para essa resposta. Interagindo diretamente com a API, é possível recuperar essa `LastEvaluatedKey` e transmiti-la para uma chamada de acompanhamento como `ExclusiveStartKey` para ler a próxima parte desse ponto de partida. Se não for retornada nenhuma `LastEvaluatedKey`, significa que não há mais itens que correspondam à chamada de API `Query` ou `Scan`.

O exemplo a seguir usa a interface de nível inferior para limitar os itens a cem com base no parâmetro `keyConditionExpression`.

```
QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
        .expressionAttributeValues(Map.of(
                ":pk_val", AttributeValue.fromS("123"),
                ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName(TABLE_NAME);

while (true) {
    QueryResponse queryResponse = DYNAMODB_CLIENT.query(queryRequestBuilder.build());

    queryResponse.items().forEach(item -> {
        LOGGER.info("item PK: [" + item.get("pk") + "] and SK: [" + item.get("sk") +
 "]");
```

```
    });

    if (!queryResponse.hasLastEvaluatedKey()) {
        break;
    }
    queryRequestBuilder.exclusiveStartKey(queryResponse.lastEvaluatedKey());
}
```

O AWS SDK for Java 2.x pode simplificar essa interação com o DynamoDB, fornecendo métodos de paginação automática que fazem várias chamadas de serviço para obter as próximas páginas de resultados automaticamente. Isso simplifica o código, mas elimina uma parte do controle sobre o uso de recursos que você manteria lendo as páginas manualmente.

Usando os métodos `Iterable` disponíveis no cliente do DynamoDB, como `QueryPaginator` e `ScanPaginator`, o SDK cuida da paginação. O tipo de retorno desses métodos é um iterável personalizado que você pode usar para percorrer todas as páginas. O SDK lida internamente com as chamadas de serviço para você. Usando a API do Java Stream, é possível lidar com o resultado de `QueryPaginator`, conforme mostrado no exemplo a seguir.

```
QueryPublisher queryPublisher =
    DYNAMODB_CLIENT.queryPaginator(QueryRequest.builder()
        .expressionAttributeValues(Map.of(
            ":pk_val", AttributeValue.fromS("123"),
            ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName("YourTableName")
        .build());

queryPublisher.items().subscribe(item ->
    System.out.println(item.get("itemData"))).join();
```

Anotações de classes de dados

O SDK para Java fornece várias anotações que você pode colocar nos atributos da sua classe de dados. Essas anotações influenciam a forma como o SDK interage com os atributos. Ao adicionar uma anotação, é possível fazer com que um atributo se comporte como um contador atômico implícito, mantenha um valor de carimbo de data e hora gerado automaticamente ou rastreie o número da versão de um item. Para ter mais informações, consulte Data class annotations.

Tratamento de erros com o DynamoDB

Esta seção descreve erros de runtime e como lidar com eles. Ela também descreve códigos e mensagens de erro que são específicos para o Amazon DynamoDB. Para obter uma lista de erros comuns que se aplicam a todos os serviços da AWS, consulte Gerenciamento de acesso.

Tópicos

- Componentes de erros
- Erros transacionais
- Mensagens e códigos de erro
- Tratamento de erros na aplicação
- Repetições de erro e recuo exponencial
- Operações em lote e tratamento de erros

Componentes de erros

Quando seu programa envia uma solicitação, o DynamoDB tenta processá-la. Se a solicitação for bem-sucedida, o DynamoDB retornará um código de status HTTP de êxito (`200 OK`), juntamente com os resultados da operação solicitada.

Se a solicitação falhar, o DynamoDB retornará um erro. Cada erro tem três componentes:

- Um código de status HTTP (como `400`).
- Um nome de exceção (como `ResourceNotFoundException`).
- Uma mensagem de erro (como `Requested resource not found: Table: tablename not found`).

Os AWS SDKs cuidam da propagação de erros para sua aplicação para que você possa tomar as medidas apropriadas. Por exemplo, em um programa Java, você pode escrever a lógica `try-catch` para lidar com um `ResourceNotFoundException`.

Se você não estiver usando um AWS SDK, será necessário analisar o conteúdo da resposta de baixo nível do DynamoDB. Veja a seguir um exemplo dessa resposta.

```
HTTP/1.1 400 Bad Request
```

```
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 240
Date: Thu, 15 Mar 2012 23:56:23 GMT

{"__type":"com.amazonaws.dynamodb.v20120810#ResourceNotFoundException",
"message":"Requested resource not found: Table: tablename not found"}
```

Erros transacionais

Para obter informações sobre erros transacionais, consulte Tratamento de conflitos em transações com o DynamoDB

Mensagens e códigos de erro

Veja a seguir uma lista de exceções retornadas pelo DynamoDB, agrupadas por código de status HTTP. Se OK para tentar novamente? for Sim, você poderá enviar a mesma solicitação novamente. Se OK to retry? (OK tentar novamente?) for No (Não), você deverá corrigir o problema no lado do cliente antes de enviar uma nova solicitação.

Código de status HTTP 400

Um código de status HTTP `400` indica um problema com sua solicitação, como falha de autenticação, parâmetros necessários ausentes ou limite excedido do throughput provisionado de uma tabela. Será necessário corrigir o problema no aplicativo antes de enviar a solicitação novamente.

AccessDeniedException

Mensagem: Acesso negado.

The client did not correctly sign the request. Se estiver usando um AWS SDK, as solicitações serão assinadas para você automaticamente. Do contrário, acesse o Processo de assinatura do Signature versão 4 na Referência geral da AWS.

OK para tentar novamente? Não

ConditionalCheckFailedException

Mensagem: A solicitação condicional falhou.

Você especificou uma condição avaliada como false. Por exemplo, você pode ter tentado realizar uma atualização condicional em um item, mas o valor real do atributo não correspondeu ao valor esperado na condição.

OK para tentar novamente? Não

IncompleteSignatureException

Mensagem: A assinatura da solicitação não atende aos padrões da AWS.

A assinatura da solicitação não incluía todos os componentes necessários. Se estiver usando um AWS SDK, as solicitações serão assinadas para você automaticamente. Do contrário, acesse o Processo de assinatura do Signature versão 4 na Referência geral da AWS.

OK para tentar novamente? Não

ItemCollectionSizeLimitExceededException

Mensagem: Tamanho da coleção excedido.

Para uma tabela com um índice secundário local, um grupo de itens com o mesmo valor de chave de partição excedeu o limite de tamanho máximo de 10 GB. Para obter mais informações sobre coleções de itens, consulte Conjuntos de itens em índices secundários locais.

OK para tentar novamente? Sim

LimitExceededException

Mensagem: Muitas operações para um assinante específico.

Existem muitas operações simultâneas no plano de controle. O número cumulativo de tabelas e índices no estado `CREATING`, `DELETING` ou `UPDATING` não pode exceder 500.

OK para tentar novamente? Sim

MissingAuthenticationTokenException

Mensagem: A solicitação deve conter um ID de chave de acesso válido (registrado) na AWS.

A solicitação não incluía o cabeçalho de autorização necessário ou estava mal formada. Consulte API de baixo nível do DynamoDB.

OK para tentar novamente? Não

ProvisionedThroughputExceededException

Mensagem: You exceeded your maximum allowed provisioned throughput for a table or for one or more global secondary indexes. (Você excedeu seu throughput provisionado máximo permitida para uma tabela ou para um ou mais índices secundários globais.) Para visualizar métricas de performance para throughput provisionado versus throughput consumido, abra o console do Amazon CloudWatch.

O erro inclui uma lista de campos `ThrottlingReason` que apresenta um contexto específico sobre o motivo do controle de utilização, seguindo o formato `ResourceType+OperationType+LimitType` (p. ex., `TableReadProvisionedThroughputExceeded`) e o ARN do recurso afetado. Isso ajuda a identificar qual recurso está sofrendo controle de utilização (tabela ou índice), qual tipo de operação acionou o controle (leitura ou gravação) e o limite específico que foi excedido (nesse caso, capacidade provisionada). Para ter mais informações sobre como diagnosticar e resolver problemas de controle de utilização, consulte Diagnosticar o controle de utilização.

Exemplo: Sua taxa de solicitação é muito alta. Os AWS SDKs para o DynamoDB realizam automaticamente novas tentativas de envio de uma mesma solicitação que recebe essa exceção. Sua solicitação em algum momento terá êxito, a menos que a fila de novas tentativas seja muito grande para concluir. Reduza a frequência de solicitações usando Repetições de erro e recuo exponencial.

OK para tentar novamente? Sim

ReplicatedWriteConflictException

Mensagem: um ou mais itens desta solicitação estão sendo modificados por uma solicitação em outra região.

Exemplo: uma operação de gravação foi solicitada para um item em uma tabela global multirregional fortemente consistente (MRSC) que está sendo modificada atualmente por uma solicitação em outra região.

OK para tentar novamente? Sim

RequestLimitExceeded

Mensagem: Throughput exceeds the current throughput limit for your account (o throughput excede o limite de throughput atual de sua conta). Para solicitar um aumento de limite, entre em contato com o AWS Support em https://aws.amazon.com/support.

O erro inclui uma lista de campos `ThrottlingReason` que apresenta um contexto específico sobre o motivo do controle de utilização, seguindo o formato `ResourceType+OperationType+LimitType` (p. ex., `TableWriteAccountLimitExceeded` ou `IndexReadAccountLimitExceeded`) e o ARN do recurso afetado. Isso ajuda você a identificar qual recurso está sofrendo controle de utilização (tabela ou índice), qual tipo de operação acionou o controle (leitura ou gravação) e se você excedeu as cotas de serviço em nível de conta. Para ter mais informações sobre como diagnosticar e resolver problemas de controle de utilização, consulte Diagnosticar o controle de utilização.

Exemplo: a taxa de solicitações sob demanda excede o throughput permitido da conta e a tabela não pode ser escalada além disso.

OK para tentar novamente? Sim

ResourceInUseException

Mensagem: O recurso que você está tentando alterar está em uso.

Exemplo: você tentou recriar uma tabela existente ou excluir uma tabela atualmente no estado `CREATING`.

OK para tentar novamente? Não

ResourceNotFoundException

Mensagem: O recurso solicitado não foi encontrado.

Exemplo: a tabela que está sendo solicitada não existe ou está há pouco tempo no estado `CREATING`.

OK para tentar novamente? Não

ThrottlingException

Mensagem: A taxa de solicitações excede o throughput permitido.

Essa exceção é retornada como uma resposta da AmazonServiceException com um código de status THROTTLING_EXCEPTION. Essa exceção poderá ser retornada se você executar operações de API de plano de controle muito rapidamente.

Para as tabelas que usam o modo sob demanda, essa exceção poderá ser retornada para qualquer operação de API do plano de dados se a sua taxa de solicitação for muito alta. Para saber mais sobre escalabilidade sob demanda, consulte Throughput inicial e propriedades de escalabilidade.

O erro inclui uma lista de campos `ThrottlingReason` que apresenta um contexto específico sobre o motivo do controle de utilização, seguindo o formato `ResourceType+OperationType+LimitType` (p. ex., `TableReadKeyRangeThroughputExceeded` ou `IndexWriteMaxOnDemandThroughputExceeded`) e o ARN do recurso afetado. Essas informações ajudam a identificar qual recurso está sofrendo controle de utilização (tabela ou índice), qual tipo de operação acionou o controle (leitura ou gravação) e o limite específico que foi excedido (limites de partição ou throughput máximo sob demanda). Para ter mais informações sobre como diagnosticar e resolver problemas de controle de utilização, consulte Diagnosticar o controle de utilização.

OK para tentar novamente? Sim

UnrecognizedClientException

Mensagem: o ID da chave de acesso ou o token de segurança é inválido.

A assinatura da solicitação está incorreta. A causa mais provável é um ID de chave de acesso da AWS ou uma chave secreta inválidos.

OK para tentar novamente? Sim

ValidationException

Mensagem: varia dependendo do erro específico encontrado

Esse erro pode ocorrer por vários motivos, como um parâmetro necessário ausente, um valor fora do intervalo ou tipos de dados incompatíveis. A mensagem de erro contém detalhes sobre a parte específica da solicitação que causou o erro.

OK para tentar novamente? Não

Código de status HTTP 5xx

Um código de status HTTP `5xx` indica um problema que deve ser resolvido pela AWS. Pode ser um erro transitório e, nesse caso, é possível repetir a solicitação até que ela tenha êxito. Caso contrário, acesse o AWS Service Health Dashboard para ver se há problemas operacionais com o serviço.

Para obter mais informações, consulte Como resolver erros de HTTP 5xx no Amazon DynamoDB?

InternalServerError (HTTP 500)

O DynamoDB não pôde processar a solicitação.

OK para tentar novamente? Sim

> **Note**
>
> Você pode encontrar erros de servidor internos enquanto trabalha com itens. Esses são esperados durante a vida útil de uma tabela. Todas as solicitações com falha podem ser repetidas imediatamente.
> Quando você recebe um código de status 500 em uma operação de gravação, a operação pode ter sido concluída com sucesso ou com erro. Se a operação de gravação for uma solicitação `TransactWriteItem`, a operação pode ser repetida.
> Se a operação de gravação for uma solicitação de gravação de um único item, como `PutItem`, `UpdateItem` ou `DeleteItem`, sua aplicação deve ler o estado do item antes de tentar novamente a operação e/ou usar Exemplo de expressão de condição do DynamoDB na CLI para garantir que o item permaneça em um estado correto após tentar novamente, independentemente de a operação anterior ter sido concluída com sucesso ou com erro. Se a idempotência for um requisito para a operação de gravação, use `TransactWriteItem`, que é compatível com solicitações idempotentes, especificando automaticamente um `ClientRequestToken` para eliminar ambiguidades em várias tentativas de executar a mesma ação.

ServiceUnavailable (HTTP 503)

O DynamoDB está indisponível no momento. (Esse estado deve ser temporário.)

OK para tentar novamente? Sim

Tratamento de erros na aplicação

Para que seu aplicativo seja executado sem problemas, é necessário adicionar uma lógica para detectar erros e reagir a eles. As abordagens comuns incluem o uso de blocos `try-catch` ou de uma instrução `if-then`.

Os AWS SDKs realizam por conta própria novas tentativas e verificações de erros. Se você se deparar com um erro enquanto usa um dos AWS SDKs, o código de erro e sua descrição poderão ajudar a solucioná-lo.

Você também deve ver um `Request ID` na resposta. O `Request ID` pode ser útil se você precisa trabalhar com o AWS Support para diagnosticar um problema.

Repetições de erro e recuo exponencial

Vários componentes em uma rede, como servidores DNS, switches, load balancers e outros, podem gerar erros em qualquer momento do ciclo de vida de uma determinada solicitação. A técnica usual para lidar com essas respostas de erro em um ambiente de rede é implementar novas tentativas no aplicativo cliente. Essa técnica aumenta a confiabilidade da aplicação.

Cada AWS SDK implementa a lógica de novas tentativas automaticamente. Você pode modificar os parâmetros de novas tentativas de acordo com as suas necessidades. Por exemplo, considere um aplicativo Java que exija uma estratégia rápida contra falhas, sem permitir novas tentativas em caso de erro. Com o AWS SDK para Java, você poderia usar a classe `ClientConfiguration` e fornecer um valor `maxErrorRetry` de `0` para desativar as novas tentativas. Para obter mais informações, consulte a documentação do AWS SDK para sua linguagem de programação.

Se não estiver usando um AWS SDK, você deverá tentar novamente as solicitações originais que recebem erros do servidor (5xx). No entanto, erros de cliente (4xx, diferente de `ThrottlingException` ou `ProvisionedThroughputExceededException`) indicam que você precisa revisar a solicitação para corrigir o problema antes de tentar novamente. Para obter recomendações detalhadas para abordar cenários específicos de controle de utilização, consulte a seção Solução de problemas de controle de utilização do DynamoDB.

Além de novas tentativas simples, cada AWS SDK implementa um algoritmo de recuo exponencial para um melhor controle de fluxo. O conceito por detrás do recuo exponencial é usar esperas progressivamente mais longas entre as novas tentativas para respostas de erro consecutivas. Por exemplo, até 50 milissegundos antes da primeira nova tentativa, até 100 milissegundos antes da segundo, até 200 milissegundos antes da terceira e assim por diante. No entanto, depois de

um minuto, se a solicitação não tiver sido bem-sucedida, talvez o problema esteja relacionado ao tamanho da solicitação que exceda o throughput provisionado e não à taxa de solicitação. Defina um tempo de interrupção de cerca de um minuto para o número máximo de novas tentativas. Se a solicitação não for bem-sucedida, investigue suas opções de throughput provisionado.

> **Note**
>
> Os AWS SDKs implementam uma lógica de novas tentativas automáticas e de recuo exponencial.

A maioria dos algoritmos de recuo exponencial usam variação (atraso randomizado) para evitar colisões sucessivas. Como você não está tentando evitar essas colisões nesses casos, não precisa usar esse número aleatório. No entanto, se você usar clientes simultâneos, a variação pode ajudar suas solicitações a serem bem-sucedidas mais depressa. Para obter mais informações, consulte a postagem no blog sobre Recuo exponencial e variação.

Operações em lote e tratamento de erros

A API de baixo nível do DynamoDB oferece suporte a operações em lote para leituras e gravações. `BatchGetItem` lê os itens de uma ou mais tabelas, e `BatchWriteItem` insere ou exclui itens de uma ou mais tabelas. Essas operações em lote são implementadas como wrappers em torno de outras operações do DynamoDB que não são feitas em lote. Em outras palavras, `BatchGetItem` invoca `GetItem` uma vez para cada item do lote. Da mesma forma, `BatchWriteItem` invoca `DeleteItem` ou `PutItem`, conforme apropriado, para cada item do lote.

Uma operação em lote pode tolerar a falha de solicitações individuais no lote. Por exemplo, considere uma solicitação `BatchGetItem` para ler cinco itens. Mesmo se algumas das solicitações `GetItem` subjacentes falharem, isso não faz com que toda a operação `BatchGetItem` falhe. Entretanto, se todas as cinco operações de leitura falharem, todo o `BatchGetItem` falhará.

As operações em lote retornam informações sobre solicitações individuais que apresentam falhas, para que você possa diagnosticar o problema e repetir a operação. Para `BatchGetItem`, as tabelas e chaves primárias em questão são retornadas no valor de `UnprocessedKeys` da resposta. Para `BatchWriteItem`, informações semelhantes são retornadas em `UnprocessedItems`.

A causa mais provável de uma falha de leitura ou gravação é a controle de utilização. Para `BatchGetItem`, uma ou mais das tabelas na solicitação em lote não tem capacidade de leitura

provisionada suficiente para dar suporte à operação. Para `BatchWriteItem`, uma ou mais das tabelas não tem capacidade de gravação provisionada suficiente.

Se o DynamoDB retornar itens não processados, você deverá repetir a operação em lote nesses itens. No entanto, recomendamos que você use um algoritmo de recuo exponencial. Se você repetir a operação em lote imediatamente, solicitações subjacentes de leitura ou gravação ainda poderão falhar devido ao controle de utilização nas tabelas individuais. Se você atrasar a operação em lote usando o recuo exponencial, as solicitações individuais no lote terão muito mais chances de sucesso.

Usar o DynamoDB com um AWS SDK

Os kits de desenvolvimento de software (SDKs) da AWS estão disponíveis para muitas linguagens de programação populares. Cada SDK fornece uma API, exemplos de código e documentação que permitem que os desenvolvedores criem facilmente aplicações em seu idioma de preferência.

Documentação do SDK	Exemplos de código
AWS SDK para C++	Exemplos de código do AWS SDK para C++
AWS CLI	Exemplos de código do AWS CLI
AWS SDK para Go	Exemplos de código do AWS SDK para Go
AWS SDK para Java	Exemplos de código do AWS SDK para Java
AWS SDK para JavaScript	Exemplos de código do AWS SDK para JavaScript
AWS SDK para Kotlin	Exemplos de código do AWS SDK para Kotlin
AWS SDK para .NET	Exemplos de código do AWS SDK para .NET
AWS SDK para PHP	Exemplos de código do AWS SDK para PHP
Ferramentas da AWS para PowerShell	Exemplos de código do Ferramentas da AWS para PowerShell
AWS SDK para Python (Boto3)	Exemplos de código do AWS SDK para Python (Boto3)

Documentação do SDK	Exemplos de código
AWS SDK para Ruby	Exemplos de código do AWS SDK para Ruby
AWS SDK para Rust	Exemplos de código do AWS SDK para Rust
SDK da AWS para SAP ABAP	Exemplos de código do SDK da AWS para SAP ABAP
AWS SDK for Swift	Exemplos de código do AWS SDK for Swift

Para obter exemplos específicos do DynamoDB, consulte Exemplos de código do DynamoDB usando AWS SDKs.

> **Exemplo de disponibilidade**
>
> Não consegue encontrar o que precisa? Solicite um exemplo de código usando o link Fornecer feedback na parte inferior desta página.

Trabalhar com tabelas, itens, consultas, verificações e índices

Esta seção fornece detalhes sobre como trabalhar com tabelas, itens, consultas e muito mais no Amazon DynamoDB.

Tópicos

- Trabalhar com tabelas e dados no DynamoDB
- Tabelas globais: replicação multiativa e multirregional
- Trabalhar com itens e atributos no DynamoDB
- Melhorar o acesso aos dados com índices secundários no DynamoDB
- Gerenciar fluxos de trabalho complexos com transações do DynamoDB
- Captura de dados de alterações com o Amazon DynamoDB

Trabalhar com tabelas e dados no DynamoDB

Esta seção descreve como usar a AWS Command Line Interface (AWS CLI) e os AWS SDKs para criar, atualizar e excluir tabelas no Amazon DynamoDB.

> **Note**
>
> Você também pode executar essas mesmas tarefas usando o AWS Management Console. Para obter mais informações, consulte Utilizar o console.

Esta seção também fornece mais informações sobre a capacidade de throughput usando o Auto Scaling do DynamoDB ou configurando manualmente o throughput provisionado.

Tópicos

- Operações básicas em tabelas do DynamoDB
- Considerações ao escolher uma classe de tabela no DynamoDB
- Adicionar etiquetas e rótulos a recursos no DynamoDB

Operações básicas em tabelas do DynamoDB

De forma semelhante a outros sistemas de banco de dados, o Amazon DynamoDB armazena dados em tabelas. É possível gerenciar suas tabelas usando algumas operações básicas.

Tópicos

- Criar uma tabela
- Descrever uma tabela
- Atualizar uma tabela
- Excluir uma tabela
- Usar a proteção contra exclusão
- Nomes de tabela de listagem
- Descrever cotas de throughput provisionado

Criar uma tabela

Use a operação `CreateTable` para criar uma tabela no Amazon DynamoDB. Para criar a tabela, você deve fornecer as seguintes informações:

- Nome da tabela. O nome deve estar de acordo com as regras de nomenclatura do DynamoDB e deve ser exclusivo na conta e na região atuais da AWS. Por exemplo, você poderia criar uma tabela `People` no Leste dos EUA (Norte da Virgínia) e outra tabela `People` na Europa (Irlanda). No entanto, essas duas tabelas devem ser inteiramente diferente uma da outra. Para obter mais informações, consulte Tipos de dados compatíveis e regras de nomenclatura no Amazon DynamoDB.
- Chave primária. A chave primária pode consistir em um atributo (chave de partição) ou de dois atributos (chave de partição e chave de classificação). Você precisa fornecer os nomes de atributos, os tipos de dados e a função de cada um: `HASH` (para uma chave de partição) e `RANGE` (para uma chave de classificação). Para obter mais informações, consulte Chave primária.
- Configurações de throughput (para tabelas provisionadas). Se estiver usando o modo provisionado, você deve especificar as configurações de throughput de leitura e gravação inicial da tabela. Você pode modificar essas configurações mais tarde ou habilitar o Auto Scaling do DynamoDB para gerenciar as configurações para você. Para obter mais informações, consulte Modo de capacidade provisionada do DynamoDB e Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB.

Exemplo 1: criar uma tabela sob demanda

Para criar a mesma tabela `Music` usando modo sob demanda.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode=PAY_PER_REQUEST
```

A operação `CreateTable` retorna metadados para a tabela, conforme mostrado a seguir.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 0,
            "ReadCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        },
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
```

```
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397468.348
    }
}
```

> **Important**
>
> Ao chamar `DescribeTable` em uma tabela sob demanda, as unidades de capacidade de leitura e unidades de capacidade de gravação são definidas como 0.

Exemplo 2: criar uma tabela provisionada

O exemplo da AWS CLI a seguir mostra como criar uma tabela (`Music`). A chave primária consiste em `Artist` (chave de partição) e `SongTitle` (chave de classificação), cada uma delas tem um tipo de dados de `String`. O throughput máximo da tabela é 10 unidades de capacidade de leitura e 5 unidades de capacidade de gravação.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

A operação `CreateTable` retorna metadados para a tabela, conforme mostrado a seguir.

```
{
    "TableDescription": {
```

```
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

O elemento `TableStatus` indica o estado atual da tabela (`CREATING`). Pode demorar um pouco para criar a tabela, dependendo dos valores que você especificar para `ReadCapacityUnits` e `WriteCapacityUnits`. Valores maiores exigem que o DynamoDB aloque mais recursos para a tabela.

Exemplo 3: criar uma tabela usando a classe de tabela Standard-Infrequent Access do DynamoDB

Para criar a mesma tabela de `Music` usando a classe de tabela Standard-Infrequent Access (Padrão – Acesso Infrequente) do DynamoDB.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

A operação `CreateTable` retorna metadados para a tabela, conforme mostrado a seguir.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableClassSummary": {
            "LastUpdateDateTime": 1542397215.37,
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
```

```
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

Descrever uma tabela

Para visualizar detalhes sobre uma tabela, use a operação `DescribeTable`. Você deve fornecer o nome da tabela. A saída de `DescribeTable` está no mesmo formato de `CreateTable`. Ela inclui o a marca de data e hora em que a tabela foi criada, o esquema de chaves, as configurações de throughput provisionado, o tamanho estimado e os índices secundários que estão presentes.

> **Important**
>
> Ao chamar `DescribeTable` em uma tabela sob demanda, as unidades de capacidade de leitura e unidades de capacidade de gravação são definidas como 0.

Example

```
aws dynamodb describe-table --table-name Music
```

A tabela estará pronta para uso quando `TableStatus` tiver sido alterado de `CREATING` para `ACTIVE`.

> **Note**
>
> Se você emitir uma solicitação `DescribeTable` imediatamente após uma solicitação `CreateTable`, o DynamoDB poderá retornar um erro (`ResourceNotFoundException`).

Isso ocorre porque `DescribeTable` usa uma consulta eventualmente consistente e os metadados da sua tabela podem não estar disponíveis nesse momento. Aguarde alguns segundos e, em seguida, tente a solicitação `DescribeTable` novamente.
Para fins de faturamento, os custos de armazenamento do DynamoDB incluem uma sobrecarga de 100 bytes por item. (Para obter mais informações, acesse a Preços do DynamoDB.) Esses 100 bytes extras por item são usados em cálculos da unidade de capacidade ou pela operação `DescribeTable`.

Atualizar uma tabela

A operação `UpdateTable` permite que você execute uma das seguintes ações:

- Modificar as configurações de throughput provisionad de uma tabela (para tabelas de modo provisionadas).
- Alterar o modo de capacidade de leitura/gravação de tabela.
- Manipular índices secundários globais na tabela (consulte Como usar índices secundários globais no DynamoDB).
- Habilite ou desabilite o DynamoDB Streams na tabela (consulte Capturar dados de alterações para o DynamoDB Streams).

Example

O exemplo de AWS CLI a seguir mostra como modificar as configurações de throughput provisionado de uma tabela.

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

Note

Quando você emite uma solicitação `UpdateTable`, o status da tabela muda de `AVAILABLE` para `UPDATING`. A tabela permanecerá totalmente disponível para uso enquanto estiver `UPDATING`. Quando esse processo for concluído, o status da tabela mudará de `UPDATING` para `AVAILABLE`.

Example

O exemplo de AWS CLI a seguir mostra como modificar o modo de capacidade de leitura/gravação de uma tabela para o modo sob demanda.

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

Excluir uma tabela

Você pode remover uma tabela não utilizada com a operação `DeleteTable`. Excluir uma tabela é uma operação irrecuperável. Para excluir uma tabela usando o AWS Management Console, consulte the section called “Etapa 6: (Opcional) limpar”.

Example

O exemplo de AWS CLI a seguir mostra como excluir uma tabela.

```
aws dynamodb delete-table --table-name Music
```

Quando você emite uma solicitação `DeleteTable`, o status da tabela muda de `ACTIVE` para `DELETING`. A exclusão da tabela pode demorar um pouco, dependendo dos recursos que ela usa (como os dados armazenados e os streams ou índices da tabela).

Quando a operação `DeleteTable` é concluída, a tabela deixa de existir no DynamoDB.

Usar a proteção contra exclusão

É possível proteger uma tabela contra exclusão acidental com a propriedade de proteção contra exclusão. Habilitar essa propriedade para tabelas ajuda a garantir que elas não sejam excluídas acidentalmente durante as operações regulares de gerenciamento de tabelas pelos administradores. Isso ajudará a evitar interrupções nas operações empresariais normais.

O proprietário da tabela ou um administrador autorizado controla a propriedade de proteção contra exclusão de cada tabela. A propriedade de proteção contra exclusão de cada tabela está desativada por padrão. Isso inclui réplicas globais e tabelas restauradas com base em backups. Quando a proteção contra exclusão está desabilitada para uma tabela, ela pode ser excluída por qualquer usuário autorizado por uma política do Identity and Access Management (IAM). Quando a proteção contra exclusão está habilitada para uma tabela, ninguém pode excluí-la.

Para alterar essa configuração, acesse Configurações adicionais da tabela, navegue até o painel Proteção contra exclusão e selecione Ativar Proteção contra exclusão.

A propriedade de proteção contra exclusão é compatível com o console do DynamoDB, API, CLI/SDK e AWS CloudFormation. A API `CreateTable` é compatível com a propriedade de proteção contra exclusão no momento da criação da tabela, e a API `UpdateTable` é compatível com a alteração da propriedade de proteção contra exclusão para tabelas existentes.

> **Note**
>
> - Se uma conta da AWS for excluída, todos os dados dessa conta, incluindo as tabelas, também serão excluídos em 90 dias.
> - Se o DynamoDB perder o acesso a uma chave gerenciada pelo cliente que foi usada para criptografar uma tabela, ela ainda será arquivada. O arquivamento requer um backup da tabela e a exclusão da original.

Nomes de tabela de listagem

A operação `ListTables` retorna os nomes das tabelas do DynamoDB da conta e da região atuais da AWS.

Example

O exemplo da AWS CLI a seguir mostra como listar os nomes de tabelas do DynamoDB.

```
aws dynamodb list-tables
```

Descrever cotas de throughput provisionado

A operação `DescribeLimits` retorna as cotas de capacidade de leitura e gravação atuais da conta e da região atuais da AWS.

Example

O exemplo de AWS CLI a seguir mostra como descrever as cotas atuais de throughput provisionado.

```
aws dynamodb describe-limits
```

A saída mostra as cotas superiores das unidades de capacidade de leitura e gravação da conta e da região atuais da AWS.

Para obter mais informações sobre essas cotas e como solicitar aumentos de cota, consulte Cotas padrão de throughput.

Considerações ao escolher uma classe de tabela no DynamoDB

O DynamoDB apresenta duas classes de tabela projetadas para ajudar você a otimizar o custo. A classe de tabela Standard do DynamoDB é a padrão e é recomendada para a grande maioria das workloads. A classe de tabela do DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) é otimizada para tabelas em que o armazenamento é o custo dominante. Por exemplo, tabelas que armazenam dados acessados com pouca frequência, como logs da aplicação, postagens antigas de mídia social, histórico de pedidos de comércio eletrônico e antigas conquistas de jogos, são bons candidatos para a classe de tabela Standard-IA.

Cada tabela do DynamoDB está associada a uma classe de tabela. Todos os índices secundários associados à tabela usam a mesma classe de tabela. Você pode definir sua classe de tabela ao criar sua tabela (DynamoDB Standard por padrão) e atualizar a classe de tabela de uma tabela existente usando o AWS Management Console, a CLI da AWS ou o AWS SDK. O DynamoDB também permite o gerenciamento de sua classe de tabela usando o AWS CloudFormation para tabelas de região única (tabelas que não são tabelas globais). Cada classe de tabela oferece preços diferentes para armazenamento de dados, bem como para solicitações de leitura e gravação. Ao escolher uma classe de tabela para sua tabela, lembre-se do seguinte:

- A classe de tabela Standard do DynamoDB oferece custos de throughput mais baixos do que o DynamoDB Standard-IA e é a opção mais econômica para tabelas em que o throughput é o custo dominante.
- A classe de tabela DynamoDB Standard-IA oferece custos de armazenamento mais baixos do que o DynamoDB Standard e é a opção mais econômica para tabelas em que o armazenamento é o custo dominante. Quando o armazenamento excede 50% do custo de throughput (leituras e gravações) de uma tabela usando a classe de tabela Standard do DynamoDB, a classe de tabela DynamoDB Standard-IA pode ajudá-lo a reduzir o custo total da tabela.
- As tabelas DynamoDB Standard – IA oferecem a mesma performance, durabilidade e disponibilidade que as tabelas padrão do DynamoDB.
- A alternância entre as classes de tabela DynamoDB Standard e DynamoDB Standard – IA não requer a alteração do código da aplicação. Você usa as mesmas APIs do DynamoDB e endpoints de serviço, independentemente da classe de tabela que suas tabelas utilizem.

- As tabelas do DynamoDB Standard – IA são compatíveis com todos os recursos existentes do DynamoDB, como autoescalabilidade, modo sob demanda, vida útil (TTL), backups sob demanda, recuperação em um ponto anterior no tempo (PITR) e índices secundários globais.

A classe de tabela mais econômica para sua tabela depende dos padrões de uso de armazenamento e throughput esperados da tabela. Você pode ver o histórico de armazenamento e o custo de throughput da sua tabela com o AWS Cost and Usage Reports e o AWS Cost Explorer. Use esses dados de histórico para determinar a classe de tabela mais econômica para sua tabela. Para saber mais sobre o uso do AWS Cost and Usage Reports e o AWS Cost Explorer, veja a Documentação de Billing and Cost Management do AWS. Para mais informações sobre a precificação das classes de tabela, acesse Preço do Amazon DynamoDB.

> **Note**
>
> Uma atualização de classe de tabela é um processo em segundo plano. Você ainda pode acessar sua tabela normalmente durante uma atualização de classe de tabela. O tempo para atualizar sua classe de tabela depende do tráfego da tabela, do tamanho do armazenamento e de outras variáveis relacionadas. Não são permitidas mais do que duas atualizações de classe de tabela em sua tabela em um período de trilha de 30 dias.

Adicionar etiquetas e rótulos a recursos no DynamoDB

Você pode rotular os recursos do Amazon DynamoDB com tags. As tags permitem categorizar recursos de diferentes maneiras, por exemplo, por finalidade, proprietário, ambiente ou outros critérios. As tags podem ajudar a fazer o seguinte:

- Identificar rapidamente um recurso com base nas tags que você atribuiu a ele.
- Veja as faturas da AWS discriminadas por tags.

> **Note**
>
> Todos os índices secundários locais (LSI) e índices secundários globais (GSI) relacionados a tabelas marcadas são rotulados com as mesmas tags automaticamente. Atualmente, o uso de Streams pelo DynamoDB não pode ser marcado.

A marcação é suportada por serviços da AWS como Amazon EC2, Amazon S3, DynamoDB e outros. Uma marcação eficiente é capaz de fornecer insights de custos, permitindo criar relatórios entre serviços que possuem uma tag específica.

Para começar a usar tags, faça o seguinte:

1. Compreender Restrições de marcação no DynamoDB.
2. Criar tags usando Marcar recursos no DynamoDB.
3. Use Usar etiquetas do DynamoDB para criar relatórios de alocação de custos para controlar seus custos da AWS por tag ativa.

Por fim, é recomendável seguir estratégias de marcação ideais. Para obter informações, consulte Estratégias de marcação da AWS.

Restrições de marcação no DynamoDB

Cada tag consiste em uma chave e um valor, ambos definidos por você. As seguintes restrições são aplicáveis:

- Cada tabela do DynamoDB pode ter apenas uma tag com a mesma chave. Se você tentar adicionar uma tag existente (mesma chave), o valor da tag existente será atualizado para o novo valor.
- As chaves e valores das tags diferenciam maiúsculas de minúsculas.
- O comprimento máximo da chave é 128 caracteres Unicode.
- O número máximo de tags que você pode atribuir a um recurso é 50.
- Os caracteres permitidos são letras, espaço em branco e números, além dos seguintes caracteres especiais: `+ - = . _ : /`
- O número máximo de tags por recurso é 50.
- O tamanho máximo aceito para todas as tags em uma tabela é de 10 KB.
- Nomes e valores de tags atribuídos pela AWS recebem automaticamente o prefixo `aws:`, o qual não pode ser atribuído por você. Nomes de tags atribuídos pela AWS não contam para o limite de tag de recurso definido pelo usuário de 50 ou o limite máximo de tamanho de 10 KB. Nomes de tags atribuídos pelo usuário têm o prefixo `user:` no relatório de alocação de custos.
- Não é possível colocar uma data retroativa na aplicação de uma tag.

Marcar recursos no DynamoDB

Você pode usar o console do Amazon DynamoDB ou a AWS Command Line Interface (AWS CLI) para adicionar, listar, editar ou excluir tags. Em seguida, você pode ativar essas tags definidas pelo usuário para que elas apareçam no console do Gerenciamento de Faturamento e Custos da AWS para o controle da alocação de custos. Para obter mais informações, consulte Usar etiquetas do DynamoDB para criar relatórios de alocação de custos.

Para edição em massa, também é possível usar o Tag Editor no AWS Management Console. Para obter mais informações, consulte Trabalhar com o Tag Editor.

Para usar a API do DynamoDB, consulte as operações a seguir na Referência da API do Amazon DynamoDB:

- TagResource
- UntagResource
- ListTagsOfResource

Tópicos

- Configuração de permissões para filtrar por tags
- Adição de tags a tabelas novas ou existentes (AWS Management Console)
- Adição de tags a tabelas novas ou existentes (AWS CLI)

Configuração de permissões para filtrar por tags

Para usar etiquetas a fim de filtrar sua lista de tabelas no console do DynamoDB, as políticas do seu usuário devem incluir acesso às seguintes operações:

- `tag:GetTagKeys`
- `tag:GetTagValues`

Para acessar essas operações, anexe uma nova política do IAM ao seu usuário seguindo as etapas abaixo.

1. Acesse o console do IAM com um usuário administrativo.
2. No painel de navegação à esquerda, selecione "Policies" (Políticas).

3. Selecione "Create Policy" (Criar política).
4. No editor, a política a seguir no editor de JSON.

   JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Conclua o assistente e atribua um nome à política (por exemplo, `TagKeysAndValuesReadAccess`).
6. No menu de navegação à esquerda, selecione "Users" (Usuários).
7. Na lista, selecione o usuário que você normalmente usa para acessar o console do DynamoDB.
8. Selecione "Add permissions" (Adicionar permissões).
9. Selecione "Attach existing policies directly" (Anexar políticas existentes diretamente).
10. Na lista, selecione a política que você criou anteriormente.
11. Assista todo o assistente.

Adição de tags a tabelas novas ou existentes (AWS Management Console)

Você pode usar o console do DynamoDB para adicionar tags a novas tabelas ao criá-las, ou para adicionar, editar ou excluir tags de tabelas existentes.

Para marcar recursos na criação (console)

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, selecione Tables (Tabelas) e Create table (Criar tabela).

3. Na página Create DynamoDB table (Criar tabela DynamoDB), forneça um nome e uma chave primária. Na seção Tags (Etiquetas), escolha Add new tag (Adicionar nova etiqueta) e insira as etiquetas que você deseja usar.

   Para obter informações sobre a estrutura da tag, consulte Restrições de marcação no DynamoDB.

   Para obter mais informações sobre como criar tabelas, consulte Operações básicas em tabelas do DynamoDB.

Para marcar recursos existentes (console)

Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.

1. No painel de navegação, selecione Tables (Tabelas).
2. Escolha uma tabela na lista e escolha a guia Additional settings (Configurações adicionais). Você pode adicionar, editar ou excluir seus marcadores na seção Tags (Etiquetas) na parte inferior da página.

Adição de tags a tabelas novas ou existentes (AWS CLI)

Os exemplos a seguir mostram como usar a AWS CLI para especificar tags ao criar tabelas e índices e para marcar recursos existentes.

Para marcar recursos na criação (AWS CLI)

- O exemplo a seguir cria uma nova tabela `Movies` e adiciona a tag `Owner` com um valor de `blueTeam`:

```
aws dynamodb create-table \
    --table-name Movies \
    --attribute-definitions AttributeName=Title,AttributeType=S \
    --key-schema AttributeName=Title,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Para marcar recursos existentes (AWS CLI)

- O exemplo a seguir adiciona a tag `Owner` com um valor de `blueTeam` à tabela `Movies`:

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

Para listar todas as tags de uma tabela (AWS CLI)

- O exemplo a seguir relaciona todas as tags associadas à tabela `Movies`.

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

Usar etiquetas do DynamoDB para criar relatórios de alocação de custos

A AWS usa tags para organizar os custos de recursos no seu relatório de alocação de custos. A AWS fornece dois tipos de tags alocação de custos:

- Uma tag gerada pela AWS. A AWS define, cria e aplica essa tag para você.
- Tags definidas pelo usuário. Você define, cria e aplica essas tags.

É necessário ativar os dois tipos de tags separadamente para que elas possam ser exibidas no Cost Explorer ou em um relatório de alocação de custos.

Para ativar tags geradas pela AWS:

1. Faça login no AWS Management Console e abra o console do Billing and Cost Management em https://console.aws.amazon.com/billing/home#/.
2. No painel de navegação, escolha Cost Allocation Tags.
3. Em AWSGenerated Cost Allocation Tags (Etiquetas de alocação de custos geradas), escolha Activate (Ativar).

Para ativar tags definidas pelo usuário:

1. Faça login no AWS Management Console e abra o console do Billing and Cost Management em https://console.aws.amazon.com/billing/home#/.
2. No painel de navegação, escolha Cost Allocation Tags.

3. Em Tags de alocação de custos definidos pelo usuário, escolha Ativar.

Após você criar e ativar tags, a AWS gera um relatório de alocação de custos com a utilização e os custos agrupados de acordo com as tags ativas. O relatório de alocação de custos inclui todos os seus custos da AWS para cada período de faturamento. O relatório inclui recursos com e sem tags, para que você possa organizar claramente as cobranças de cada um deles.

> **Note**
>
> No momento, os dados transferidos do DynamoDB não são categorizados por tags em relatórios de alocação de custos.

Para obter mais informações, consulte Usar tags de alocação de custos.

Tabelas globais: replicação multiativa e multirregional

As tabelas globais do Amazon DynamoDB são um atributo de banco de dados totalmente gerenciado, multiativo e multirregional que fornece replicação de dados fácil de usar e performance de leitura e gravação local rápida para aplicações globalmente escaladas.

As tabelas globais replicam automaticamente os dados da tabela do DynamoDB em várias Regiões da AWS sem exigir que você crie e mantenha sua própria solução de replicação. Quando você escolhe as Regiões da AWS para suas réplicas de tabela, as tabelas globais tratam todas as replicações automaticamente. As tabelas globais estão disponíveis em todas as regiões onde o DynamoDB está disponível.

As tabelas globais oferecem os seguintes benefícios:

- Replique automaticamente os dados da tabela do DynamoDB em todas as regiões de sua escolha da AWS para localizar os dados mais perto de seus usuários
- Permita maior disponibilidade de aplicativos durante o isolamento ou degradação regional
- Elimine a resolução de conflitos de atualização para que você possa se concentrar na lógica de negócios do seu aplicativo

As tabelas globais são ideais para aplicativos que exigem continuidade de negócios e alta disponibilidade por meio da implantação em várias regiões. Qualquer réplica de tabela global pode

servir para leituras e gravações. Os aplicativos podem alcançar alta resiliência com um objetivo de ponto de recuperação (RPO) baixo ou zero, transferindo o tráfego para uma região diferente se o processamento do aplicativo for interrompido em uma região.

Você pode configurar uma tabela global usando o AWS Management Console. As tabelas globais usam as APIs existentes do DynamoDB para ler e gravar dados em suas tabelas, portanto, nenhuma alteração no aplicativo é necessária. Você paga apenas pelos recursos provisionados ou usados, sem nenhum custo ou compromisso inicial. Para começar com tabelas globais, consulte Tutoriais: Criação de tabelas globais.

Tópicos

- Como funcionam as tabelas globais do DynamoDB
- Tutoriais: Criação de tabelas globais
- Segurança de tabelas globais do DynamoDB
- Conceitos básicos sobre faturamento do Amazon DynamoDB para tabelas globais
- Versões de tabelas globais do DynamoDB
- Melhores práticas para tabelas globais

Como funcionam as tabelas globais do DynamoDB

As seções a seguir descrevem os conceitos e os comportamentos das tabelas globais no Amazon DynamoDB.

Conceitos

As tabelas globais são um recurso do DynamoDB que replica os dados da tabela em todas as regiões da AWS.

Uma tabela-réplica (ou réplica) é uma única tabela do DynamoDB que funciona como parte de uma tabela global. Uma tabela global consiste em duas ou mais tabelas de réplica em diferentes regiões da AWS. Cada tabela global só pode ter uma réplica por região da AWS. Todas as réplicas em uma tabela global compartilham o mesmo nome da tabela, esquema de chave primária e dados do item.

Quando uma aplicação grava dados em uma réplica em uma região, o DynamoDB replica automaticamente a gravação para as outras réplicas na tabela global. Para obter mais informações sobre conceitos básicos de tabelas globais, consulte Tutoriais: Criação de tabelas globais.

Versões

Há duas versões disponíveis das tabelas globais do DynamoDB: versão 2019.11.21 (atual) e versão 2017.11.29 (legada). Você deve usar a versão 2019.11.21 (atual) sempre que possível. As informações nesta seção de documentação são para a versão 2019.11.21 (atual). Para obter mais informações, consulte Determinar a versão de uma tabela global.

Disponibilidade

As tabelas globais ajudam a melhorar a continuidade de seus negócios, facilitando a implementação de uma arquitetura de alta disponibilidade multirregional. Se uma workload em uma única região da AWS se tornar prejudicada, você poderá transferir o tráfego do aplicativo para uma região diferente e executar leituras e gravações em uma tabela de réplica diferente na mesma tabela global.

Cada tabela de réplica em uma tabela global oferece a mesma durabilidade e disponibilidade de uma tabela do DynamoDB de região única. As tabelas globais oferecem um Acordo de Serviço (SLA) com 99,999% de disponibilidade, em comparação com 99,99% das tabelas de região única.

Modos de consistência

Ao criar uma tabela global, você pode configurar o modo de consistência. As tabelas globais oferecem suporte a dois modos de consistência: consistência final multirregional (MREC) e consistência forte multirregional (MRSC).

Se você não especificar um modo de consistência ao criar uma tabela global, ela usará como padrão a consistência final multirregional (MREC). Uma tabela global não pode conter réplicas configuradas com modos de consistência diferentes. Não é possível alterar o modo de consistência de uma tabela global depois da criação.

Consistência final multirregional (MREC)

A consistência final multirregional (MREC) é o modo de consistência padrão para tabelas globais. As alterações feitas em um item em uma réplica da tabela global MREC são replicadas de forma assíncrona em todas as outras réplicas, normalmente em um segundo ou menos. No caso improvável de uma réplica em uma tabela global do MREC ficar isolada ou danificada, todos os dados ainda não replicados para outras regiões serão replicados quando a réplica ficar íntegra.

Se o mesmo item for modificado em várias regiões simultaneamente, o DynamoDB resolverá o conflito usando a modificação com o registro de data e hora interno mais recente por item, conhecido como método de resolução de conflitos do tipo “último escritor vence”. Um item acabará por convergir em todas as réplicas para a versão criada pela última gravação.

Operações de leitura altamente consistentes retornam a versão mais recente de um item se ele tiver sido atualizado pela última vez na região onde a leitura ocorreu, mas podem retornar dados obsoletos se o item tiver sido atualizado pela última vez em uma região diferente. As gravações condicionais avaliam a expressão da condição em relação à versão do item na região.

Você cria uma tabela global MREC adicionando uma réplica a uma tabela existente do DynamoDB. Adicionar uma réplica não afeta o desempenho das tabelas existentes do DynamoDB de região única ou nas réplicas de tabelas globais existentes. Você pode adicionar réplicas a uma tabela global do MREC para expandir o número de regiões em que os dados são replicados ou remover réplicas de uma tabela global do MREC se elas não forem mais necessárias. Uma tabela global do MREC pode ter uma réplica em qualquer região onde o DynamoDB esteja disponível e pode ter quantas réplicas houver regiões na partição AWS.

Consistência forte multirregional (MRSC)

Você pode configurar o modo de consistência forte multirregional (MRSC) ao criar uma tabela global. As alterações de item em uma réplica da tabela global do MRSC são replicadas de forma síncrona em pelo menos uma outra região antes que a operação de gravação retorne uma resposta bem-sucedida. Ao converter uma tabela de região única existente em uma tabela global com MRSC, você deve garantir que a tabela se mantenha vazia até que a conclusão da conversão para assegurar que a inicialização e a configuração da replicação sejam adequadas.

Operações de leitura altamente consistentes em qualquer réplica do MRSC sempre retornam a versão mais recente de um item. As gravações condicionais sempre avaliam a expressão da condição em relação à versão mais recente de um item.

Uma operação de gravação falha com um erro `ReplicatedWriteConflictException` quando ela tenta modificar um item que já está sendo modificado em outra região. As gravações que falharem com o erro `ReplicatedWriteConflictException` poderão ser repetidas e serão bem-sucedidas se o item não estiver mais sendo modificado em outra região.

Você pode configurar uma tabela global MRSC com três réplicas ou duas réplicas e uma testemunha. Uma testemunha é um componente de uma tabela global do MRSC que contém dados gravados em réplicas da tabela global e fornece uma alternativa opcional a uma réplica completa, ao mesmo tempo em que oferece suporte à arquitetura de disponibilidade do MRSC. Você não pode realizar operações de leitura ou gravação em uma testemunha. Uma testemunha está localizada em uma região diferente das duas réplicas.

Ao criar uma tabela global do MRSC, você escolhe as regiões para suas réplicas e para a implantação da testemunha no momento da criação da tabela MRSC. Você pode determinar se e

em qual região uma tabela global do MRSC tem uma testemunha configurada a partir da saída da API `DescribeTable`. A testemunha pertence e é gerenciada pelo DynamoDB, e a testemunha não aparecerá na sua conta da AWS na região em que está configurada.

Uma tabela global do MRSC deve ser implantada em exatamente três regiões. Você cria uma tabela global MRSC adicionando uma réplica e uma testemunha ou duas réplicas a uma tabela existente do DynamoDB que não contém dados. Você não pode adicionar réplicas adicionais a uma tabela global MRSC existente. Não é possível excluir uma única réplica ou testemunha de uma tabela global do MRSC. Você pode excluir duas réplicas ou excluir uma réplica e uma testemunha de uma tabela global do MRSC, convertendo a réplica restante em uma tabela do DynamoDB de região única.

As seguintes considerações se aplicam às tabelas globais do MRSC:

- Ao converter uma tabela de região única em uma tabela global com MRSC, você deve garantir que a tabela esteja vazia. Não é possível converter uma tabela de região única em uma tabela global com MRSC que já contenha itens. Nenhum dado deve ser gravado na tabela durante o processo de conversão.
- As tabelas globais do MRSC estão disponíveis nos seguintes conjuntos de regiões:
  - Conjunto de regiões dos EUA: Leste dos EUA (Norte da Virgínia), Leste dos EUA (Ohio), Oeste dos EUA (Oregon)
  - Conjunto de regiões da Europa: Europa (Irlanda), Europa (Londres), Europa (Paris), Europa (Frankfurt)
  - Conjunto de regiões da AP: Ásia-Pacífico (Tóquio), Ásia-Pacífico (Seul) e Ásia-Pacífico (Osaka).
- As tabelas globais do MRSC não podem abranger conjuntos de regiões (por exemplo, uma tabela global do MRSC não pode conter réplicas dos conjuntos de regiões dos EUA e da UE).
- Tempo de vida (TTL) não é aceito em tabelas globais do MRSC.
- Índices secundários locais (LSIs) não são aceitos em tabelas globais do MRSC.
- As informações do CloudWatch Contributor Insights são relatadas somente para a região em que uma operação ocorreu.

Escolher um modo de consistência

O principal critério para escolher um modo de consistência multirregional é se seu aplicativo prioriza gravações de menor latência e leituras altamente consistentes ou prioriza uma consistência global forte.

As tabelas globais MREC terão latências de gravação e de leitura altamente consistente mais baixas em comparação às tabelas globais MRSC. As tabelas globais do MREC podem dar suporte a um objetivo de ponto de recuperação (RPO) igual ao atraso de replicação entre as réplicas, geralmente de alguns segundos, dependendo das regiões da réplica.

Você deve usar o modo MREC quando:

- A aplicação pode tolerar dados obsoletos exibidos por operações de leitura altamente consistente se esses dados tiverem sido atualizados em outra região.
- Você prioriza latências de gravação e de leitura altamente consistente mais baixas em relação à consistência de leitura multirregional.
- Sua estratégia de alta disponibilidade multirregional pode tolerar um RPO maior que zero.

As tabelas globais MRSC terão latências de gravação e leitura altamente consistente mais altas em comparação às tabelas globais MREC. As tabelas globais do MRSC oferecem suporte a um objetivo de ponto de recuperação (RPO) de zero.

É necessário usar o modo MRSC quando:

- Você precisa de leituras altamente consistentes em várias regiões.
- Você prioriza a consistência global de leitura em detrimento da menor latência de gravação.
- Sua estratégia de alta disponibilidade multirregional exige um RPO igual a zero.

Monitorar tabelas globais

Tabelas globais configuradas para consistência final em várias regiões (MREC) publicam a métrica `ReplicationLatency` no CloudWatch. Essa métrica controla o tempo decorrido entre quando um item foi gravado em uma tabela-réplica e quando esse item aparece em outra réplica na tabela global. `ReplicationLatency` é expresso em milissegundos e é emitido para cada par de regiões de origem e destino em uma tabela global.

Os valores típicos de `ReplicationLatency` dependem da distância entre as regiões da AWS escolhidas, bem como de outras variáveis como tipo de workload e throughput. Por exemplo, uma réplica de origem na região Oeste dos EUA (N. da Califórnia) (us-west-1) tem uma `ReplicationLatency` menor para a região Oeste dos EUA (Oregon) (us-west-2) em comparação com a região África (Cidade do Cabo) (af-south-1).

Um valor elevado para `ReplicationLatency` pode indicar que as atualizações de uma réplica não se propagaram para outras tabelas-réplica em tempo hábil. Nesse caso, você pode redirecionar temporariamente as atividades de leitura e gravação da aplicação para outra região da AWS.

As tabelas globais configuradas para consistência forte multirregional (MRSC) não publicam uma métrica de `ReplicationLatency`.

Teste de injeção de falhas

As tabelas globais também se integram ao AWS Fault Injection Service (AWS FIS) para realizar experimentos de injeção de falhas em suas workloads de tabelas globais. Isso permite que você teste a resposta do seu aplicativo a um isolamento simulado de região pausando a replicação de e para uma réplica selecionada. Para obter mais informações, consulte pausar a replicação global da tabela.

Vida útil (TTL)

As tabelas globais configuradas para o MREC dão suporte à configuração da exclusão Time To Live (TTL). As configurações de TTL são sincronizadas automaticamente para todas as réplicas em uma tabela global. Quando o TTL exclui um item de uma réplica em uma região, a exclusão é replicada para todas as outras réplicas na tabela global. O TTL não consome capacidade de gravação, portanto, você não é cobrado pela exclusão TTL na região em que a exclusão ocorreu. No entanto, você é cobrado pela exclusão replicada em cada outra região com uma réplica na tabela global.

A replicação de exclusão TTL consome capacidade de gravação nas réplicas nas quais a exclusão está sendo replicada. As réplicas configuradas para capacidade provisionada podem limitar as solicitações de controle de utilização se a combinação de throughput de gravação e throughput de exclusão TTL for maior que a capacidade de gravação provisionada.

As tabelas globais configuradas para consistência forte multirregional (MRSC) não suportam a configuração da exclusão Time To Live (TTL).

Fluxos

As tabelas globais configuradas para consistência final multirregional (MREC) replicam as alterações lendo essas alterações de um DynamoDB Stream em uma tabela de réplica e aplicando essa alteração a todas as outras tabelas de réplica. Portanto, os streams são habilitados por padrão em todas as réplicas em uma tabela global do MREC e não podem ser desativados nessas réplicas. O processo de replicação do MREC pode combinar várias alterações em um curto período de

tempo em uma única gravação replicada, resultando em cada fluxo de réplica contendo registros ligeiramente diferentes. Os registros de streams em réplicas do MREC são sempre ordenados por item, mas a ordenação entre itens pode ser diferente entre as réplicas.

As tabelas globais configuradas para consistência forte multirregionais (MRSC) não usam o DynamoDB Streams para replicação, portanto, o Streams não é habilitado por padrão nas réplicas do MRSC. Você pode habilitar o Streams em uma réplica do MRSC. Os registros de streams nas réplicas do MRSC são idênticos para cada réplica, incluindo a ordenação dos registros do Stream.

Se você quiser escrever um aplicativo que processe registros do Streams para alterações que ocorreram em uma determinada região, mas não em outras em uma tabela global, você pode adicionar um atributo a cada item que define em qual região a alteração desse item ocorreu. Você pode usar esse atributo para filtrar os registros do Streams em busca de alterações que ocorreram em outras regiões, incluindo o uso de filtros de eventos do Lambda para invocar somente as funções do Lambda para alterações em uma região específica.

Transações

Em uma tabela global configurada para MREC, as operações de transação (`TransactWriteItems` e `TransactGetItems`) do DynamoDB são atômicas somente na região em que a operação foi invocada. As gravações transacionais não são replicadas como uma unidade em todas as regiões, o que significa que somente algumas das gravações em uma transação podem ser retornadas por operações de leitura em outras réplicas em um determinado momento.

Por exemplo, se você tiver uma tabela global com réplicas nas regiões Leste dos EUA (Ohio) e Oeste dos EUA (Oregon) e realizar uma operação `TransactWriteItems` na região Leste dos EUA (Ohio), poderá observar transações parcialmente concluídas na região Oeste dos EUA (Oregon) à medida que as alterações forem replicadas. As alterações só serão replicadas para outras regiões quando forem confirmadas na região de origem.

As tabelas globais configuradas para consistência forte multirregional (MRSC) não dão suporte a operações de transação e retornarão um erro se essas operações forem invocadas em uma réplica do MRSC.

Throughput de leitura e gravação

A replicação consome capacidade de gravação. As réplicas configuradas para capacidade provisionada podem limitar as solicitações de controle de utilização se a combinação de throughput de gravação e throughput de replicação for maior que a capacidade de gravação provisionada. A

capacidade de gravação no modo sob demanda é sincronizada para todas as réplicas em uma tabela global. As tabelas globais configuradas para capacidade provisionada sincronizam as configurações de ajuste de escala automático entre réplicas. A configuração real da capacidade de gravação provisionada pode variar entre as réplicas de acordo com o throughput de gravação consumida.

Você pode definir de forma independente as configurações de capacidade de leitura para cada réplica em uma tabela global. Ao adicionar uma réplica a uma tabela global, a capacidade de leitura da tabela ou réplica de origem é usada como valor inicial, a menos que um valor de substituição seja especificado.

Sincronização das configurações

As configurações nas tabelas globais do DynamoDB são parâmetros de configuração que controlam vários aspectos do comportamento e da replicação da tabela. Essas configurações são gerenciadas por meio das APIs do ambiente de gerenciamento do DynamoDB e podem ser configuradas ao criar ou modificar tabelas globais. As tabelas globais sincronizam automaticamente determinadas configurações em todas as réplicas para manter a consistência e, ao mesmo tempo, permitir flexibilidade para otimizações específicas da região. Entender quais configurações são sincronizadas e como elas se comportam ajuda você a configurar sua tabela global de forma eficaz. As configurações se dividem em três categorias principais com base em como elas são sincronizadas entre as réplicas.

As configurações a seguir são sempre sincronizadas entre as réplicas em uma tabela global:

- Modo de capacidade (capacidade provisionada ou sob demanda)
- Capacidade de gravação provisionada da tabela
- Ajuste de escala automático de gravação da tabela
- Definição do Índice Secundário Global (GSI)
- Capacidade de gravação provisionada do GSI
- Ajuste de escala automático de gravação do GSI
- Tipo de criptografia do lado do servidor (SSE)
- Definição de streams no modo MREC
- Vida útil (TTL)
- Throughput a quente
- Throughput máximo de gravação sob demanda

As configurações a seguir são sincronizadas entre as réplicas, mas podem ser substituídas por réplica:

- Capacidade de leitura provisionada da tabela
- Ajuste de escala automático de leitura de tabela
- Capacidade de leitura provisionadas do GSI
- Ajuste de escala automático de leitura do GSI
- Classe de tabela
- Throughput máximo de leitura sob demanda

> **Note**
>
> Os valores de configuração substituíveis são alterados se a configuração for modificada em qualquer outra réplica. Como exemplo, você tem uma tabela global do MREC com réplicas nas regiões Leste dos EUA (Norte da Virgínia) e Oeste dos EUA (Oregon). A réplica do Leste dos EUA (Norte da Virgínia) tem o throughput de leitura provisionada definido para 200 RCUs. A réplica no Oeste dos EUA (Oregon) tem a substituição do throughput de leitura provisionada definido para 100 RCUs. Se você atualizar a configuração de throughput de leitura provisionada na réplica do Leste dos EUA (Norte da Virgínia) de 200 RCUs para 300 RCUs, o novo valor da taxa de transferência de leitura provisionada também será aplicado à réplica no Oeste dos EUA (Oregon). Isso altera a configuração de throughput de leitura provisionada para a réplica do Oeste dos EUA (Oregon) do valor substituído de 100 RCUs para o novo valor de 300 RCUs.

As configurações a seguir nunca são sincronizadas entre as réplicas:

- Proteção contra exclusão
- Recuperação para um ponto no tempo
- Tags
- Habilitação da tabela do CloudWatch Contributor Insights
- Habilitação do GSI do CloudWatch Contributor Insights
- Definição do Kinesis Data Streams
- Políticas de recursos
- Definição de streams no modo MRSC

Todas as outras configurações não são sincronizadas entre as réplicas.

DynamoDB Accelerator (DAX)

As gravações em réplicas de tabelas globais ignoram o DynamoDB Accelerator (DAX), atualizando o DynamoDB diretamente. Como resultado, os caches do DAX podem ficar obsoletos, pois as gravações não estão atualizando o cache do DAX. Os caches do DAX configurados para réplicas de tabelas globais só serão atualizados quando o TTL do cache expirar.

Considerações para gerenciar tabelas globais

Você não pode excluir uma tabela usada para adicionar uma nova réplica de tabela global até que tenham decorrido 24 horas desde que a nova réplica foi criada.

Se você desabilitar uma região da AWS que contém réplicas de tabelas globais, essas réplicas serão convertidas permanentemente em tabelas de região única 20 horas após a desativação da região.

Tutoriais: Criação de tabelas globais

Esta seção fornece instruções passo a passo para criar tabelas globais do DynamoDB configuradas para o modo de consistência preferido. Escolha entre os modos consistência final multirregional (MREC) ou consistência forte multirregional (MRSC) com base nos requisitos do seu aplicativo.

As tabelas globais MREC fornecem menor latência de gravação com consistência final entre Regiões da AWS. As tabelas globais MRSC fornecem leituras altamente consistentes em todas as regiões com latências de gravação ligeiramente maiores do que as do MREC. Escolha o modo de consistência que melhor atenda às necessidades de consistência, latência e disponibilidade dos dados do seu aplicativo.

Tópicos

- Criação de uma tabela global configurada para MREC
- Criação de uma tabela global configurada para MRSC

Criação de uma tabela global configurada para MREC

Esta seção mostra como criar uma tabela global com o modo consistência final multirregional (MREC). MREC é o modo de consistência padrão para tabelas globais e fornece gravações de baixa latência com replicação assíncrona em Regiões da AWS. As alterações feitas em um item em uma região geralmente são replicadas em todas as outras regiões em um segundo. Isso torna o MREC

ideal para aplicativos que priorizam a baixa latência de gravação e podem tolerar breves períodos em que diferentes regiões podem retornar versões de dados ligeiramente diferentes.

Você pode criar tabelas globais MREC com réplicas em qualquer região da AWS que o DynamoDB esteja disponível e adicionar ou remover réplicas a qualquer momento. Os exemplos a seguir mostram como criar uma tabela global MREC com réplicas em várias regiões.

Criação de uma tabela global MREC usando o DynamoDB Console

Siga estas etapas para criar uma tabela global usando o AWS Management Console. O exemplo a seguir cria uma tabela global com tabelas-réplica nos Estados Unidos e na Europa.

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. Para este exemplo, escolha Leste dos EUA (Ohio) no seletor de região na barra de navegação.
3. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
4. Selecione Create Table (Criar tabela).
5. Na página Criar tabela:
   a. Em Table name (Nome da tabela), insira **`Music`**.
   b. Em Partition key, (Chave de partição), insira **`Artist`**.
   c. Em Chave de classificação, insira **`SongTitle`**.
   d. Mantenha o restante das configurações padrão e escolha Criar tabela.

      Essa nova tabela serve como a primeira tabela de réplica em uma nova tabela global. Ela é o protótipo das outras tabelas-réplica que serão adicionadas posteriormente.
6. Depois que a tabela ficar ativa:
   a. Selecione a tabela Music (Música) na lista de tabelas.
   b. Selecione a guia Global Tables (Tabelas globais).
   c. Escolha Create replica (Criar réplica).
7. Na lista suspensa Regiões de replicação disponíveis, escolha US West (Oregon) us-west-2 (Oeste dos EUA [Oregon], us-west-2).

   O console garante que não exista uma tabela com o mesmo nome na região selecionada. Se existir uma tabela com o mesmo nome, será necessário excluir a tabela existente para criar outra tabela-réplica nessa região.

8. Escolha Create replica (Criar réplica). Isso inicia o processo de criação da tabela na região Oeste dos EUA (Oregon), us-west-2.

   A guia Tabelas globais da tabela Music (e de qualquer outra tabela de réplica) mostra que a tabela foi replicada em várias regiões.

9. Adicione outra região repetindo as etapas anteriores, mas escolha Europa (Frankfurt) eu-central-1 como região.

10. Como testar a replicação:

    a. Continue usando o AWS Management Console na região Leste dos EUA (Ohio).

    b. Escolha Explore table items (Explorar itens da tabela).

    c. Selecione Create Item (Criar item).

    d. Insira **`item_1`** para Artist (Artista) e **`Song Value 1`** para SongTitle (Título da música).

    e. Selecione Create Item (Criar item).

11. Verifique a replicação mudando para outras regiões:

    a. No seletor de regiões no canto superior direito, escolha Europa (Frankfurt).

    b. Verifique se a tabela Music (Música) contém o item que você criou.

    c. Repita a verificação para Oeste dos EUA (Oregon).

Criar uma tabela global MREC usando a AWS CLI ou Java

CLI

O exemplo de código a seguir mostra como gerenciar tabelas globais do DynamoDB com consistência final de replicação mutirregional (MREC).

- Crie uma tabela com replicação mutirregional (MREC).
- Insira e obtenha itens de tabelas-réplica.
- Remova as réplicas uma por uma.
- Exclua a tabela para fazer uma limpeza.

AWS CLI com script Bash

Crie uma tabela com replicação mutirregional.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

Descreva a tabela multirregional.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Insira itens em uma tabela-réplica.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Obtenha itens de tabelas-réplica.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Remova as réplicas.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
```

```
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Exclua a tabela para fazer uma limpeza.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem

- UpdateTable

Java

O exemplo de código a seguir mostra como criar e gerenciar tabelas globais do DynamoDB com réplicas entre várias regiões.

- Crie uma tabela com um índice secundário global e DynamoDB Streams.
- Adicione réplicas em diferentes regiões para criar uma tabela global.
- Remova réplicas de uma tabela global.
- Adicione itens de teste para verificar a replicação entre regiões.
- Descreva a configuração da tabela global e o status da réplica.

SDK para Java 2.x

Crie uma tabela com um índice secundário global e DynamoDB Streams usando AWS SDK for Java 2.x.

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
```

```
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
```

```
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Aguarde até que uma tabela se torne ativa usando AWS SDK for Java 2.x.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Adicione uma réplica para criar ou estender uma tabela global usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse addReplica(
```

```
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }

        try {
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
```

```
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Remova uma réplica de uma tabela global usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
```

```
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Adicione itens de teste para verificar a replicação usando AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
```

```
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Descreva a configuração e as réplicas da tabela global usando AWS SDK for Java 2.x.

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
```

```
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Exemplo completo de operações de tabela global usando AWS SDK for Java 2.x.

```
   public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

       String tableName = "Music";
       String indexName = "SongTitleIndex";
       Long readCapacity = 15L;

       // Create DynamoDB client for the source region
       try (DynamoDbClient dynamoDbClient =
           DynamoDbClient.builder().region(sourceRegion).build()) {

           try {
               // Step 1: Create the initial table with GSI and streams
               LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
               createTableWithGSI(dynamoDbClient, tableName, indexName);

               // Step 2: Wait for table to become active
               LOGGER.info("Step 2: Waiting for table to become active");
               waitForTableActive(dynamoDbClient, tableName);

               // Step 3: Add replica in destination region
               LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
               addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

               // Step 4: Wait a moment for replica creation to start
               Thread.sleep(5000);

               // Step 5: Describe table to view replica information
               LOGGER.info("Step 5: Describing table to view replicas");
               describeTable(dynamoDbClient, tableName);

               // Step 6: Add a test item to verify replication
               LOGGER.info("Step 6: Adding test item to verify replication");
               putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

               LOGGER.info("Global table setup completed successfully!");
               LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

               // Step 7: Remove replica and clean up table
```

```
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Criação de uma tabela global configurada para MRSC

Esta seção mostra como criar uma tabela global de consistência forte multirregional (MRSC). As tabelas globais do MRSC replicam de forma síncrona as alterações de itens em todas as regiões, garantindo que as operações de leitura altamente consistentes em qualquer réplica sempre retornem a versão mais recente de um item. Ao converter uma tabela de região única em uma tabela global com MRSC, você deve garantir que a tabela esteja vazia. Não é possível converter uma tabela de região única em uma tabela global com MRSC que já contenha itens. Nenhum dado deve ser gravado na tabela durante o processo de conversão.

Você pode configurar uma tabela global MRSC com três réplicas ou duas réplicas e uma testemunha. Ao criar uma tabela global do MRSC, você escolhe as regiões em que as réplicas

e uma testemunha opcional são implantadas. O exemplo a seguir cria uma tabela global MRSC com réplicas nas regiões Leste dos EUA (Norte da Virgínia) e Leste dos EUA (Ohio), com uma testemunha na região Oeste dos EUA (Oregon).

> **Note**
>
> Antes de criar uma tabela global, verifique se os limites de throughput da cota de serviço são consistentes em todas as regiões de destino, pois isso é necessário para criar uma tabela global. Para ter mais informações sobre os limites globais de throughput de tabelas, consulte as cotas para tabelas globais.

Criação de uma tabela global MRSC usando o DynamoDB Console

Siga estas etapas para criar uma tabela global MRSC usando o AWS Management Console.

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No seletor de regiões na barra de navegação, escolha a região onde as tabelas globais com MRSC são aceitas, por exemplo, **`us-east-2`**.
3. No painel de navegação, selecione Tabelas.
4. Escolha Create table.
5. Na página Criar tabela:
   a. Em Table name (Nome da tabela), insira **`Music`**.
   b. Para a Chave de partição, insira **`Artist`** e mantenha o tipo String padrão.
   c. Em Chave de classificação, insira **`SongTitle`** e mantenha o tipo String padrão.
   d. Mantenha o restante das configurações padrão e escolha Criar tabela.

      Essa nova tabela serve como a primeira tabela de réplica em uma nova tabela global. Ela é o protótipo das outras tabelas-réplica que serão adicionadas posteriormente.
6. Aguarde até que a tabela fique ativa e, em seguida, selecione-a na lista de tabelas.
7. Selecione a guia Tabelas globais e escolha Criar réplica.
8. Na página Criar réplica:
   a. Em Consistência multirregional, escolha Consistência forte.
   b. Para a Região de Replicação 1, escolha **`US East (N. Virginia) us-east-1`**.

c. Para a Região de Replicação 2, escolha **US West (Oregon) us-west-2**.

d. Marque Configurar como Testemunha para a região Oeste dos EUA (Oregon).

e. Escolha Criar réplicas.

9. Aguarde a conclusão do processo de criação da réplica e da testemunha. O status da tabela será exibido como Ativo quando a tabela estiver pronta para uso.

Criar uma tabela global MRSC usando a AWS CLI ou Java

Antes de começar, verifique se a entidade principal do IAM tem as permissões necessárias para criar uma tabela global com MRSC com uma região testemunha.

O seguinte exemplo de política do IAM permite que você crie uma tabela do DynamoDB (`MusicTable`) no Leste dos EUA (Ohio) com uma réplica no Leste dos EUA (Norte da Virgínia) e uma região testemunha no Oeste dos EUA (Oregon):

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:DeleteTable",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DeleteGlobalTableWitness",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
```

```
                "arn:aws:dynamodb:us-west-2:123456789123:table/MusicTable",
                "arn:aws:dynamodb:us-east-1:123456789123:table/MusicTable",
                "arn:aws:dynamodb:us-east-2:123456789123:table/MusicTable"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "replication.dynamodb.amazonaws.com"
                }
            }
        }
    ]
}
```

O exemplo de código a seguir mostra como criar e gerenciar tabelas globais do DynamoDB com consistência forte multirregional (MRSC).

- Crie uma tabela com consistência forte multirregional.
- Verifique a configuração de MRSC e o status da réplica.
- Teste a consistência forte entre regiões com leituras imediatas.
- Faça gravações condicionais com garantia de MRSC.
- Limpe os recursos da tabela global com MRSC.

Bash

AWS CLI com script Bash

Crie uma tabela com consistência forte multirregional.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
```

```
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Verifique a configuração de MRSC e o status da réplica.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Teste a consistência forte com leituras imediatas entre regiões.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
```

```
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Faça gravações condicionais com garantia de MRSC.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Limpe os recursos da tabela global com MRSC.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK para Java 2.x

Crie uma tabela regional pronta para conversão MRSC usando AWS SDK for Java 2.x.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
```

```
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Converta uma tabela regional para MRSC com réplicas e testemunhas usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
```

```
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
```

```
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Descreva uma configuração de tabela global do MRSC usando AWS SDK for Java 2.x.

```
    public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing MRSC global table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
```

```
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            if (response.table().globalTableWitnesses() != null
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Adicione itens de teste para verificar a forte consistência do MRSC usando AWS SDK for Java 2.x.

```
   public static PutItemResponse putTestItem(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final String artist,
       final String songTitle,
       final String album,
       final String year) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Adding test item to MRSC global table: " + tableName);

           Map<String, AttributeValue> item = new HashMap<>();
           item.put("Artist", AttributeValue.builder().s(artist).build());
           item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           if (album != null && !album.trim().isEmpty()) {
               item.put("Album", AttributeValue.builder().s(album).build());
           }
           if (year != null && !year.trim().isEmpty()) {
               item.put("Year", AttributeValue.builder().n(year).build());
           }

           PutItemRequest putItemRequest =
               PutItemRequest.builder().tableName(tableName).item(item).build();

           PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
           LOGGER.info("Test item added successfully with strong consistency");
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Leia itens com leituras consistentes de réplicas do MRSC usando AWS SDK for Java 2.x.

```
    public static GetItemResponse getItemWithConsistentRead(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            GetItemRequest getItemRequest = GetItemRequest.builder()
```

```
                .tableName(tableName)
                .key(key)
                .consistentRead(true)
                .build();

            GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

            if (response.hasItem()) {
                LOGGER.info("Item found with strong consistency - no wait time
needed");
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Faça atualizações condicionais com garantias de MRSC usando AWS SDK for Java 2.x.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
```

```
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
```

```
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Aguarde até que as réplicas e testemunhas do MRSC se tornem ativas usando AWS SDK for Java 2.x.

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
```

```
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
```

```
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
```

```
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Limpe as réplicas e testemunhas do MRSC usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
```

```
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Demonstração completa do fluxo de trabalho do MRSC usando AWS SDK for Java 2.x.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
```

```
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);
```

```
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
```

```
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.

- CreateTable
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- UpdateItem
- UpdateTable

Segurança de tabelas globais do DynamoDB

As réplicas de tabelas globais são tabelas do DynamoDB, então você usa os mesmos métodos para controlar o acesso às réplicas que usa para tabelas de região única, incluindo políticas de identidade e políticas baseadas em recursos do Identity and Access Management (IAM) da AWS.

Como as tabelas globais usam o IAM da AWS

Função vinculada ao serviço de replicação

Na primeira vez que você cria uma tabela global, o Amazon DynamoDB cria automaticamente para você uma função vinculada ao serviço de Identity and Access Management (IAM) da AWS. O SLR é chamado de `AWSServiceRoleForDynamoDBReplication` e permite que o DynamoDB gerencie a replicação em tabelas globais entre regiões em seu nome.

Ao aplicar políticas baseadas em recursos às réplicas, não negue nenhuma das permissões definidas na política `AWSServiceRoleForDynamoDBReplication` à entidade principal da SLR, pois isso interromperá a replicação. Se você negar as permissões SLR necessárias, a replicação de e para as réplicas afetadas será interrompida e o status da tabela de réplicas mudará para `REPLICATION_NOT_AUTHORIZED`.

- Se uma réplica em uma tabela global configurada para consistência final multirregional (MREC) permanecer no estado `REPLICATION_NOT_AUTHORIZED` por mais de 20 horas, a réplica será convertida irreversivelmente em uma tabela do DynamoDB de região única.
- Réplicas de tabelas globais configuradas para consistência forte multirregional (MRSC) retornam um `AccessDeniedException` para operações de gravação e leitura altamente consistentes. Se uma réplica em uma tabela global do MRSC permanecer no estado `REPLICATION_NOT_AUTHORIZED` por mais de sete dias, a réplica ficará permanentemente

inacessível e as operações de gravação e leitura altamente consistentes continuarão falhando com um erro. Algumas operações de gerenciamento, como a exclusão de réplicas, serão bem-sucedidas.

Função vinculada a serviço do IAM para ajuste de escala automático

Ao configurar uma tabela global para o modo de capacidade provisionada, você também deve configurar o ajuste de escala automático para a tabela global. O ajuste de escala automático do Amazon DynamoDB usa o serviço de ajuste de escala automático de aplicativo da AWS para ajustar dinamicamente a capacidade de throughput em suas réplicas de tabela global. O serviço de ajuste de escala automático cria uma função vinculada ao serviço (SLR) chamada de `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable` para gerenciar a capacidade da tabela provisionada e criar alarmes do CloudWatch. Ao aplicar políticas baseadas em recursos às réplicas, não negue nenhuma das permissões definidas na política `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable` à entidade principal do serviço SLR de ajuste automático do aplicativo, pois isso interromperá a funcionalidade de ajuste de escala automático.

Para obter mais informações sobre funções vinculadas a serviços, consulte Usando funções vinculadas a serviços no Guia do Usuário do IAM.

Permissões obrigatórias para tabelas globais

Para criar uma réplica, você deve ter a seguinte permissão na tabela ou réplica à qual você está adicionando novas réplicas:

- `dynamodb:UpdateTable`

Para criar uma réplica, você deve ter as seguintes permissões em cada região onde uma nova réplica será criada:

- `dynamodb:CreateTable`
- `dynamodb:CreateTableReplica`

Para criar uma testemunha, você deve ter a seguinte permissão na região em que uma nova testemunha será criada:

- `dynamodb:CreateGlobalTableWitness`

Para excluir uma réplica, é necessário ter as permissões a seguir na réplica:

- `dynamodb:DeleteTable`
- `dynamodb:DeleteTableReplica`

Para excluir uma testemunha, você precisa ter as seguintes permissões na testemunha:

- `dynamodb:DeleteGlobalTableWitness`

Para atualizar a política de ajuste de escala automático da réplica usando a API `UpdateTableReplicaAutoScaling`, é necessário ter as permissões a seguir em todas as regiões que contenham réplicas:

- `application-autoscaling:DeleteScalingPolicy`
- `application-autoscaling:DeleteScheduledAction`
- `application-autoscaling:DeregisterScalableTarget`
- `application-autoscaling:DescribeScalableTargets`
- `application-autoscaling:DescribeScalingActivities`
- `application-autoscaling:DescribeScalingPolicies`
- `application-autoscaling:DescribeScheduledActions`
- `application-autoscaling:PutScalingPolicy`
- `application-autoscaling:PutScheduledAction`
- `application-autoscaling:RegisterScalableTarget`

Para atualizar as configurações de Time to Live com a `UpdateTimeToLive` API, você deve ter a seguinte permissão em todas as réplicas:

- `dynamodb:UpdateTimeToLive`

Políticas de exemplo do IAM

Exemplo: gerenciar uma tabela global

A política do IAM a seguir concede permissões para criar e excluir réplicas e testemunhas de tabelas globais para a tabela “usuários” em três regiões:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AllowManagingUsersGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness",
                "dynamodb:DeleteTable",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DeleteGlobalTableWitness",
                "dynamodb:UpdateTable",
                "dynamodb:UpdateTimeToLive"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/users",
                "arn:aws:dynamodb:us-east-2:123456789012:table/users",
                "arn:aws:dynamodb:us-west-2:123456789012:table/users"
            ]
        }
    ]
}
```

Exemplo: excluir as permissões SLR necessárias das políticas de negação de caracteres curinga

Uma política do IAM com a condição a seguir não afeta as permissões necessárias para a SLR de replicação e a SLR de ajuste de escala automático da AWS do DynamoDB. Essa condição pode ser adicionada a políticas amplamente restritivas para evitar a interrupção involuntária da replicação ou do ajuste de escala automático:

```
"Condition": {
    "StringNotEquals": {
        "aws:PrincipalArn": [
            "arn:aws:iam::YOUR_ACCOUNT_ID:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "arn:aws:iam::YOUR_ACCOUNT_ID:role/aws-service-role/dynamodb.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable"
```

```
        ]
    }
}
```

Como as tabelas globais usam AWS KMS

Como todas as tabelas do DynamoDB, as réplicas de tabelas globais sempre criptografam dados em repouso usando chaves de criptografia armazenadas no AWS Key Management Service (AWS KMS).

Todas as réplicas em uma tabela global devem ser configuradas com o mesmo tipo de chave KMS (chave própria da AWS, chave gerenciada da AWS ou chave gerenciada pelo cliente).

> **Important**
>
> O DynamoDB exige acesso à chave de criptografia da réplica para excluir uma réplica. Se você quiser desativar ou excluir uma chave gerenciada pelo cliente usada para criptografar uma réplica porque você está excluindo a réplica, primeiro exclua a réplica, aguarde até que o status da tabela em uma das réplicas restantes mude para `ACTIVE`, então desative ou exclua a chave.

Para uma tabela global configurada para consistência final multirregional (MREC), se você desativar ou revogar o acesso do DynamoDB a uma chave gerenciada pelo cliente usada para criptografar uma réplica, a replicação de e para a réplica será interrompida e o status da réplica mudará para `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Se uma réplica em uma tabela global do MREC permanecer no estado `INACCESSIBLE_ENCRYPTION_CREDENTIALS` por mais de 20 horas, a réplica será convertida irreversivelmente em uma tabela do DynamoDB de região única.

Para uma tabela global configurada para consistência forte multirregional (MRSC), se você desativar ou revogar o acesso do DynamoDB a uma chave gerenciada pelo cliente usada para criptografar uma réplica, a replicação de e para a réplica será interrompida, as tentativas de realizar gravações ou leituras altamente consistentes na réplica retornarão um erro, e o status da réplica mudará para `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Se uma réplica em uma tabela global do MRSC permanecer no estado `INACCESSIBLE_ENCRYPTION_CREDENTIALS` por mais de sete dias, dependendo das permissões específicas revogadas, a réplica será arquivada ou ficará permanentemente inacessível.

Conceitos básicos sobre faturamento do Amazon DynamoDB para tabelas globais

Este guia descreve como o faturamento do DynamoDB funciona para tabelas globais, identificando os componentes que contribuem com o custo das tabelas globais, incluindo um exemplo prático.

As tabelas globais do Amazon DynamoDB são um banco de dados totalmente gerenciado, multirregional e sem servidor. As tabelas globais oferecem disponibilidade de 99,999%, entregando maior resiliência das aplicações e melhor continuidade dos negócios. As tabelas globais replicam as tabelas do DynamoDB automaticamente nas regiões da AWS de sua escolha, portanto, é possível ter alta performance e rapidez de leitura e gravação locais.

Como funciona

O modelo de faturamento para tabelas globais é diferente do aplicado a tabelas de região única do DynamoDB. As operações de gravação para tabelas do DynamoDB de região única são cobradas usando as seguintes unidades:

- Unidades de solicitação de gravação (WRUs) para o modo de capacidade sob demanda, em que uma WRU é cobrada por cada gravação de até 1 KB
- Unidades de capacidade de gravação (WCUs) para o modo de capacidade provisionada, em que uma WCU fornece uma gravação por segundo para até 1 KB

Quando você cria uma tabela global adicionando uma tabela de réplica a uma tabela de região única existente, essa tabela de região única se torna uma tabela de réplica, o que significa que as unidades usadas para faturar as gravações na tabela também mudam. As operações de gravação em tabelas de réplica são cobradas usando as seguintes unidades:

- Unidades de solicitação de gravação replicadas (rWRUs) para o modo de capacidade sob demanda, em que uma rWRU por tabela de réplica é cobrada por cada gravação de até 1 KB
- Unidades de capacidade de gravação replicadas (rWCUs) para o modo de capacidade provisionada, em que uma WCU por tabela de réplica fornece uma gravação por segundo para até 1 KB

As atualizações dos índices secundários globais (GSIs) são cobradas usando as mesmas unidades das tabelas do DynamoDB de região única, mesmo que a tabela base do GSI seja uma tabela de réplica. As operações de atualização de GSIs são cobradas usando as seguintes unidades:

- Unidades de solicitação de gravação (WRUs) para o modo de capacidade sob demanda, em que uma WRU é cobrada por cada gravação de até 1 KB
- Unidades de capacidade de gravação (WCUs) para o modo de capacidade provisionada, em que uma WCU fornece uma gravação por segundo para até 1 KB

As unidades de gravação replicadas (rWCUs e rWRUs) têm o mesmo preço das unidades de gravação de região única (WCUs e WRUs). Além disso, as taxas de transferência de dados entre regiões se aplicam às tabelas globais, pois os dados são replicados entre regiões. As cobranças de gravação replicada (rWCU ou rWRU) são geradas em cada região que contém uma tabela de réplica para a tabela global.

As operações de leitura de tabelas de região única e de tabelas de réplica usam as seguintes unidades:

- Unidades de solicitação de leitura (RRUs) para o modo de capacidade sob demanda, em que uma RRU é cobrada por cada leitura altamente consistente de até 4 KB
- Unidades de capacidade de leitura (RCUs) para tabelas provisionadas, em que uma RCU fornece uma leitura altamente consistente por segundo de até 4 KB

Modos de consistência e cobrança

As unidades de gravação replicadas (rWCUs e rWRUs) usadas para cobrar as operações de gravação são idênticas nos modos de consistência forte multirregional (MRSC) e consistência final multirregional (MREC). As tabelas globais que usam o modo de consistência forte multirregional (MRSC) configurado com testemunha não incorrem em custos de unidades de gravação replicadas (rWCUs e rWRUs), custos de armazenamento ou custos de transferência de dados para replicação para a testemunha.

Exemplo de faturamento para tabelas globais do DynamoDB

Vamos analisar um exemplo de cenário de vários dias para ver como o faturamento de solicitações de gravação de tabela global funciona na prática (observe que este exemplo leva em conta apenas solicitações de gravação e não inclui as cobranças de restauração de tabelas e transferência de dados entre regiões que seriam geradas no exemplo):

Dia 1: tabela de região única: você tem uma tabela de região única sob demanda do DynamoDB chamada Tabela_A na região us-west-2. Você grava cem itens de 1 KB na Tabela_A. Com relação a

operações de gravação de região única, você paga uma unidade de solicitação de gravação (WRU) por 1 KB gravado. Suas cobranças do dia 1 são:

- 100 WRUs na região us-west-2 para gravações em região única

O total de unidades de solicitação cobradas no dia 1: 100 WRUs.

Dia 2: criar uma tabela global: você cria uma tabela global adicionando uma réplica à Tabela_A na região us-east-2. Agora, a Tabela_A é uma tabela global com duas tabelas de réplica; uma na região us-west-2 e uma na região us-east-2. Você grava 150 itens de 1 KB na tabela de réplica na região us-west-2. Suas cobranças do dia 2 são:

- 150 rWRUs na região us-west-2 para gravações replicadas
- 150 rWRUs na região us-east-2 para gravações replicadas

O total de unidades de solicitação cobradas no dia 2: 300 rWRUs.

Dia 3: adicionar um índice secundário global: você adiciona um índice secundário global (GSI) à tabela de réplica na região us-east-2 que projeta todos os atributos da tabela base (réplica). A tabela global cria automaticamente o GSI na tabela de réplica na região us-west-2. Você grava 200 novos registros de 1 KB na tabela de réplica na região us-west-2. Suas cobranças do dia 3 são:

- • 200 rWRUs na região us-west-2 para gravações replicadas
- • 200 WRUs na região us-west-2 para atualizações de GSI
- • 200 rWRUs na região us-east-2 para gravações replicadas
- • 200 WRUs na região us-east-2 para atualizações de GSI

Total de unidades de solicitação de gravação cobradas no dia 3: 400 WRUs e 400 rWRUs.

O total de cobranças de unidade de gravação nos três dias equivale a 500 WRUs (100 WRUs no dia 1 + 400 WRUs no dia 3) e 700 rWRUs (300 rWRUs no dia 2 + 400 rWRUs no dia 3).

Em resumo, as operações de gravação da tabela de réplica são faturadas em unidades de gravação replicadas em todas as regiões que contêm uma tabela de réplica. Se você tiver índices secundários globais, serão cobradas unidades de gravação pelas atualizações dos GSIs em todas as regiões que contêm um GSI (que em uma tabela global são todas as regiões que contêm uma tabela de réplica).

Versões de tabelas globais do DynamoDB

Há duas versões disponíveis das tabelas globais do DynamoDB: tabelas globais versão 2019.11.21 (atual) e tabelas globais versão 2017.11.29 (legada). Recomendamos usar a versão 2019.11.21 (atual) das tabelas globais, pois ela é mais fácil de usar, com suporte em mais regiões e oferece menor custo para a maioria das workloads em comparação com a versão 2017.11.29 (legada).

Determinar a versão de uma tabela global

Determinar a versão usando a AWS CLI

Identificar uma réplica de tabela global da versão 2019.11.21 (atual)

Para determinar se uma tabela é uma réplica da versão 2019.11.21 (atual) das tabelas globais, invoque o comando `describe-table` para a tabela. Se a saída contiver o atributo `GlobalTableVersion` com o valor "2019.11.21", a tabela é uma réplica da tabela global da versão 2019.11.21 (atual).

Um exemplo de comando CLI para `describe-table`:

```
aws dynamodb describe-table \
--table-name users \
--region us-east-2
```

A saída (abreviada) contém o atributo `GlobalTableVersion` com o valor "2019.11.21", então essa tabela é uma réplica da tabela global da versão 2019.11.21 (atual).

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
            },
            {
                "AttributeName": "name",
                "AttributeType": "S"
            }
        ],
        "TableName": "users",
        ...
        "GlobalTableVersion": "2019.11.21",
```

```
        "Replicas": [
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE",
            }
        ],
        ...
    }
}
```

Identificar uma réplica de tabela global da versão 2017.11.29 (legada)

As tabelas globais versão 2017.11.29 (legada) usam um conjunto dedicado de comandos para gerenciamento global de tabelas. Para determinar se uma tabela é uma réplica da versão 2017.11.21 (legada) das tabelas globais, invoque o comando `describe-global-table` para a tabela. Se você receber uma resposta bem-sucedida, a tabela será uma réplica de tabela global da versão 2017.11.29 (legada). Se o comando `describe-global-table` retornar um erro `GlobalTableNotFoundException`, a tabela não será uma réplica da versão 2017.11.29 (legada).

Um exemplo de comando CLI para `describe-global-table`:

```
aws dynamodb describe-global-table \
--table-name users \
--region us-east-2
```

O comando retorna uma resposta bem-sucedida, então essa tabela será uma réplica de tabela global da versão 2017.11.29 (legada).

```
{
    "GlobalTableDescription": {
        "ReplicationGroup": [
            {
                "RegionName": "us-west-2"
            },
            {
                "RegionName": "us-east-2"
            }
        ],
        "GlobalTableArn": "arn:aws:dynamodb::123456789012:global-table/users",
        "CreationDateTime": "2025-06-10T13:55:53.630000-04:00",
        "GlobalTableStatus": "ACTIVE",
        "GlobalTableName": "users"
```

```
    }
}
```

Determinar a versão usando o DynamoDB Console

Para identificar a versão de uma réplica de tabela global, faça o seguinte:

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Selecione a tabela para a qual você deseja identificar a versão das tabelas globais.
4. Selecione a guia Global Tables (Tabelas globais).

   A seção Resumo exibe a versão das tabelas globais em uso.

Diferenças de comportamento entre as versões herdada e atual

A lista a seguir descreve as diferenças de comportamento entre as versões herdada e atual das tabelas globais.

- A versão 2019.11.21 (atual) consome menor capacidade de gravação para várias operações do DynamoDB em comparação à versão 2017.11.29 (herdada) e, portanto, é mais econômica para a maioria dos clientes. As diferenças dessas operações do DynamoDB são as seguintes:
  - Invocar PutItem para um item de 1 KB em uma região e replicar para outras regiões requer 2 rWRUs por região para a 2017.11.29 (herdada), mas apenas 1 rWRU para a 2019.11.21 (atual).
  - Invocar UpdateItem para um item de 1 KB requer 2 rWRUs na região de origem e 1 rWRU por região de destino para a 2017.11.29 (herdada), mas apenas 1 rWRU para as regiões de origem e de destino para a 2019.11.21 (atual).
  - Invocar DeleteItem para um item de 1 KB requer 1 rWRU na região de origem e 2 rWRUs por região de destino para a 2017.11.29 (herdada), mas apenas 1 rWRU para as regiões de origem e de destino para a 2019.11.21 (atual).

  A tabela a seguir mostra o consumo de rWRU das tabelas das versões 2017.11.29 (herdada) e 2019.11.21 (atual) de um item de 1 KB em duas regiões.

Operação	2017.11.29 (herdada)	2019.11.21 (atual)	Economia
PutItem	4 rWRUs	2 rWRUs	50%

Operação	2017.11.29 (herdada)	2019.11.21 (atual)	Economia
UpdateItem	3 rWRUs	2 rWRUs	33%
DeleteItem	3 rWRUs	2 rWRUs	33%

- A versão 2017.11.29 (herdada) está disponível somente em 11 Regiões da AWS. No entanto, a versão 2019.11.21 (atual) está disponível em todas as Regiões da AWS.
- Você deve criar tabelas globais da versão 2017.11.29 (herdada) criando primeiro um conjunto de tabelas regionais vazias e, depois, invocando a API CreateGlobalTable para formar a tabela global. As tabelas globais da versão 2019.11.21 (atual) são criadas invocando a API UpdateTable para adicionar uma réplica a uma tabela regional existente.
- A versão 2017.11.29 (herdada) exige que você esvazie todas as réplicas na tabela antes de adicionar uma réplica em uma nova região (inclusive durante a criação). A versão 2019.11.21 (atual) aceita adicionar réplicas a regiões e removê-las em uma tabela que já contenha dados.
- A versão 2017.11.29 (herdada) usa o seguinte conjunto dedicado de APIs de ambiente de gerenciamento para gerenciar réplicas:
  - CreateGlobalTable
  - DescribeGlobalTable
  - DescribeGlobalTableSettings
  - ListGlobalTables
  - UpdateGlobalTable
  - UpdateGlobalTableSettings

  A versão 2019.11.21 (atual) usa as APIs DescribeTable e UpdateTable para gerenciar réplicas.
- A versão 2017.11.29 (herdada) publica dois registros do DynamoDB Streams para cada gravação. A versão 2019.11.21 (atual) publica apenas um registro do DynamoDB Streams para cada gravação.
- A versão 2017.11.29 (herdada) preenche e atualiza os atributos `aws:rep:deleting`,`aws:rep:updateregion` e `aws:rep:updatetime`. A versão 2019.11.21 (atual) não preenche nem atualiza esses atributos.
- A versão 2017.11.29 (herdada) não sincroniza as configurações Usar a vida útil (TTL) no DynamoDB entre réplicas. A versão 2019.11.21 (atual) sincroniza as configurações de TTL entre réplicas.

- A versão 2017.11.29 (herdada) não replica exclusões de TTL para outras réplicas. A versão 2019.11.21 (atual) replica exclusões de TTL para todas as réplicas.
- A versão 2017.11.29 (herdada) não sincroniza configurações de ajuste de escala automático entre réplicas. A versão 2019.11.21 (atual) sincroniza as configurações de ajuste de escala automático entre réplicas.
- A versão 2017.11.29 (herdada) não sincroniza as configurações de índice secundário global (GSI) entre réplicas. A versão 2019.11.21 (atual) sincroniza as configurações do GSI entre réplicas.
- A versão 2017.11.29 (herdada) não sincroniza as configurações de criptografia em repouso entre réplicas. A versão 2019.11.21 (atual) sincroniza as configurações de criptografia em repouso entre réplicas.
- A versão 2017.11.29 (herdada) publica a métrica `PendingReplicationCount`. A versão 2019.11.21 (atual) não publica essa métrica.

Atualizar para a versão atual

Permissões obrigatórias para a atualização de tabelas globais

Para realizar a atualização para a versão 2019.11.21 (atual), é necessário ter permissões `dynamodb:UpdateGlobalTableversion` em todas as regiões com réplicas. Essas permissões se somam às permissões necessárias para acessar o console do DynamoDB e visualizar tabelas.

A política do IAM a seguir concede permissões para atualizar qualquer tabela global para a versão 2019.11.21 (atual).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

A política do IAM a seguir concede permissões para atualizar apenas a tabela global `Music`, com réplicas em duas regiões, para a versão 2019.11.21 (atual).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Music",
                "arn:aws:dynamodb:ap-southeast-1:123456789012:table/Music",
                "arn:aws:dynamodb:us-east-2:123456789012:table/Music"
            ]
        }
    ]
}
```

O que esperar durante a atualização

- Todas as réplicas de tabelas globais continuarão processando o tráfego de leitura e gravação durante a atualização.
- O processo de atualização requer de alguns minutos a várias horas, dependendo do tamanho da tabela e do número de réplicas.
- Durante o processo de atualização, o valor de TableStatus mudará de `ACTIVE` para `UPDATING`. É possível ver o status da tabela invocando a API DescribeTable ou com a visualização Tablesno .
- O ajuste de escala automático não alterará as configurações de capacidade provisionada para uma tabela global enquanto a tabela estiver sendo atualizada. É altamente recomendável definir a tabela como o modo de capacidade sob demanda durante a atualização.
- Se você optar por usar o modo de capacidade provisionada com ajuste de escala automático durante a atualização, será necessário aumentar o throughput mínimo de leitura e gravação em suas políticas para atender aos aumentos esperados no tráfego e evitar o controle de utilização durante a atualização.
- A métrica `ReplicationLatency` pode relatar temporariamente picos de latência ou parar de relatar dados de métricas durante o processo de atualização. Consulte mais informações em the section called “ReplicationLatency”.
- Quando o processo de atualização estiver concluído, o status da tabela mudará para `ACTIVE`.

Comportamento do DynamoDB Streams antes, durante e depois da atualização

Operação	Região de réplica	Comportamento antes da atualização	Comportamento durante a atualização	Comportamento após a atualização
Inserir ou atualizar	Origem	O preenchimento do carimbo de data e hora é feito por meio de UpdateItem.	O preenchimento do carimbo de data e hora é feito por meio de PutItem.	Nenhum carimbo de data e hora visível para o cliente é gerado.
		Dois registros do Streams são gerados. O primeiro contém os atributos gravados pelo cliente. O segundo contém os atributos `aws:rep:*` .	Dois registros do Streams são gerados. O primeiro contém os atributos gravados pelo cliente. O segundo contém os atributos `aws:rep:*` .	Um único registro do Streams é gerado contendo os atributos gravados pelo cliente.
		Duas rWCUs são consumidas para cada gravação do cliente.	Duas rWCUs são consumidas para cada gravação do cliente.	Uma rWCU é consumida para cada gravação do cliente.
		As métricas `ReplicationLatency` e `PendingReplicationCount` são publicadas no CloudWatch.	As métricas `ReplicationLatency` e `PendingReplicationCount` são publicadas no CloudWatch.	A métrica `ReplicationLatency` é publicada no CloudWatch.

Operação	Região de réplica	Comportamento antes da atualização	Comportamento durante a atualização	Comportamento após a atualização
	Destination (Destino)	A replicação acontece usando PutItem.	A replicação acontece usando PutItem.	A replicação acontece usando PutItem.
		É gerado um único registro do Streams, que contém os atributos gravados pelo cliente e os atributos `aws:rep:*` .	É gerado um único registro do Streams, que contém os atributos gravados pelo cliente e os atributos `aws:rep:*` .	É gerado um único registro do Streams, que contém somente os atributos gravados pelo cliente e nenhum atributo de replicação.
		Uma rWCU será consumida se o item existir na região de destino. Duas rWCUs serão consumidas se o item não existir na região de destino.	Uma rWCU será consumida se o item existir na região de destino. Duas rWCUs serão consumidas se o item não existir na região de destino.	Uma rWCU é consumida para cada gravação do cliente.
		As métricas `ReplicationLatency` e `PendingReplicationCount` são publicadas no CloudWatch.	As métricas `ReplicationLatency` e `PendingReplicationCount` são publicadas no CloudWatch.	A métrica `ReplicationLatency` é publicada no CloudWatch.

Operação	Região de réplica	Comportamento antes da atualização	Comportamento durante a atualização	Comportamento após a atualização
Excluir	Origem	Exclua qualquer item com carimbo de data e hora menor usando DeleteItem.	Exclua qualquer item com carimbo de data e hora menor usando DeleteItem.	Exclua qualquer item com carimbo de data e hora menor usando DeleteItem.
		É gerado um único registro do Streams, que contém os atributos gravados pelo cliente e os atributos `aws:rep:*` .	É gerado um único registro do Streams, que contém os atributos gravados pelo cliente e os atributos `aws:rep:*` .	É gerado um único registro do Streams, que contém os atributos gravados pelo cliente.
		Uma rWCU é consumida para cada exclusão do cliente.	Uma rWCU é consumida para cada exclusão do cliente.	Uma rWCU é consumida para cada exclusão do cliente.
		As métricas `ReplicationLatency` e `PendingReplicationCount` são publicadas no CloudWatch.	As métricas `ReplicationLatency` e `PendingReplicationCount` são publicadas no CloudWatch.	A métrica `ReplicationLatency` é publicada no CloudWatch.

Operação	Região de réplica	Comportamento antes da atualização	Comportamento durante a atualização	Comportamento após a atualização
	Destination (Destino)	Ocorrem exclusões em duas fases: • Na Fase 1, UpdateItem define o sinalizador de exclusão. • Na Fase 2, DeleteItem exclui o item.	Exclui o item usando DeleteItem.	Exclui o item usando DeleteItem.
		Dois registros do Streams são gerados. O primeiro registro contém a alteração no campo `aws:rep:deleting`. O segundo registro contém os atributos gravados pelo cliente e os atributos `aws:rep:*`.	É gerado um único registro do Streams, que contém os atributos gravados pelo cliente.	É gerado um único registro do Streams, que contém os atributos gravados pelo cliente.

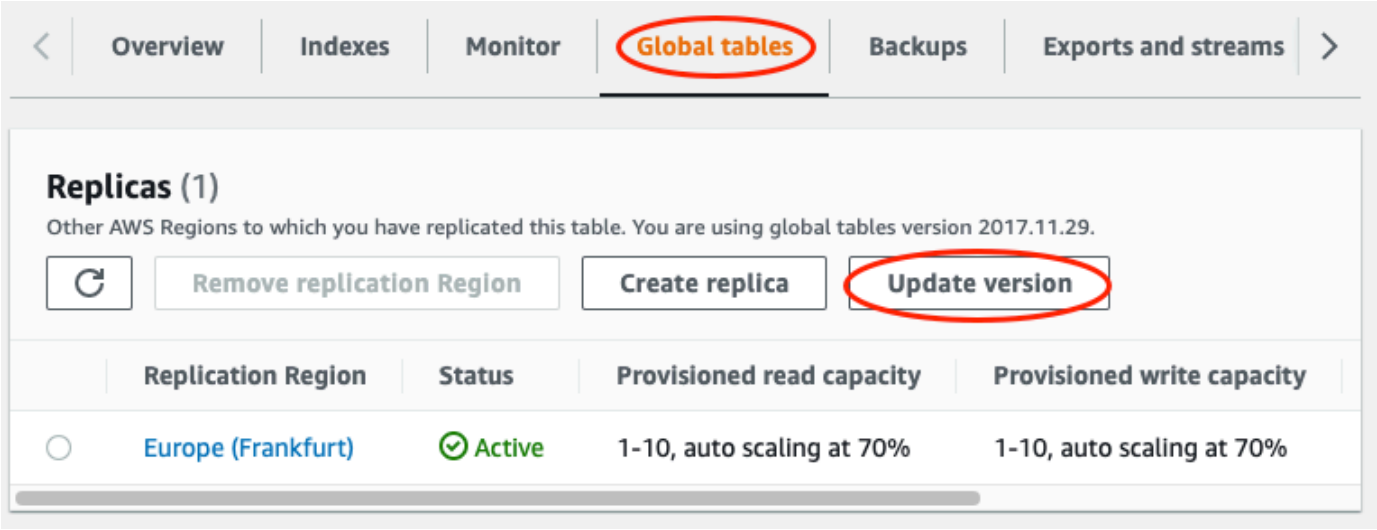
Operação	Região de réplica	Comportamento antes da atualização	Comportamento durante a atualização	Comportamento após a atualização
		Duas rWCUs são consumidas para cada exclusão do cliente.	Uma rWCU é consumida para cada exclusão do cliente.	Uma rWCU é consumida para cada exclusão do cliente.
		As métricas `ReplicationLatency` e `PendingReplicationCount` são publicadas no CloudWatch.	A métrica `ReplicationLatency` é publicada no CloudWatch.	A métrica `ReplicationLatency` é publicada no CloudWatch.

Atualizar para a versão 2019.11.21 (atual)

Siga estas etapas para atualizar a versão das tabelas globais do DynamoDB usando o AWS Management Console.

Como atualizar tabelas globais para a versão 2019.11.21 (atual)

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação no lado esquerdo do console, escolha Tabelas e, depois, selecione a tabela global que você deseja atualizar para a versão 2019.11.21 (atual).
3. Selecione a guia Global Tables (Tabelas globais).
4. Escolha Update version (Atualizar versão).



5. Leia e concorde com os novos requisitos e escolha Update version (Atualizar versão).
6. Após a conclusão do processo de atualização, a versão de tabelas globais exibida no console muda para 2019.11.21.

Melhores práticas para tabelas globais

As seções a seguir descrevem as melhores práticas para implantar e usar tabelas globais.

Versão

Há duas versões disponíveis das tabelas globais do DynamoDB: versão 2019.11.21 (atual) e versão 2017.11.29 (legada). Você deve usar a versão 2019.11.21 (atual) sempre que possível.

Proteção contra exclusão

Você deve ativar a proteção contra exclusão nas réplicas de tabelas globais que deseja proteger contra exclusão acidental. Você deve ativar a proteção contra exclusão em cada réplica.

Usar o AWS CloudFormation

O AWS CloudFormation atualmente não oferece suporte à coordenação de recursos multirregionais, como tabelas globais entre pilhas. Se você definir cada réplica de uma tabela global em uma pilha regional separada, encontrará erros devido ao desvio detectado entre as pilhas ao realizar atualizações de réplicas. Para evitar esse problema, você deve escolher uma região como a região de referência para implantar suas tabelas globais e definir todas as réplicas da tabela global na pilha dessa região.

> **Important**
>
> Você não pode alterar o tipo de um recurso no seu modelo para converter um recurso `AWS::DynamoDB::Table` em um recurso `AWS::DynamoDB::GlobalTable`. Tentar converter uma tabela de região única em uma tabela global alterando o tipo de recurso do CloudFormation pode resultar na exclusão da tabela do DynamoDB.

Em vez disso, você pode utilizar o recurso `AWS::DynamoDB::GlobalTable` para criar uma tabela em uma única região. Essa tabela será implantada como qualquer outra tabela de região única. Se você atualizar a pilha posteriormente para adicionar outras regiões a um recurso, as réplicas serão adicionadas à tabela e ela será convertida com segurança em uma tabela global.

Se você tiver um recurso `AWS::DynamoDB::Table` existente que deseja converter em um recurso `AWS::DynamoDB::GlobalTable`, as etapas recomendadas para converter o tipo de recurso são:

1. Defina a política de exclusão `AWS::DynamoDB::Table` a ser mantida.
2. Remova a tabela da definição de pilha.
3. Adicione réplicas à tabela de região única no console da AWS, convertendo-a em uma tabela global.
4. Importe a tabela global como um novo recurso `AWS::DynamoDB::GlobalTable` para a pilha.

Backups e recuperação pontual

Habilitar backups automatizados e recuperação para um ponto no tempo (PITR) para uma réplica em uma tabela global pode ser suficiente para atingir seus objetivos de recuperação de desastres. Os backups de réplicas criados com backup para AWS podem ser replicados automaticamente em todas as regiões para maior resiliência. Considere as metas do seu plano de recuperação de desastres no contexto da alta disponibilidade multirregional ao escolher sua estratégia de backup e habilitação de PITR.

Projetando para alta disponibilidade multirregional

Para obter orientação prescritiva sobre a implantação de tabelas globais, consulte Melhores práticas para o design de tabelas globais do DynamoDB.

Trabalhar com itens e atributos no DynamoDB

No Amazon DynamoDB, um item é uma coleção de atributos. Cada atributo tem um nome e um valor. Um valor de atributo pode ser uma escalar, um conjunto ou um tipo de documento. Para ter mais informações, consulte Amazon DynamoDB: como funciona.

O DynamoDB fornece quatro operações para a funcionalidade básica criar, ler, atualizar e excluir (CRUD). Todas essas operações são atômicas.

- `PutItem`: criar um item.
- `GetItem`: ler um item.
- `UpdateItem`: atualizar um item.
- `DeleteItem`: excluir um item.

Cada uma dessas operações exige que você especifique a chave primária do item com o qual deseja trabalhar. Por exemplo, para ler um item usando `GetItem`, você deve especificar a chave de partição e a chave de classificação (se aplicável) desse item.

Além das quatro operações CRUD básicas, o DynamoDB também fornece o seguinte:

- `BatchGetItem`: ler até 100 itens de uma ou mais tabelas.
- `BatchWriteItem`: criar ou excluir até 25 itens em uma ou mais tabelas.

Essas operações em lote combinam várias operações CRUD em uma única solicitação. Além disso, as operações em lote leem e gravam itens em paralelo para reduzir as latências de resposta.

Esta seção descreve como usar essas operações e inclui tópicos relacionados, como contadores atômicos e atualizações condicionais. Esta seção também inclui código de exemplo que usa os AWS SDKs.

Tópicos

- Tamanhos e formatos de item do DynamoDB
- Ler um item
- Gravar um item
- Retornar valores

- Operações em lote
- Contadores atômicos
- Gravações condicionais
- Usar expressões no DynamoDB
- Usar a vida útil (TTL) no DynamoDB
- Consultar tabelas no DynamoDB
- Verificar tabelas no DynamoDB
- PartiQL: uma linguagem de consultas compatível com SQL para o Amazon DynamoDB
- Trabalhar com itens: Java
- Trabalhar com itens: .NET

Tamanhos e formatos de item do DynamoDB

As tabelas do DynamoDB são sem esquema, exceto para a chave primária. Portanto, os itens em uma tabela podem ter atributos, tamanhos e tipos de dados diferentes.

O tamanho total de um item é a soma dos tamanhos de seus nomes e valores de atributos, somado a qualquer sobrecarga aplicável como descrito abaixo. Você pode usar as seguintes diretrizes para estimar os tamanhos de atributo:

- Strings são Unicode com codificação binária UTF-8. O tamanho de uma string é (número de bytes codificados por UTF-8 do nome do atributo) + (número de bytes codificados em UTF-8).
- Os números são de tamanho variável, com até 38 dígitos significativos. Zeros iniciais e finais são cortados. O tamanho de um número é aproximadamente (número de bytes codificados por UTF-8 do nome do atributo) + (1 byte por dois dígitos significativos) + (1 byte).
- Um valor binário deve ser codificado no formato base64 antes que possa ser enviado para o DynamoDB, mas o tamanho de byte bruto do valor é usado para calcular o tamanho. O tamanho de um atributo binário é (número de bytes codificados por UTF-8 do nome do atributo) + (número de bytes brutos).
- O tamanho de um atributo nulo ou de um atributo booliano é (número de bytes codificados por UTF-8 do nome do atributo) + (1 byte).
- Um atributo do tipo `List` ou `Map` requer 3 bytes de sobrecarga, independentemente de seu conteúdo. O tamanho de uma `List` ou um `Map` é (número de bytes codificados por UTF-8 do

nome do atributo) + soma (tamanho dos elementos aninhados) + (3 bytes). O tamanho de uma `List` ou um `Map` vazio é (número de bytes codificados por UTF-8 do nome do atributo) + (3 bytes).

- Cada elemento da `List` ou do `Map` também requer 1 byte de sobrecarga.

> **Note**
>
> Recomendamos que você escolha nomes mais curtos para os atributos. Isso ajuda a reduzir a quantidade de armazenamento necessária, mas também pode reduzir a quantidade de RCU/WCUs usada.

Para fins de faturamento de armazenamento, cada item inclui uma sobrecarga de armazenamento por item que depende dos recursos ativados.

- Todos os itens no DynamoDB requerem 100 bytes de sobrecarga de armazenamento para indexação.
- Alguns recursos do DynamoDB (tabelas globais, transações, captura de dados de alteração para o Kinesis Data Streams com o DynamoDB) exigem sobrecarga de armazenamento adicional para contabilizar atributos criados pelo sistema resultantes da ativação desses recursos. Por exemplo, tabelas globais exigem 48 bytes adicionais de sobrecarga de armazenamento.

Ler um item

Para ler um item de uma tabela do DynamoDB use a operação `GetItem`. Você deve fornecer o nome da tabela, juntamente com a chave primária do item desejado.

Example

O exemplo da AWS CLI a seguir mostra como ler um item da tabela `ProductCatalog`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> Com `GetItem`, você deve especificar a chave primária inteira, não apenas parte dela. Por exemplo, se uma tabela tiver uma chave primária composta (chave de partição e chave de

classificação), você deverá fornecer um valor para a chave de partição e um valor para a chave de classificação.

A solicitação `GetItem` executa uma leitura final consistente, por padrão. Você pode usar o parâmetro `ConsistentRead` para solicitar uma leitura altamente consistente. (Isso consome unidades de capacidade de leitura adicionais, mas retorna a versão mais recente do item.)

`GetItem` retornará todos os atributos do item. Você pode usar uma expressão de projeção para retornar apenas alguns dos atributos. Para ter mais informações, consulte Usar expressões de projeção no DynamoDB.

Para retornar o número de unidades de capacidade de leitura consumidas por `GetItem`, defina o parâmetro `ReturnConsumedCapacity` como `TOTAL`.

Example

O exemplo da AWS Command Line Interface (AWS CLI) a seguir mostra alguns dos parâmetros opcionais de `GetItem`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

Gravar um item

Para criar, atualizar ou excluir um item em uma tabela do DynamoDB use uma das seguintes operações:

- `PutItem`
- `UpdateItem`
- `DeleteItem`

Para cada uma dessas operações, você deve especificar a chave primária inteira, não apenas parte dela. Por exemplo, se uma tabela tiver uma chave primária composta (chave de partição e chave

de classificação), você deve fornecer um valor para a chave de partição e um valor para a chave de classificação.

Para retornar o número de unidades de capacidade de gravação consumidas por qualquer uma dessas operações, defina o parâmetro `ReturnConsumedCapacity` de uma das seguintes formas:

- `TOTAL`: retorna o número total de unidades de capacidade de gravação consumidas.
- `INDEXES`: retorna o número total de unidades de capacidade de gravação consumidas, com subtotais para a tabela e para todos os índices secundários que foram afetados pela operação.
- `NONE`: nenhum detalhe da capacidade de gravação é retornado. (Esse é o padrão.)

PutItem

`PutItem` cria um novo item. Se um item com a mesma chave já existir na tabela, ele será substituído pelo novo item.

Example

Gravar um novo item na tabela `Thread`. A chave primária de `Thread` consiste em `ForumName` (chave de partição) e `Subject` (chave de classificação).

```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

Os argumentos de `--item` são armazenados no arquivo `item.json`.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"},
    "Message": {"S": "First post in this thread"},
    "LastPostedBy": {"S": "fred@example.com"},
    "LastPostDateTime": {"S": "201603190422"}
}
```

UpdateItem

Se um item com a chave especificada não existir, `UpdateItem` criará um item. Caso contrário, ele modificará os atributos de um item existente.

Você usa uma expressão de atualização para especificar os atributos que deseja modificar e seus novos valores. Para ter mais informações, consulte Usar expressões de atualização no DynamoDB.

Na expressão de atualização, use valores de atributos de expressão como espaços reservados para os valores reais. Para ter mais informações, consulte Usar valores de atributos de expressão no DynamoDB.

Example

Modifique vários atributos no item `Thread`. O parâmetro opcional `ReturnValues` mostra o item como ele aparece após a atualização. Para ter mais informações, consulte Retornar valores.

```
aws dynamodb update-item \
    --table-name Thread \
    --key file://key.json \
    --update-expression "SET Answered = :zero, Replies = :zero, LastPostedBy
 = :lastpostedby" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

Os argumentos de `--key` são armazenados no arquivo `key.json`.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `expression-attribute-values.json`.

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

DeleteItem

`DeleteItem` exclui o item com a chave especificada.

Example

O exemplo da AWS CLI a seguir mostra como excluir o item `Thread`.

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

Retornar valores

Em alguns casos, talvez você queira que o DynamoDB retorne determinados valores de atributos como eles apareciam antes ou depois de modificados. As operações `PutItem`, `UpdateItem` e `DeleteItem` têm um parâmetro `ReturnValues` que você pode usar para retornar os valores de atributo antes ou depois que eles sejam modificados.

O valor padrão de `ReturnValues` é `NONE`, o que significa que o DynamoDB não retornará nenhuma informação sobre os atributos que foram modificados.

Veja a seguir as outras configurações válidas de `ReturnValues`, organizadas pela operação da API do DynamoDB.

PutItem

- `ReturnValues`: `ALL_OLD`
  - Se você substituir um item existente, `ALL_OLD` retornará o item inteiro, conforme ele aparecia antes da substituição.
  - Se você gravar um item que não existe, `ALL_OLD` não terá efeito.

UpdateItem

O uso mais comum de `UpdateItem` é para atualizar um item existente. No entanto, `UpdateItem` realmente executa um upsert, o que significa que ele criará o item automaticamente se ele ainda não existir.

- `ReturnValues`: `ALL_OLD`
  - Se você atualizar um item existente, `ALL_OLD` retornará o item inteiro, conforme ele aparecia antes da atualização.
  - Se você atualizar um item que não existe (upsert), `ALL_OLD` não terá efeito.
- `ReturnValues`: `ALL_NEW`
  - Se você atualizar um item existente, `ALL_NEW` retornará o item inteiro, conforme ele aparecia depois da atualização.

  - Se você atualizar um item que não existe (upsert), `ALL_NEW` retornará o item inteiro.
- `ReturnValues`: `UPDATED_OLD`
  - Se você atualizar um item existente, `UPDATED_OLD` retornará apenas os atributos atualizados, como eles apareciam antes da atualização.
  - Se você atualizar um item que não existe (upsert), `UPDATED_OLD` não terá efeito.
- `ReturnValues`: `UPDATED_NEW`
  - Se você atualizar um item existente, `UPDATED_NEW` retornará apenas os atributos afetados, como eles apareciam depois da atualização.
  - Se você atualizar um item que não existe (upsert), `UPDATED_NEW` retornará apenas os atributos atualizados, conforme eles aparecem após a atualização.

DeleteItem

- `ReturnValues`: `ALL_OLD`
  - Se você excluir um item existente, `ALL_OLD` retornará o item inteiro, como ele aparecia antes de você excluí-lo.
  - Se você excluir um item que não existe, `ALL_OLD` não retornará nenhum dado.

Operações em lote

Para aplicações que precisam ler ou gravar vários itens, o DynamoDB fornece as operações `BatchGetItem` e `BatchWriteItem`. Usar essas operações pode reduzir o número de round trips da rede da sua aplicação para o DynamoDB. Além disso, o DynamoDB executa as operações de leitura ou gravação individuais em paralelo. Suas aplicações se beneficiam desse paralelismo sem a necessidade de gerenciar a simultaneidade ou threading.

As operações em lote são essencialmente wrappers em torno de várias solicitações de leitura ou de gravação. Por exemplo, se uma solicitação `BatchGetItem` contiver cinco itens, o DynamoDB executará cinco operações `GetItem` em seu nome. Da mesma forma, se uma solicitação `BatchWriteItem` contiver duas solicitações Put e quatro solicitações Delete, o DynamoDB realizará duas solicitações `PutItem` e quatro `DeleteItem`.

Em geral, não há falha em uma operação em lote, a menos que haja falha em todas as solicitações no lote. Por exemplo, suponha que você execute uma operação `BatchGetItem`, mas que haja falha em uma das solicitações `GetItem` individuais no lote. Nesse caso, `BatchGetItem` retorna as

chaves e os dados da solicitação `GetItem` com falha. As outras solicitações `GetItem` no lote não são afetadas.

BatchGetItem

Uma única operação `BatchGetItem` pode conter até 100 solicitações `GetItem` individuais e recuperar até 16 MB de dados. Além disso, uma operação `BatchGetItem` pode recuperar itens de várias tabelas.

Example

Recuperar dois itens da tabela `Thread` usando uma expressão de projeção para retornar apenas alguns dos atributos.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

Os argumentos de `--request-items` são armazenados no arquivo `request-items.json`.

```
{
    "Thread": {
        "Keys": [
            {
                "ForumName":{"S": "Amazon DynamoDB"},
                "Subject":{"S": "DynamoDB Thread 1"}
            },
            {
                "ForumName":{"S": "Amazon S3"},
                "Subject":{"S": "S3 Thread 1"}
            }
        ],
        "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"
    }
}
```

BatchWriteItem

A operação `BatchWriteItem` pode conter até 25 solicitações `PutItem` e `DeleteItem` individuais e gravar até 16 MB de dados. (O tamanho máximo de um item individual é 400 KB.) Além disso, uma operação `BatchWriteItem` pode inserir ou excluir itens em várias tabelas.

> **Note**
>
> `BatchWriteItem` não é compatível com solicitações `UpdateItem`.

Example

Gravar dois itens na tabela `ProductCatalog`.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

Os argumentos de `--request-items` são armazenados no arquivo `request-items.json`.

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
            }
        }
    ]
}
```

Contadores atômicos

Você pode usar a operação `UpdateItem` para implementar um contador atômico, um atributo numérico que é incrementado incondicionalmente sem interferir em outras solicitações de gravação.

(Todas as solicitações de gravação são aplicadas na ordem em que foram recebidas.) Com um contador atômico, as atualizações não são imutáveis. Em outras palavras, o valor numérico é incrementado ou reduzido cada vez que você chama `UpdateItem`. Se o valor de incremento usado para atualizar o contador atômico for positivo, isso poderá causar contagem a mais. Se o valor do incremento for negativo, isso poderá causar contagem a menos.

Você pode usar um contador atômico para rastrear o número de visitantes de um site. Neste caso, sua aplicação poderia incrementar um valor numérico, independentemente do seu valor atual. Se houve falha em uma operação `UpdateItem`, a aplicação poderá simplesmente tentar a operação novamente. Isso arriscaria atualizar o contador duas vezes, mas provavelmente você poderia tolerar uma pequena contagem a mais ou a menos de visitantes do site.

Um contador atômico não seria apropriado onde uma contagem a mais ou a menos não pudesse ser tolerada (por exemplo, em uma aplicação bancária). Nesse caso, é mais seguro usar uma atualização condicional em vez de um contador atômico.

Para obter mais informações, consulte Incrementar e reduzir atributos numéricos.

Example

O exemplo da AWS CLI a seguir incrementa o `Price` de um produto em 5. Neste exemplo, sabia-se que o item existia antes da atualização do contador. Como `UpdateItem` não é idempotente, o `Price` aumenta cada vez que você executa esse código.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": { "N": "601" }}' \
    --update-expression "SET Price = Price + :incr" \
    --expression-attribute-values '{":incr":{"N":"5"}}' \
    --return-values UPDATED_NEW
```

Gravações condicionais

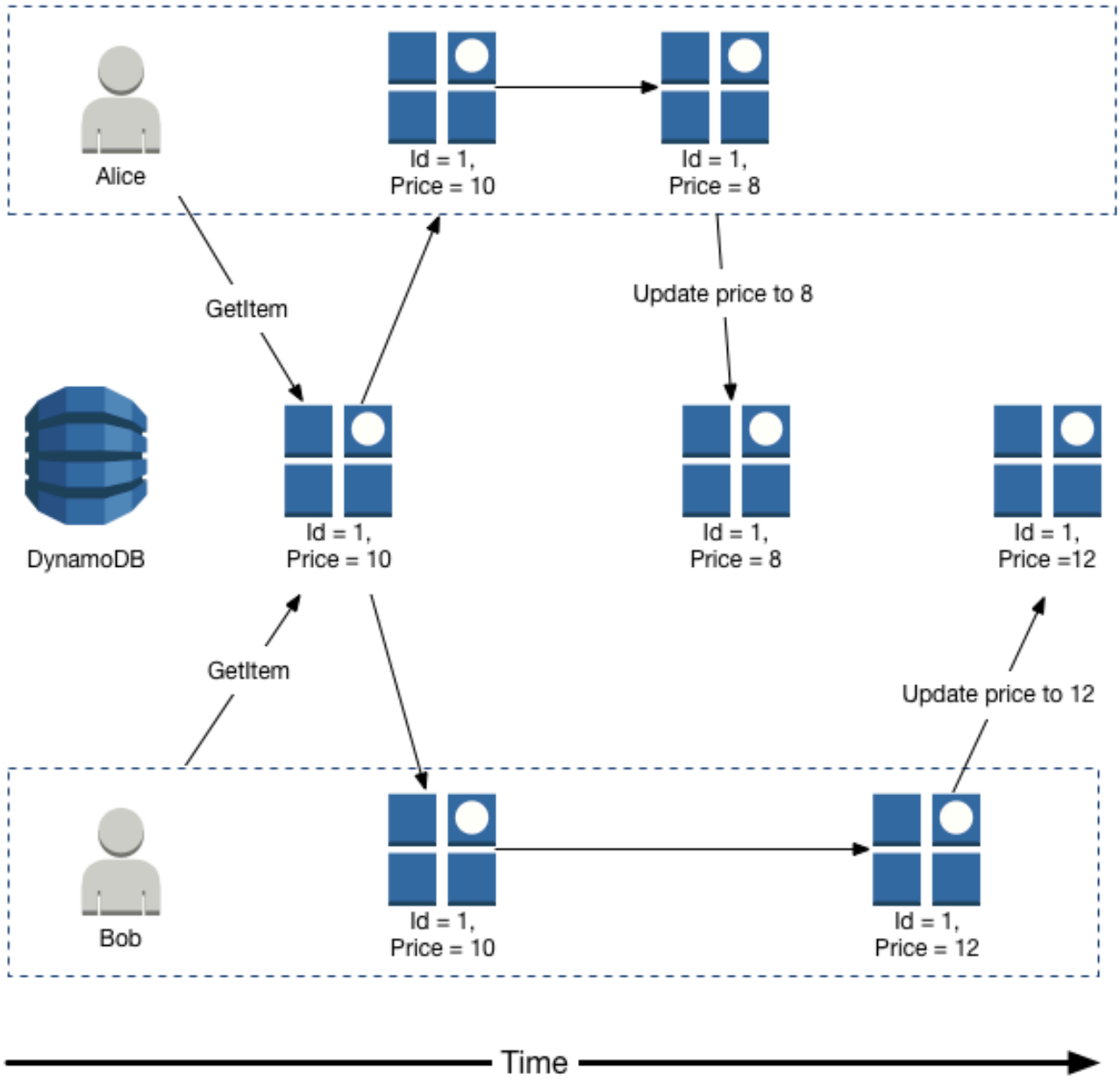
Por padrão, as operações de gravação do DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`) são incondicionais: cada operação substitui um item existente que possui a chave primária especificada.

Opcionalmente, o DynamoDB é compatível com gravações condicionais dessas operações. Uma gravação condicional terá êxito somente se os atributos de item atenderem a uma ou mais condições esperadas. Caso contrário, um erro será retornado.

As gravações condicionais comparam suas condições com a versão atualizada mais recentemente do item. Observe que, se o item não existisse anteriormente ou se a operação bem-sucedida mais recente referente a esse item tivesse sido uma exclusão, a gravação condicional não encontraria nenhum item anterior.

Gravações condicionais são úteis em muitas situações. Por exemplo, talvez você queira que uma operação `PutItem` tenha êxito somente se já não houver um item com a mesma chave primária. Ou você poderia impedir que uma operação `UpdateItem` modificasse um item, se um de seus atributos tivesse um determinado valor.

As gravações condicionais são úteis nos casos em que vários usuários tentam modificar o mesmo item. Considere o diagrama a seguir, no qual dois usuários (Alice e Bob) estão trabalhando com o mesmo item de uma tabela do DynamoDB.



Suponha que Alice usa a AWS CLI para atualizar o atributo `Price` para 8.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `expression-attribute-values.json`:

```
{
    ":newval":{"N":"8"}
}
```

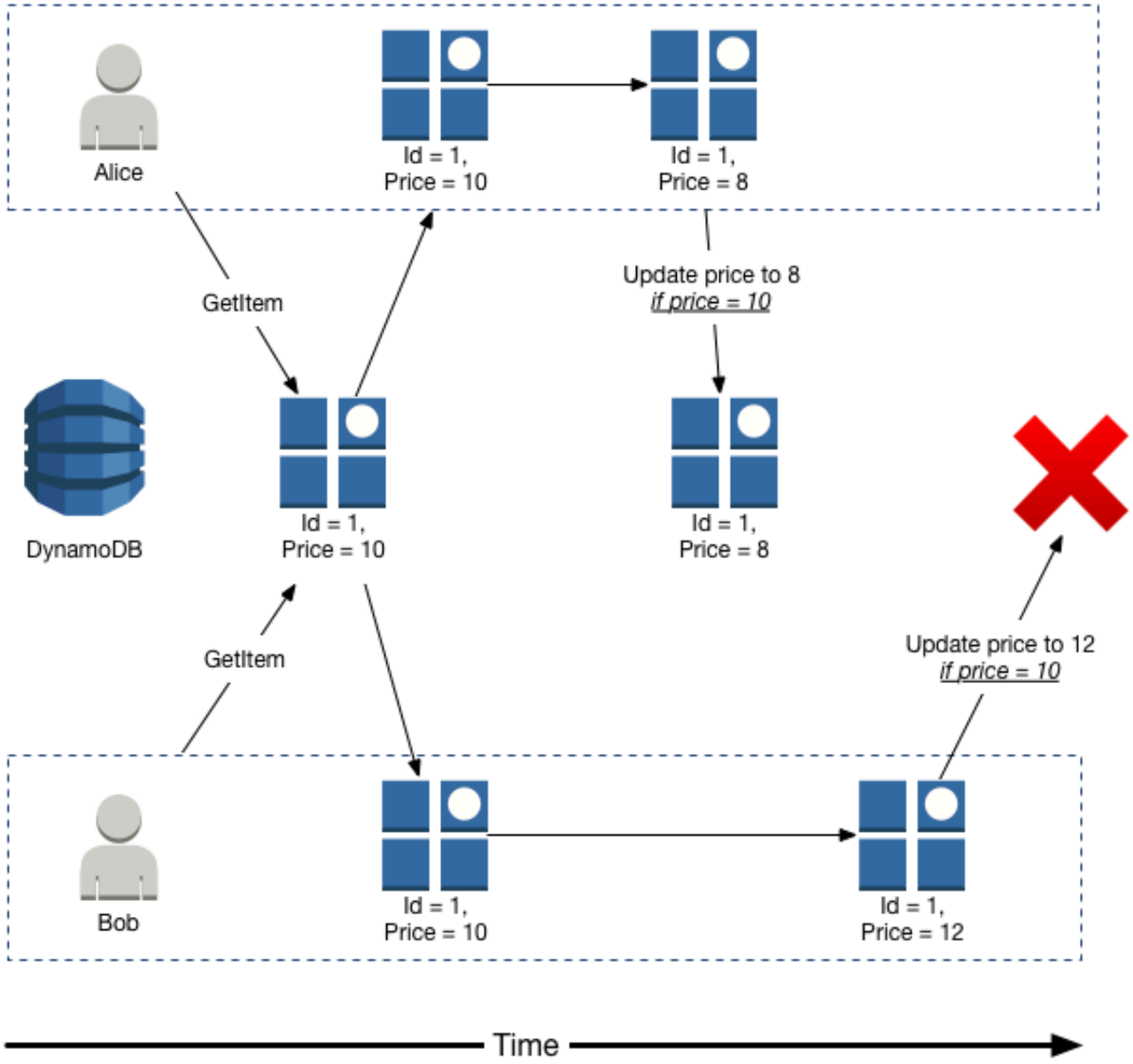
Agora, suponha que Bob emita uma solicitação `UpdateItem` semelhante mais tarde, mas altere o `Price` para 12. Para Bob, o parâmetro `--expression-attribute-values` tem a seguinte aparência.

```
{
    ":newval":{"N":"12"}
}
```

A solicitação de Bob é bem-sucedida, mas a atualização anterior de Alice é perdida.

Para solicitar uma condicional `PutItem`, `DeleteItem` ou `UpdateItem`, especifique uma expressão de condição. Uma expressão de condição é uma string que contém nomes de atributos, operadores condicionais e funções internas. A expressão inteira deve ser avaliada como verdadeira. Caso contrário, haverá falha na operação.

Agora, considere o seguinte diagrama que mostra como gravações condicionais impediriam que a atualização de Alice fosse substituída.



Alice primeiro tenta atualizar `Price` para 8, mas somente se o `Price` atual for 10.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
    --condition-expression "Price = :currval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `expression-attribute-values.json`.

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

A atualização de Alice é bem-sucedida porque a condição é avaliada como verdadeira.

Em seguida, Bob tenta atualizar o `Price` para 12, mas somente se o `Price` atual for 10. Para Bob, o parâmetro `--expression-attribute-values` tem a seguinte aparência.

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Como Alice mudou o `Price` para 8 anteriormente, a expressão de condição é avaliada como falsa, e há falha na atualização de Bob.

Para ter mais informações, consulte Exemplo de expressão de condição do DynamoDB na CLI.

Idempotência de gravação condicional

As gravações condicionais podem ser imutáveis se a verificação condicional estiver no mesmo atributo que está sendo atualizado. Isso significa que o DynamoDB realiza uma determinada solicitação de gravação somente se certos valores de atributo no item corresponderem aos valores estimados no momento da solicitação.

Por exemplo, suponha que você emita uma solicitação de `UpdateItem` para aumentar o `Price` de um item em 3, mas somente se o `Price` atual for 20. Depois de enviar a solicitação, mas antes de obter os resultados de volta, ocorre um erro de rede e você não sabe se a solicitação teve êxito. Como essa gravação condicional é imutável, você pode tentar novamente a mesma solicitação de `UpdateItem`, e o DynamoDB atualizará o item somente se `Price` for igual a 20 no momento.

Unidades de capacidade consumidas por gravações condicionais

Se uma `ConditionExpression` for avaliada como false durante uma gravação condicional, o DynamoDB ainda consumirá capacidade de gravação da tabela. A quantidade consumida depende do tamanho do item existente (ou no mínimo 1). Por exemplo, se um item existente tiver 300 kb e o item que você está tentando criar ou atualizar tiver 310 kb, as unidades de capacidade de gravação consumidas serão 300 se a condição falhar e 310 se a condição for bem-sucedida. Se for um item novo (nenhum item existente), as unidades de capacidade de gravação consumidas serão 1 se a condição falhar e 310 se a condição for bem-sucedida.

> **Note**
>
> As operações de gravação consomem apenas unidades de capacidade de gravação. Elas nunca consomem unidades de capacidade de leitura.

Uma gravação condicional com falha retorna uma `ConditionalCheckFailedException`. Quando isso ocorre, você não recebe nenhuma informação na resposta sobre a capacidade de gravação que foi consumida.

Para retornar o número de unidades de capacidade de gravação consumidas durante uma gravação condicional, você deve usar o parâmetro `ReturnConsumedCapacity`:

- `TOTAL`: retorna o número total de unidades de capacidade de gravação consumidas.
- `INDEXES`: retorna o número total de unidades de capacidade de gravação consumidas, com subtotais para a tabela e para todos os índices secundários que foram afetados pela operação.
- `NONE`: nenhum detalhe da capacidade de gravação é retornado. (Esse é o padrão.)

> **Note**
>
> Ao contrário de um índice secundário global, um índice secundário local compartilha a capacidade de throughput provisionada com sua tabela. A atividade de leitura e gravação em um índice secundário local consome a capacidade de throughput provisionado da tabela.

Usar expressões no DynamoDB

No Amazon DynamoDB, você pode usar expressões para especificar quais atributos devem ser lidos em um item, gravar dados quando uma condição é atendida, especificar como atualizar um item, definir consultas e filtrar os resultados de uma consulta.

Esta tabela descreve a gramática de expressão básica e os tipos de expressão disponíveis.

Tipo de expressão	Descrição
Expressão de projeção	Uma expressão de projeção identifica os atributos que você deseja recuperar de um item ao usar operações como GetItem, Query ou Scan.
Expressão de condição	Uma expressão de condição determina quais itens devem ser modificados ao usar as operações PutItem, UpdateItem e DeleteItem.
Expressão de atualização	Uma expressão de atualização especifica como UpdateItem modificará os atributos de um item. Por exemplo, definindo um valor escalar ou removendo elementos de uma lista ou de um mapa.
Expressão de condição principal	Uma expressão de condição principal determina quais itens uma consulta lerá em uma tabela ou índice.
Expressão de filtro	Uma expressão de filtro determina quais itens dos resultados de Query devem ser retornado s para você. Todos os outros resultados serão descartados.

Para saber mais sobre sintaxe de expressão e mais detalhes sobre cada tipo de expressão, consulte as seções a seguir.

Tópicos

- Referir-se a atributos de item ao usar expressões no DynamoDB
- Nomes (aliases) de atributo de expressão no DynamoDB
- Usar valores de atributos de expressão no DynamoDB
- Usar expressões de projeção no DynamoDB
- Usar expressões de atualização no DynamoDB
- Expressões de condição e filtro, operadores e funções no DynamoDB
- Exemplo de expressão de condição do DynamoDB na CLI

> **Note**
>
> Para fins de compatibilidade com versões anteriores, o DynamoDB também aceita parâmetros condicionais que não usam expressões. Para ter mais informações, consulte Parâmetros condicionais legados do DynamoDB.
> Novas aplicações devem usar expressões em vez de parâmetros herdados.

Referir-se a atributos de item ao usar expressões no DynamoDB

Esta seção descreve como consultar atributos de item em uma expressão no Amazon DynamoDB. Você pode trabalhar com qualquer atributo, mesmo se ele estiver profundamente aninhado dentro de várias listas e mapas.

Tópicos

- Atributos de nível superior
- Atributos aninhados
- Caminhos de documentos

Um item de exemplo: ProductCatalog

Os exemplos desta página usam o item de amostra a seguir na tabela `ProductCatalog`. (Esta tabela é descrita na seção Exemplos de tabelas e dados para uso no DynamoDB.)

```
{
    "Id": 123,
```

```
    "Title": "Bicycle 123",
    "Description": "123 description",
    "BicycleType": "Hybrid",
    "Brand": "Brand-Company C",
    "Price": 500,
    "Color": ["Red", "Black"],
    "ProductCategory": "Bicycle",
    "InStock": true,
    "QuantityOnHand": null,
    "RelatedItems": [
        341,
        472,
        649
    ],
    "Pictures": {
        "FrontView": "http://example.com/products/123_front.jpg",
        "RearView": "http://example.com/products/123_rear.jpg",
        "SideView": "http://example.com/products/123_left_side.jpg"
    },
    "ProductReviews": {
     "FiveStar": [
       "Excellent! Can't recommend it highly enough! Buy it!",
       "Do yourself a favor and buy this."
     ],
     "OneStar": [
       "Terrible product! Do not buy this."
     ]
    },
    "Comment": "This product sells out quickly during the summer",
    "Safety.Warning": "Always wear a helmet"
}
```

Observe o seguinte:

- O valor da chave de partição (`Id`) é `123`. Não há chave de classificação.
- A maioria dos atributos têm tipos de dados escalares, como `String`, `Number`, `Boolean` e `Null`.
- Um atributo (`Color`) é um `String Set`.
- Os atributos a seguir são tipos de dados de documento:
  - Uma lista de `RelatedItems`. Cada elemento é um `Id` de um produto relacionado.
  - Um mapa de `Pictures`. Cada elemento é uma breve descrição de uma imagem, juntamente com um URL para o arquivo de imagem correspondente.

- Um mapa de `ProductReviews`. Cada elemento representa uma classificação e uma lista de avaliações correspondentes a essa classificação. Inicialmente, esse mapa é preenchido com avaliações de cinco estrelas e de uma estrela.

Atributos de nível superior

Um atributo será considerado como de nível superior se ele não estiver incorporado a outro atributo. Para o item do `ProductCatalog`, os atributos de nível superior são os seguintes:

- `Id`
- `Title`
- `Description`
- `BicycleType`
- `Brand`
- `Price`
- `Color`
- `ProductCategory`
- `InStock`
- `QuantityOnHand`
- `RelatedItems`
- `Pictures`
- `ProductReviews`
- `Comment`
- `Safety.Warning`

Todos esses atributos de nível superior são escalares, exceto `Color` (lista), `RelatedItems` (lista), `Pictures` (mapa) e `ProductReviews` (mapa).

Atributos aninhados

Um atributo é considerado aninhado se ele estiver incorporado a outro atributo. Para acessar um atributo aninhado, utiliza-se operadores de cancelamento de referência:

- `[n]`: para elementos de lista

- `.` (ponto): para elementos de mapa

Acesso a elementos de lista

O operador de desreferência de um elemento de lista é [N], onde n é o número do elemento. Os elementos de lista são baseados em zero, portanto, [0] representa o primeiro elemento na lista, [1] representa o segundo, e assim por diante. Veja alguns exemplos:

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

O próprio elemento `ThisList[5]` é uma lista aninhada. Portanto, `ThisList[5][11]` refere-se ao décimo segundo elemento na lista.

O número dentro de colchetes deve ser um inteiro não negativo. Portanto, as seguintes expressões são inválidas:

- `MyList[-1]`
- `MyList[0.4]`

Acessar elementos de mapas

O operador de cancelamento de referência de um elemento de mapa é . (um ponto). Use um ponto como um separador entre os elementos em um mapa:

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

Caminhos de documentos

Em uma expressão, você usa um caminho de documento para informar ao DynamoDB onde encontrar um atributo. Para um atributo de nível superior, o caminho do documento é simplesmente o nome do atributo. Para um atributo aninhado, você pode construir o caminho do documento usando operadores de cancelamento de referência.

Veja a seguir alguns exemplos de caminhos de documentos. (Consulte o item mostrado em Referir-se a atributos de item ao usar expressões no DynamoDB.)

- Um atributo escalar de nível superior.

  `Description`

- Um atributo de lista de nível superior. (Isso retorna a lista inteira, não apenas alguns dos elementos.)

  `RelatedItems`

- O terceiro elemento na lista `RelatedItems`. (Lembre-se de que os elementos de lista são baseadas em zero.)

  `RelatedItems[2]`

- A imagem da visão frontal do produto.

  `Pictures.FrontView`

- Todas as críticas de cinco estrelas.

  `ProductReviews.FiveStar`

- A primeira das críticas de cinco estrelas.

  `ProductReviews.FiveStar[0]`

> **Note**
>
> A profundidade máxima de um caminho de documento é 32. Portanto, o número de operadores de cancelamento de referência em um caminho não pode exceder esse limite.

É possível usar qualquer nome de atributo em um caminho de documento, desde ele que cumpra estes requisitos:

- O primeiro caractere deve ser `a-z` ou `A-Z` e ou `0-9`
- O segundo caractere (se presente) deve ser `a-z`, `A-Z`

> **Note**
>
> Se um nome de atributo não cumprir essas exigências, você deverá definir um nome de atributo de expressão como um espaço reservado.

Para ter mais informações, consulte Nomes (aliases) de atributo de expressão no DynamoDB.

Nomes (aliases) de atributo de expressão no DynamoDB

Um nome de atributo de expressão é um alias (ou espaço reservado) usado em uma expressão do Amazon DynamoDB como alternativa ao nome de atributo real. Um nome de atributo de expressão deve começar com um sinal de cerquilha (#) seguido de um ou mais caracteres alfanuméricos. O sublinhado (_) também é permitido.

Esta seção descreve várias situações em que você precisa usar nomes de atributos de expressão.

> **Note**
>
> Os exemplos desta seção usam a AWS Command Line Interface (AWS CLI).

Tópicos

- Palavras reservadas
- Nomes de atributos com caracteres especiais
- Atributos aninhados
- Consultar nomes de atributo de forma repetida

Palavras reservadas

Algumas vezes, pode ser necessário escrever uma expressão que contém um nome de atributo que está em conflito com uma palavra reservada do DynamoDB. (Para obter uma lista completa de palavras reservadas, consulte Palavras reservadas no DynamoDB.)

Por exemplo, haveria falha no seguinte exemplo da AWS CLI porque `COMMENT` é uma palavra reservada.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
```

```
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

Para resolver esse problema, substitua `Comment` por um nome de atributo de expressão, como `#c`. A # (cerquilha) é obrigatória e indica que esse é um espaço reservado para um nome de atributo. O exemplo da AWS CLI agora deve ter a seguinte aparência.

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
     --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> Se o nome de um atributo começar com um número, contiver um espaço ou incluir uma palavra reservada, você deverá usar um nome de atributo de expressão para substituir esse nome de atributo na expressão.

Nomes de atributos com caracteres especiais

Em uma expressão, um ponto (“.”) é interpretado como um caractere de separação em um caminho de documento. No entanto, o DynamoDB também permite que você use um ponto e outros caracteres especiais, como um hífen (“-”) como parte do nome de um atributo. Isso pode ser ambíguo em alguns casos. Para ilustrar, vamos supor que você queira recuperar o atributo `Safety.Warning` de um item do `ProductCatalog` (consulte Referir-se a atributos de item ao usar expressões no DynamoDB).

Suponha que você queira acessar `Safety.Warning` usando uma expressão de projeção.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

O DynamoDB retorna um resultado vazio, em vez da string esperada (“`Always wear a helmet`”). Isso ocorre porque o DynamoDB interpreta um ponto em uma expressão como um separador de caminho de documento. Neste caso, defina um nome de atributo de expressão (por exemplo, `#sw`) como um substituto para `Safety.Warning`. Você poderia usar a seguinte expressão de projeção.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

Em seguida, o DynamoDB deverá retornar o resultado correto.

> **Note**
>
> Se o nome de um atributo contiver um ponto (“.”) ou um hífen (“-”), você deverá usar um nome de atributo de expressão para substituir o nome desse atributo na expressão.

Atributos aninhados

Suponha que você queira acessar o atributo `ProductReviews.OneStar` aninhado. No nome de atributo de expressão, o DynamoDB trata o ponto (“.”) como um caractere no nome de um atributo. Para consultar o atributo aninhado, defina um nome de atributo de expressão para cada elemento no caminho do documento:

- `#pr – ProductReviews`
- `#1star – OneStar`

Você poderia usar `#pr.#1star` para a expressão de projeção.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

Em seguida, o DynamoDB deverá retornar o resultado correto.

Consultar nomes de atributo de forma repetida

Os nomes de atributo de expressão são úteis quando você precisa consultar o mesmo nome de atributo repetidamente. Por exemplo, considere a seguinte expressão para recuperar algumas das revisões de um item do `ProductCatalog`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

Para tornar isso mais conciso, você pode substituir `ProductReviews` por um nome de atributo de expressão, como `#pr`. A expressão revisada agora teria a seguinte aparência.

- `#pr.FiveStar, #pr.ThreeStar, #pr.OneStar`

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \
    --expression-attribute-names '{"#pr":"ProductReviews"}'
```

Caso defina um nome de atributo de expressão, você deverá usá-lo de forma consistente na expressão inteira. Além disso, não é possível omitir o símbolo #.

Usar valores de atributos de expressão no DynamoDB

Os valores de atributo de expressão no Amazon DynamoDB atuam como variáveis. Eles substituem os valores reais que você deseja comparar; isto é, valores que talvez não conheça antes do tempo de execução. Um valor de atributo de expressão deve começar com um sinal de dois pontos (`:`) seguido por um ou mais caracteres alfanuméricos.

Por exemplo, suponha que você quisesse retornar todos os itens de `ProductCatalog` que estão disponíveis em `Black` e custam `500` ou menos. Você poderia usar uma operação `Scan` com uma expressão de filtro, como neste exemplo da AWS Command Line Interface (AWS CLI).

```
aws dynamodb scan \
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

Caso defina um valor de atributo de expressão, você deverá usá-lo de forma consistente na expressão inteira. Além disso, não é possível omitir o símbolo `:`.

Os valores de atributo de expressão são usados com expressões de condições de chaves, expressões de condições, expressões de atualização e expressões de filtro.

Usar expressões de projeção no DynamoDB

Para ler dados de uma tabela, você pode usar operações como `GetItem`, `Query`you `Scan`. O Amazon DynamoDB retornará todos os atributos de item por padrão. Para obter somente alguns, em vez de todos os atributos, use uma expressão de projeção.

Uma expressão de projeção é uma string que identifica os atributos que você deseja. Para recuperar um único atributo, especifique o seu nome. Para vários atributos, os nomes devem ser separados por vírgulas.

Veja a seguir alguns exemplos de expressões de projeção, com base no item do `ProductCatalog` em Referir-se a atributos de item ao usar expressões no DynamoDB:

- Um único atributo de nível superior.

  ```
  Title
  ```

- Três atributos de nível superior. O DynamoDB recupera o conjunto `Color` inteiro.

  ```
  Title, Price, Color
  ```

- Quatro atributos de nível superior. O DynamoDB retorna todo o conteúdo de `RelatedItems` e `ProductReviews`.

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

> **Note**
>
> A expressão de projeção não afeta o consumo de throughput provisionado. O DynamoDB determina as unidades de capacidade consumidas com base no tamanho do item, e não na quantidade de dados que são retornados para uma aplicação.

Palavras reservadas e caracteres especiais

O DynamoDB inclui palavras reservadas e caracteres especiais. O DynamoDB permite usar essas palavras reservadas e caracteres especiais nos nomes, mas é recomendável evitar, pois será necessário usar aliases sempre que esses nomes forem utilizados em uma expressão. Para obter uma lista completa, consulte Palavras reservadas no DynamoDB.

Você precisará usar nomes de atributo de expressão no lugar do nome real se:

- O nome do atributo estiver na lista de palavras reservadas do DynamoDB.
- O nome do atributo não atender ao requisito de que o primeiro caractere deve ser `a-z` ou `A-Z` e que o segundo caractere (se houver) deve ser `a-Z`, `A-Z` ou `0-9`.
- O nome do atributo contiver # (cerquilha) ou : (dois-pontos).

O exemplo de AWS CLI a seguir mostra como usar uma expressão de projeção com uma operação `GetItem`. Essa expressão de projeção recupera um atributo escalar de nível superior (`Description`), o primeiro elemento em uma lista (`RelatedItems[0]`) e uma lista aninhada em um mapa (`ProductReviews.FiveStar`).

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

O JSON a seguir seria retornado para este exemplo.

```
{
    "Item": {
        "Description": {
            "S": "123 description"
        },
```

```
        "ProductReviews": {
            "M": {
                "FiveStar": {
                    "L": [
                        {
                            "S": "Excellent! Can't recommend it highly enough! Buy it!"
                        },
                        {
                            "S": "Do yourself a favor and buy this."
                        }
                    ]
                }
            }
        },
        "RelatedItems": {
            "L": [
                {
                    "N": "341"
                }
            ]
        }
    }
}
```

Usar expressões de atualização no DynamoDB

A ação `UpdateItem` atualiza um item existente ou adiciona um item novo à tabela, caso ele ainda não exista. Você deve fornecer a chave do item que deseja atualizar. Você também deve fornecer uma expressão de atualização indicando os atributos que deseja modificar e os valores que deseja atribuir a eles.

Uma expressão de atualização especifica como `UpdateItem` modificará os atributos de um item. Por exemplo, definindo um valor escalar ou removendo elementos de uma lista ou de um mapa.

Veja a seguir um resumo da sintaxe para expressões de atualização.

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

Uma expressão de atualização consiste em uma ou mais cláusulas. Cada cláusula começa com uma palavra-chave `SET`, `REMOVE`, `ADD` ou `DELETE`. Você pode incluir qualquer uma dessas cláusulas em uma expressão de atualização, em qualquer ordem. No entanto, cada palavra-chave de ação pode aparecer apenas uma vez.

Dentro de cada cláusula há uma ou mais ações, separadas por vírgulas. Cada ação representa uma modificação de dados.

Os exemplos desta seção se baseiam no item `ProductCatalog` mostrado em Usar expressões de projeção no DynamoDB.

Os tópicos abaixo abrangem alguns casos de uso diferentes da ação `SET`.

Tópicos

- SET: modificar ou adicionar atributos de um item
- REMOVE: excluir atributos de um item
- ADD: atualizar números e conjuntos
- DELETE: remover elementos de um conjunto
- Usar várias expressões de atualização

SET: modificar ou adicionar atributos de um item

Use a ação `SET` em uma expressão de atualização para adicionar um ou mais atributos a um item. Se qualquer um desses atributos já existir, eles serão substituídos pelos novos valores. Se você deseja evitar a substituição de um atributo existente, pode usar `SET` com a função `if_not_exists`. A função `if_not_exists` é específica à ação `SET` e só pode ser usada em uma expressão de atualização.

Quando você usa `SET` para atualizar um elemento de lista, o conteúdo desse elemento é substituído pelos novos dados especificados. Se o elemento ainda não existir, `SET` acrescentará o novo elemento ao final da lista.

Se você adicionar vários elementos em uma única operação `SET`, estes serão classificados por número de elemento.

Você também pode usar `SET` para adicionar ou subtrair de um atributo do tipo `Number`. Para executar várias ações `SET`, separe-as com vírgulas.

No seguinte resumo de sintaxe:

- O elemento *path* é o caminho do documento para o item.
- Um elemento *operando* pode ser o caminho de um documento para um item ou uma função.

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

Se o item não contiver um atributo no caminho especificado, `if_not_exists` será avaliado como `value`. Caso contrário, será avaliado como `path`.

A seguinte operação `PutItem` cria um item de exemplo ao qual os exemplos fazem referência.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Os argumentos de `--item` são armazenados no arquivo `item.json`. (Para simplificar, apenas alguns atributos de item são usados.)

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

Tópicos

- Modificar atributos

- Adicionar listas e mapas
- Adicionar elementos a uma lista
- Adicionar atributos de mapa aninhados
- Incrementar e reduzir atributos numéricos
- Acrescentar elementos a uma lista
- Impedir substituições de um atributo existente

Modificar atributos

Example

Atualize os atributos `ProductCategory` e `Price`.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET ProductCategory = :c, Price = :p" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> Na operação `UpdateItem`, `--return-values ALL_NEW` faz com que o DynamoDB retorne o item como ele aparece após a atualização.

Adicionar listas e mapas

Example

Adicione uma nova lista e um novo mapa.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems = :ri, ProductReviews = :pr" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

Adicionar elementos a uma lista

Example

Adicione um novo atributo à lista `RelatedItems`. (Lembre-se de que elementos de lista são baseados em zero e, portanto, [0] representa o primeiro elemento da lista, [1] representa o segundo, e assim por diante.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :ri" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> Quando você usa SET para atualizar um elemento de lista, o conteúdo desse elemento é substituído pelos novos dados especificados. Se o elemento ainda não existir, SET acrescentará o novo elemento ao final da lista.
> Se você adicionar vários elementos em uma única operação SET, estes serão classificados por número de elemento.

Adicionar atributos de mapa aninhados

Example

Adicione alguns atributos de mapa aninhados.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #pr.#5star[1] = :r5, #pr.#3star = :r3" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Os argumentos de `--expression-attribute-names` são armazenados no arquivo `names.json`.

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

Incrementar e reduzir atributos numéricos

Você pode adicionar ou subtrair de um atributo numérico existente. Para fazer isso, use os operadores + (mais) e - (menos).

Example

Diminua o valor do `Price` de um item.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Para aumentar o valor do `Price`, você usa o operador + na expressão de atualização.

Acrescentar elementos a uma lista

É possível adicionar elementos ao final de uma lista. Para fazer isso, use `SET` com a função `list_append`. (O nome da função diferencia maiúsculas de minúsculas.) A função `list_append` é específica à ação `SET` e só pode ser usada em uma expressão de atualização. A sintaxe é a seguinte.

- `list_append (`*`list1`*`, `*`list2`*`)`

A função utiliza duas listas como entrada e acrescenta todos os elementos de *`list2`* a *`list1`*.

Example

Em Adicionar elementos a uma lista, você cria a lista `RelatedItems` e a preenche com dois elementos: `Hammer` e `Nails`. Na sequência, você acrescenta mais dois elementos ao final de `RelatedItems`.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(#ri, :vals)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values file://values.json  \
    --return-values ALL_NEW
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
}
```

Por fim, você acrescenta mais um elemento ao início de `RelatedItems`. Para fazer isso, alterne a ordem dos elementos de `list_append`. (Lembre-se de que `list_append` usa duas listas como entrada e acrescenta a segunda lista à primeira.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(:vals, #ri)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values '{":vals": {"L": [ { "S": "Chisel" }]}}' \
    --return-values ALL_NEW
```

Agora, o atributo `RelatedItems` resultante contém cinco elementos, na seguinte ordem: `Chisel`, `Hammer`, `Nails`, `Screwdriver`, `Hacksaw`.

Impedir substituições de um atributo existente

Example

Defina o `Price` de um item, mas somente se o item ainda não tiver um atributo `Price`. (Se o atributo `Price` já existir, não acontecerá nada.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = if_not_exists(Price, :p)" \
    --expression-attribute-values '{":p": {"N": "100"}}' \
    --return-values ALL_NEW
```

REMOVE: excluir atributos de um item

Use a ação `REMOVE` em uma expressão de atualização para remover um ou mais atributos de um item no Amazon DynamoDB. Para executar várias ações `REMOVE`, separe-as com vírgulas.

O seguinte é um resumo da sintaxe de `REMOVE` em uma expressão de atualização. O único operando é o caminho do documento do atributo que você deseja remover.

```
remove-action ::=
    path
```

Example

Remova alguns atributos de um item. (Se os atributos não existirem, nada acontecerá.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

Remover elementos de uma lista

É possível usar `REMOVE` para excluir elementos individuais de uma lista.

Example

Em Acrescentar elementos a uma lista, você modifica um atributo da lista (`RelatedItems`) de forma que ele contenha cinco elementos:

- `[0]—Chisel`
- `[1]—Hammer`
- `[2]—Nails`
- `[3]—Screwdriver`
- `[4]—Hacksaw`

O exemplo da AWS Command Line Interface (AWS CLI) a seguir exclui `Hammer` e `Nails` da lista.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Depois que `Hammer` e `Nails` forem removidos, os elementos restantes serão deslocados. Agora, a lista contém o seguinte:

- `[0]—Chisel`
- `[1]—Screwdriver`
- `[2]—Hacksaw`

ADD: atualizar números e conjuntos

> **Note**
>
> Em geral, recomendamos o uso de `SET` em vez de `ADD`.

Use a ação `ADD` em uma expressão de atualização para adicionar um novo atributo e seus valores a um item.

Se o atributo já existir, o comportamento de `ADD` dependerá do tipo de dados do atributo:

- Se o atributo for um número, e o valor que você está adicionando também for um número, esse valor será matematicamente adicionado ao atributo existente. (Se o valor for um número negativo, ele será subtraído do atributo existente.)

- Se o atributo for um conjunto, e o valor que você está adicionando também for um conjunto, esse valor será acrescentado ao conjunto existente.

> **Note**
>
> A ação ADD oferece suporte apenas a tipos de dados de número e conjunto.

Para executar várias ações ADD, separe-as com vírgulas.

No seguinte resumo de sintaxe:

- O elemento *path* é o caminho do documento para um atributo. O atributo deve ser um `Number` ou um tipo de dados de conjunto.
- O elemento *value* é um número que você deseja adicionar ao atributo (para tipos de dados `Number`) ou um conjunto a ser acrescentado ao atributo (para tipos de conjunto).

```
add-action ::=
    path value
```

Os tópicos abaixo abrangem alguns casos de uso diferentes da ação ADD.

Tópicos

- Adicionar um número
- Adicionar elementos a um conjunto

Adicionar um número

Suponha que o atributo `QuantityOnHand` não exista. O exemplo da AWS CLI a seguir define `QuantityOnHand` como 5.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD QuantityOnHand :q" \
    --expression-attribute-values '{":q": {"N": "5"}}' \
    --return-values ALL_NEW
```

Agora que `QuantityOnHand` existe, você pode executar novamente o exemplo para incrementar `QuantityOnHand` em 5 de cada vez.

Adicionar elementos a um conjunto

Suponha que o atributo `Color` não exista. O exemplo da AWS CLI a seguir define `Color` como um conjunto de strings com dois elementos.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Orange", "Purple"]}}' \
    --return-values ALL_NEW
```

Agora que `Color` existe, você pode adicionar mais elementos a ele.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Yellow", "Green", "Blue"]}}' \
    --return-values ALL_NEW
```

DELETE: remover elementos de um conjunto

> **Important**
>
> A ação `DELETE` oferece suporte apenas a tipos de dados `Set`.

Use a ação `DELETE` em uma expressão de atualização para remover um ou mais elementos de um conjunto. Para executar várias ações `DELETE`, separe-as com vírgulas.

No seguinte resumo de sintaxe:

- O elemento *`path`* é o caminho do documento para um atributo. Esse atributo deve ser um tipo de dados de conjunto.
- O elemento *`subset`* é um ou mais elementos que você deseja excluir de *`path`*. Você deve especificar o elemento *`subset`* como um tipo set.

```
delete-action ::=
    path subset
```

Example

Em Adicionar elementos a um conjunto, você cria o conjunto de tipo string `Color`. Este exemplo remove alguns dos elementos desse conjunto.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "DELETE Color :p" \
    --expression-attribute-values '{":p": {"SS": ["Yellow", "Purple"]}}' \
    --return-values ALL_NEW
```

Usar várias expressões de atualização

É possível usar várias expressões de atualização em uma única declaração.

Example

Se quiser modificar o valor de um atributo e remover outro completamente, você poderá usar uma ação SET e uma REMOVE em uma única declaração. Essa operação reduziria o valor de `Price` para 15 e, ao mesmo tempo, removeria o atributo `InStock` do item.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p REMOVE InStock" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Example

Se quiser adicionar a uma lista e, ao mesmo tempo, alterar o valor de outro atributo, você poderá usar duas ações SET em uma única declaração. Essa operação adicionaria “Unhas” ao atributo da lista `RelatedItems` e também definiria o valor de `Price` como 21.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
```

```
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :newValue, Price = :newPrice" \
    --expression-attribute-values '{":newValue": {"S":"Nails"}, ":newPrice":
 {"N":"21"}}'  \
    --return-values ALL_NEW
```

Expressões de condição e filtro, operadores e funções no DynamoDB

Para manipular dados em uma tabela do DynamoDB, use as operações `PutItem`, `UpdateItem` e `DeleteItem`. Para essas operações de manipulação de dados, é possível especificar uma expressão de condição para determinar quais itens devem ser modificados. Se a expressão de condição for avaliada como verdadeira, a operação terá êxito. Caso contrário, haverá falha na operação.

Esta seção aborda as funções e palavras-chave integradas para escrever expressões de filtro e expressões de condição no Amazon DynamoDB. Para obter informações mais detalhadas sobre funções e programação com o DynamoDB, consulte Programação com o DynamoDB e os AWS SDKs e a Referência da API do DynamoDB.

Tópicos

- Sintaxe para expressões de filtro e de condição
- Fazer comparações
- Funções
- Avaliações lógicas
- Parênteses
- Precedência em condições

Sintaxe para expressões de filtro e de condição

No seguinte resumo de sintaxe, um *operando* pode ser o seguinte:

- Um nome de atributo de nível superior, como `Id`, `Title`, `Description` ou `ProductCategory`
- Um caminho de documento que faz referência a um atributo aninhado

```
condition-expression ::=
      operand comparator operand
    | operand BETWEEN operand AND operand
```

```
    | operand IN ( operand (',' operand (, ...) ))
    | function
    | condition AND condition
    | condition OR condition
    | NOT condition
    | ( condition )

comparator ::=
    =
    | <>
    | <
    | <=
    | >
    | >=

function ::=
    attribute_exists (path)
    | attribute_not_exists (path)
    | attribute_type (path, type)
    | begins_with (path, substr)
    | contains (path, operand)
    | size (path)
```

Fazer comparações

Use esses comparadores para comparar um operando com um único valor:

- *a* = *b*: verdadeiro se *a* for igual a *b*.
- *a* <> *b*: verdadeiro se *a* não for igual a *b*.
- *a* < *b*: verdadeiro se *a* for menor que *b*.
- *a* <= *b*: verdadeiro se *a* for menor que ou igual a *b*.
- *a* > *b*: verdadeiro se *a* for maior que *b*.
- *a* >= *b*: verdadeiro se *a* for maior ou igual a *b*.

Use as palavras-chave `BETWEEN` e `IN` para comparar um operando com um intervalo de valores ou com uma lista enumerada de valores:

- *a* `BETWEEN` *b* `AND` *c*: verdadeiro se *a* for maior ou igual a *b* e menor ou igual a *c*.
- *a* `IN (`*b*`, `*c*`, `*d*`)` : verdadeiro se *a* for igual a qualquer um dos valores na lista; por exemplo, *b*, *c* ou *d*. A lista pode conter até 100 valores, separados por vírgulas.

Funções

Use as funções a seguir para determinar se um atributo existe em um item ou para avaliar o valor de um atributo. Esses nomes de funções diferenciam maiúsculas de minúsculas. Para um atributo aninhado, você deve fornecer o caminho completo do documento.

Função	Descrição
`attribute_exists ( path)`	True se o item contiver o atributo especificado por `path`. Exemplo: verificar se um item na tabela `Product` tem uma imagem de vista lateral. • `attribute_exists (#Pictures.#SideView)`
`attribute_not_exists ( path)`	True se o atributo especificado por `path` não existir no item. Exemplo: verificar se um item tem um atributo `Manufacturer` . • `attribute_not_exists (Manufacturer)`
`attribute_type ( path, type)`	True se o atributo no caminho especificado for de um tipo de dados específico. O parâmetro `type` deve ser um dos seguintes: • `S` – String • `SS`: String Set • `N` – Number • `NS`: Number Set

Função	Descrição
	• `B` – Binary • `BS` Binary Set • `BOOL` – Boolean • `NULL` – Null • `L` – List • `M` – Map Você deve usar um valor de atributo de expressão para o parâmetro `type`. Exemplo: verificar se o atributo `QuantityOnHand` é do tipo Lista. Neste exemplo, `:v_sub` é um espaço reservado para a string `L`. • `attribute_type (ProductReviews.FiveStar, :v_sub)` Você deve usar um valor de atributo de expressão para o parâmetro `type`.

Função	Descrição
`begins_with ( path, substr)`	Verdadeiro se o atributo especificado por `path` começar com uma substring específica. Exemplo: verificar se os primeiros caracteres do URL da imagem de vista frontal são `http://`. • `begins_with (Pictures.FrontView, :v_sub)` O valor do atributo de expressão `:v_sub` é um espaço reservado para `http://`.

Função	Descrição
`contains (path, operand)`	Verdadeiro se o atributo especificado por `path` for um dos seguintes: • Uma `String` que contém uma substring específica. • Um `Set` que contém um elemento específico dentro dele. • Um `List` que contém um elemento específico dentro da lista. Se o atributo especificado por `path` for `String`, o `operand` deve ser `String`. Se o atributo especificado por `path` for um `Set`, o `operand` deverá ser o tipo de elemento do conjunto. O caminho e o operando devem ser distintos . Isto é, `contains (a, a)` retorna um erro. Exemplo: verificar se o atributo `Brand` contém a substring `Company`. • `contains (Brand, :v_sub)` O valor do atributo de expressão `:v_sub` é um espaço reservado para `Company`. Exemplo: verificar se o produto está disponível em vermelho. • `contains (Color, :v_sub)`

Função	Descrição
	O valor do atributo de expressão `:v_sub` é um espaço reservado para `Red`.

Função	Descrição
`size (path)`	Retorna um número que representa o tamanho de um atributo. Veja a seguir os tipos de dados válidos para uso com `size`. Se o atributo for do tipo `String`, `size` retornará o comprimento da string. Exemplo: verificar se a string `Brand` é menor ou igual a 20 caracteres. O valor do atributo de expressão `:v_sub` é um espaço reservado para `20`. • `size (Brand) <= :v_sub` Se o atributo for do tipo `Binary`, `size` retornará o número de bytes no valor do atributo. Exemplo: suponha que o item de `ProductCatalog` tenha um atributo binário chamado `VideoClip`, que contém um curto vídeo sobre o produto em uso. A seguinte expressão verifica se `VideoClip` excede 64.000 bytes. O valor do atributo de expressão `:v_sub` é um espaço reservado para `64000`. • `size(VideoClip) > :v_sub` Se o atributo for de um tipo de dados de `Set`, `size` retornará o número de elementos no conjunto.

Função	Descrição
	Exemplo: verificar se o produto está disponível em mais de uma cor. O valor do atributo de expressão `:v_sub` é um espaço reservado para `1`. • `size (Color) < :v_sub` Se o atributo for do tipo `List` ou `Map`, `size` retornará o número de elementos filho. Exemplo: verificar se o número de revisões `OneStar` excedeu um determinado limite. O valor do atributo de expressão `:v_sub` é um espaço reservado para `3`. • `size(ProductReviews.OneStar) > :v_sub`

Avaliações lógicas

Use as palavras-chave `AND`, `OR` e `NOT` para executar avaliações lógicas. Na lista a seguir, *a* e *b* representam condições a serem avaliadas.

- *a* `AND` *b*: verdadeiro se *a* e *b* forem ambos verdadeiros.
- *a* `OR` *b*: verdadeiro se *a* ou *b* ou ambos forem verdadeiros.
- `NOT` *a*: verdadeiro se *a* for falso. Falso se *a* for verdadeiro.

Veja a seguir um exemplo de código de AND em uma operação.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

Parênteses

Use parênteses para alterar a precedência de uma avaliação lógica. Por exemplo, suponha que as condições *a* e *b* sejam verdadeiras e que a condição *c* seja falsa. As expressões a seguir são avaliadas como verdadeiras:

- *a* OR *b* AND *c*

No entanto, se você colocar uma condição entre parênteses, ela será avaliada primeiro. Por exemplo, o seguinte é avaliado como falso:

- (*a* OR *b*) AND *c*

> **Note**
>
> Você pode aninhar parênteses em uma expressão. Os componentes mais internos são avaliados primeiro.

Veja a seguir um exemplo de código com parênteses em uma avaliação lógica.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

Precedência em condições

O DynamoDB avalia as condições da esquerda para a direita usando as seguintes regras de precedência:

- = <> < <= > >=
- IN
- BETWEEN
- attribute_exists attribute_not_exists begins_with contains
- Parênteses
- NOT
- AND

- `OR`

Exemplo de expressão de condição do DynamoDB na CLI

Veja a seguir alguns exemplos da AWS Command Line Interface (AWS CLI) para uso de expressões de condição. Estes exemplos se baseiam na tabela `ProductCatalog`, que foi apresentada em Referir-se a atributos de item ao usar expressões no DynamoDB. A chave de partição dessa tabela é `Id`. Não há uma chave de classificação. A seguinte operação `PutItem` cria um item `ProductCatalog` de amostra ao qual os exemplos se referem.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Os argumentos de `--item` são armazenados no arquivo `item.json`. (Para simplificar, apenas alguns atributos de item são usados.)

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

Tópicos

- Put condicional
- Exclusões condicionais
- Atualizações condicionais
- Exemplos de expressão condicional

Put condicional

A operação `PutItem` substitui um item com a mesma chave primária (se houver). Se quiser evitar isso, use uma expressão de condição. Isso permitirá que a gravação continue apenas se o item em questão ainda não tiver a mesma chave primária.

O exemplo a seguir usa `attribute_not_exists()` para verificar se a chave primária existe na tabela antes de tentar a operação de gravação.

> **Note**
>
> Se a chave primária consistir em uma chave de partição (pk) e uma chave de classificação (sk), o parâmetro verificará se `attribute_not_exists(pk)` E `attribute_not_exists(sk)` são avaliados como uma declaração inteiramente verdadeira ou falsa antes de tentar a operação de gravação.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

Se a expressão de condição for avaliada como falsa, o DynamoDB retornará uma mensagem de erro The conditional request failed (Falha na solicitação condicional).

> **Note**
>
> Para obter mais informações sobre `attribute_not_exists` e outras funções, consulte Expressões de condição e filtro, operadores e funções no DynamoDB.

Exclusões condicionais

Para realizar uma exclusão condicional, use uma operação `DeleteItem` com uma expressão de condição. A expressão de condição deve ser avaliada como verdadeira para que a operação tenha êxito; caso contrário, haverá falha na operação.

Considere o item definido acima.

Suponha que você queira excluir o item, mas somente nas seguintes condições:

- O valor de `ProductCategory` é “Sporting Goods” ou “Gardening Supplies”.
- O valor de `Price` está entre 500 e 600.

O exemplo a seguir tenta excluir o item.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
```

```
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo
 and :hi)" \
    --expression-attribute-values file://values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> **Note**
>
> Na expressão de condição, o caractere `:` (dois pontos) indica um valor de atributo de expressão: um espaço reservado para um valor real. Para ter mais informações, consulte Usar valores de atributos de expressão no DynamoDB.
> Para obter mais informações sobre `IN`, `AND` e outras palavras-chave, consulte Expressões de condição e filtro, operadores e funções no DynamoDB.

Neste exemplo, a comparação `ProductCategory` é avaliada como true, mas a comparação `Price` é avaliada como false. Isso faz com que a expressão de condição seja avaliada como falsa e haja falha na operação `DeleteItem`.

Atualizações condicionais

Para realizar uma atualização condicional, use uma operação `UpdateItem` com uma expressão de condição. A expressão de condição deve ser avaliada como verdadeira para que a operação tenha êxito. Caso contrário, haverá falha na operação.

> **Note**
>
> `UpdateItem` também oferece suporte a expressões de atualização, nas quais você especifica as modificações que deseja fazer em um item. Para ter mais informações, consulte Usar expressões de atualização no DynamoDB.

Suponha que você tenha começado com o item definido acima.

O exemplo a seguir realiza uma operação `UpdateItem`. Ele tenta reduzir o `Price` de um produto em 75, mas a expressão de condição impedirá a atualização se o `Price` atual for menor ou igual a 500.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --update-expression "SET Price = Price - :discount" \
    --condition-expression "Price > :limit" \
    --expression-attribute-values file://values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

Se o valor inicial de `Price` for 650, a operação `UpdateItem` reduzirá o `Price` para 575. Se você executar a operação `UpdateItem` novamente, o valor de `Price` será reduzido para 500. Se você executá-la uma terceira vez, a expressão de condição será avaliada como falsa, e haverá falha na atualização.

> **Note**
>
> Na expressão de condição, o caractere `:` (dois pontos) indica um valor de atributo de expressão: um espaço reservado para um valor real. Para ter mais informações, consulte Usar valores de atributos de expressão no DynamoDB.
> Para obter mais informações sobre ">" e outros operadores, consulte Expressões de condição e filtro, operadores e funções no DynamoDB.

Exemplos de expressão condicional

Para obter mais informações sobre as funções usadas nos exemplos a seguir, consulte Expressões de condição e filtro, operadores e funções no DynamoDB. Se você quiser saber mais sobre como

especificar diferentes tipos de atributo em uma expressão, consulte Referir-se a atributos de item ao usar expressões no DynamoDB.

Verificar atributos em um item

Você pode verificar a existência (ou inexistência) de qualquer atributo. Se a expressão da condição for avaliada como verdadeira, a operação terá êxito. Caso contrário, haverá falha.

O exemplo a seguir usa `attribute_not_exists` para excluir um produto apenas se ele não tiver um atributo `Price`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

O DynamoDB também fornece uma função `attribute_exists`. O exemplo a seguir excluirá um produto somente se ele tiver recebido revisões ruins.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Verificar tipo de atributo

É possível verificar o tipo de dados de um valor de atributo usando a função `attribute_type`. Se a expressão da condição for avaliada como verdadeira, a operação terá êxito. Caso contrário, haverá falha.

O exemplo a seguir usa `attribute_type` para excluir um produto somente se ele tiver um atributo `Color` do tipo Conjunto de strings.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_type(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo expression-attribute-values.json.

```
{
    ":v_sub":{"S":"SS"}
}
```

Verificar valor inicial da string

É possível verificar se um valor de atributo String começa com uma substring específica usando a função `begins_with`. Se a expressão da condição for avaliada como verdadeira, a operação terá êxito. Caso contrário, haverá falha.

O exemplo a seguir usará `begins_with` para excluir um produto somente se o elemento `FrontView` do mapa `Pictures` começar com um valor específico.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo expression-attribute-values.json.

```
{
    ":v_sub":{"S":"http://"}
}
```

Verificar um elemento em um conjunto

É possível verificar se há um elemento em um conjunto ou procurar uma substring em uma string usando a função `contains`. Se a expressão da condição for avaliada como verdadeira, a operação terá êxito. Caso contrário, haverá falha.

O exemplo a seguir usará `contains` para excluir um produto somente se o Conjunto de strings `Color` tiver um elemento com um valor específico.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "contains(Color, :v_sub)" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo expression-attribute-values.json.

```
{
    ":v_sub":{"S":"Red"}
}
```

Verificar o tamanho do valor de um atributo

É possível verificar o tamanho do valor de um atributo usando a função `size`. Se a expressão da condição for avaliada como verdadeira, a operação terá êxito. Caso contrário, haverá falha.

O exemplo a seguir usará `size` para excluir um produto somente se o tamanho do atributo binário `VideoClip` for maior que `64000` bytes.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "size(VideoClip) > :v_sub" \
    --expression-attribute-values file://expression-attribute-values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo expression-attribute-values.json.

```
{
    ":v_sub":{"N":"64000"}
}
```

Usar a vida útil (TTL) no DynamoDB

A vida útil (TTL) para DynamoDB é um método econômico para excluir itens que não são mais relevantes. A TTL permite definir um carimbo de data e hora de validade por item que indica quando um item não é mais necessário. O DynamoDB exclui automaticamente os itens expirados alguns dias após o vencimento, sem consumir o throughput de gravação.

Para usar a TTL, primeiro habilite-a em uma tabela e, depois, defina um atributo específico para armazenar o carimbo de data e hora de vencimento da TTL. O carimbo de data e hora deve ser

armazenado no formato de hora de época do Unix na granularidade de segundos. Sempre que um item é criado ou atualizado, é possível calcular o prazo de validade e salvá-lo no atributo TTL.

Itens com atributos TTL válidos e expirados podem ser excluídos pelo sistema a qualquer momento, normalmente alguns dias após a validade. Você ainda pode atualizar os itens expirados que estão pendentes de exclusão, incluindo alterar ou remover os atributos TTL. Ao atualizar um item expirado, recomendamos usar uma expressão de condição para garantir que o item não tenha sido excluído posteriormente. Use expressões de filtro para remover itens expirados dos resultados de Scan e Query.

Os itens excluídos funcionam de forma semelhante aos excluídos por meio de operações de exclusão típicas. Depois de excluídos, os itens entram no DynamoDB Streams como exclusões de serviços, em vez de exclusões de usuários, e são removidos dos índices secundários locais e globais, assim como outras operações de exclusão.

Se estiver usando Global Tables versão 2019.11.21 (atual) das tabelas globais e também usar o recurso TTL, o DynamoDB replicará as exclusões de TTL em todas as tabelas de réplica. A exclusão inicial de TTL não consome unidades de capacidade de gravação (WCU) na região onde a TTL expira. No entanto, a exclusão de TTL replicada para as tabelas de réplica consume uma unidade de capacidade de gravação replicada ao usar a capacidade provisionada ou a unidade de gravação replicada ao usar o modo de capacidade sob demanda, em cada uma das regiões de réplica, e serão aplicadas cobranças.

Para obter mais informações sobre TTL, consulte estes tópicos:

Tópicos

- Habilitar a vida útil (TTL) no DynamoDB
- Como calcular a vida útil (TTL) no DynamoDB
- Trabalhar com itens expirados e vida útil (TTL)

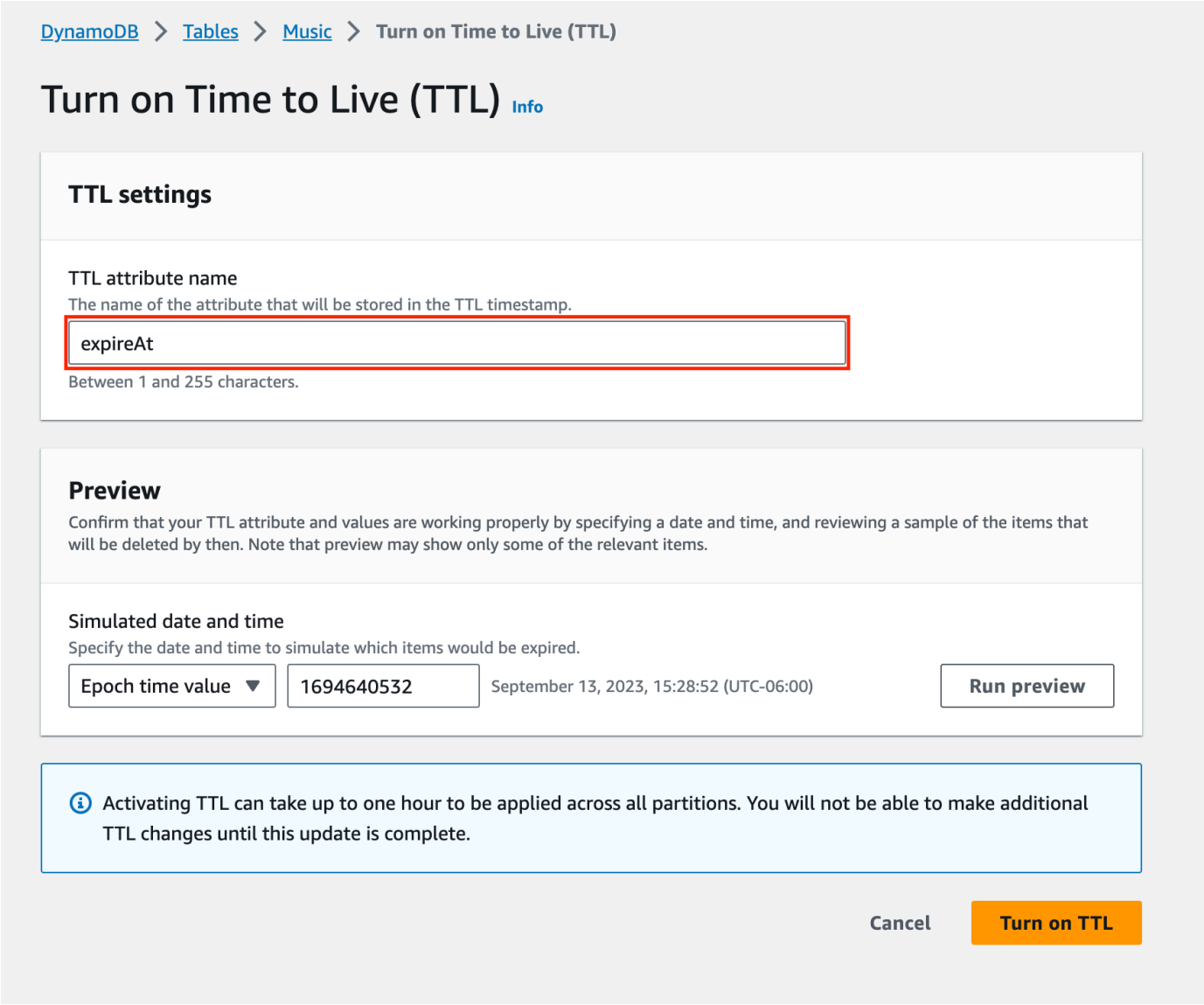
Habilitar a vida útil (TTL) no DynamoDB

> **Note**
>
> Para auxiliar na depuração e verificação da operação adequada do recurso TTL, os valores fornecidos para o item TTL são registrados em log em texto simples nos logs de diagnóstico do DynamoDB.

É possível habilitar a TTL no console do Amazon DynamoDB, na AWS Command Line Interface (AWS CLI) ou usando a Referência da API do Amazon DynamoDB com qualquer um dos supostos SDKs da AWS. Demora cerca de uma hora para que a TTL seja habilitada em todas as partições.

Habilitar a TTL do DynamoDB usando o console da AWS

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. Escolha Tables (Tabelas) e selecione a tabela que você deseja modificar.
3. Na guia Configurações adicionais, na seção Vida útil (TTL), selecione Ativar para habilitar a TTL.
4. Ao habilitar o TTL em uma tabela, o DynamoDB exige identificar um nome de atributo específico que o serviço procurará ao determinar se um item está qualificado para expiração. O nome do atributo TTL, mostrado abaixo, diferencia maiúsculas de minúsculas e deve corresponder ao atributo definido em suas operações de leitura e gravação. Uma incompatibilidade fará com que os itens expirados não sejam excluídos. Para renomear o atributo TTL, é necessário desabilitar a TTL e a reabilitar com o novo atributo daqui para frente. A TTL continuará processando as exclusões por cerca de trinta minutos depois de desabilitada. A TTL deve ser reconfigurada em tabelas restauradas.



5. (Opcional) É possível realizar um teste simulando a data e a hora da validade e combinando alguns itens. Isso fornece uma lista de exemplos de itens e confirma que há itens com o nome do atributo TTL fornecido junto com o prazo de validade.

Depois de habilitar a TTL, o atributo TTL é marcado como TTL quando você visualiza itens no console do DynamoDB. Você pode visualizar a data e a hora em que um item expira posicionando o mouse sobre o atributo.

Habilitar a TTL do DynamoDB usando a API

Python

É possível habilitar a TTL com código, usando a operação UpdateTimeToLive.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to the
 table.
    """
    try:
        dynamodb = boto3.client('dynamodb')

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={
                'Enabled': True,
                'AttributeName': ttl_attribute_name
            }
        )

        # In the returned response, check for a successful status code.
        if response['ResponseMetadata']['HTTPStatusCode'] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}")
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name, ex))
        raise


# your values
enable_ttl('your-table-name', 'expirationDate')
```

É possível confirmar se a TTL está habilitada usando a operação DescribeTimeToLive, que descreve o status da TTL em uma tabela. O status `TimeToLive` é `ENABLED` ou `DISABLED`.

```
# create a DynamoDB client
```

```
dynamodb = boto3.client('dynamodb')

# set the table name
table_name = 'YourTable'

# describe TTL
response = dynamodb.describe_time_to_live(TableName=table_name)
```

JavaScript

É possível habilitar a TTL com código, usando a operação UpdateTimeToLiveCommand.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
```

```
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Habilitar a vida útil usando a AWS CLI

1. Habilite o TTL na tabela `TTLExample`.

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. Descreva o TTL na tabela `TTLExample`.

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
    "TimeToLiveDescription": {
        "AttributeName": "ttl",
        "TimeToLiveStatus": "ENABLED"
    }
}
```

3. Para adicionar um item à tabela `TTLExample` com o conjunto de atributos de vida útil usando o shell BASH e a AWS CLI.

```
EXP=`date -d '+5 days' +%s`
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
 {"N": "'$EXP'"}}'
```

Este exemplo inicia na data atual e adiciona 5 dias a ela para criar uma data de expiração. Depois, ele converte a data de expiração em formato de hora epoch para finalmente adicionar um item à tabela "`TTLExample`".

> **Note**
>
> Uma forma de definir os valores de expiração para a vida útil é calcular o número de segundos para adicionar o tempo de expiração. Por exemplo, 5 dias é igual a 432.000 segundos. No entanto, muitas vezes é preferível começar com uma data e trabalhar a partir desse ponto.

É muito simples obter a hora atual no formato de hora epoch, como nos seguintes exemplos.

- Terminal Linux: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Habilitar a TTL do DynamoDB usando o AWS CloudFormation

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TTLExampleTable:
    Type: AWS::DynamoDB::Table
    Description: "A DynamoDB table with TTL Specification enabled"
    Properties:
      AttributeDefinitions:
        - AttributeName: "Album"
          AttributeType: "S"
        - AttributeName: "Artist"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "Album"
          KeyType: "HASH"
        - AttributeName: "Artist"
          KeyType: "RANGE"
      ProvisionedThroughput:
        ReadCapacityUnits: "5"
        WriteCapacityUnits: "5"
      TimeToLiveSpecification:
        AttributeName: "TTLExampleAttribute"
        Enabled: true
```

Detalhes adicionais sobre o uso do TTL em seus modelos do AWS CloudFormation podem ser encontrados aqui.

Como calcular a vida útil (TTL) no DynamoDB

Uma forma comum de implementar a TTL é definir um prazo de validade para os itens com base em quando eles foram criados ou atualizados pela última vez. Isso pode ser feito adicionando hora aos carimbos de data e hora `createdAt` e `updatedAt`. Por exemplo, a TTL para itens recém-criados pode ser definida como `createdAt` + noventa dias. Quando o item é atualizado, a TTL pode ser recalculada para `updatedAt` + noventa dias.

O prazo de validade calculado deve estar no formato de época, em segundos. Para ser considerada para validade e exclusão, a TTL não pode ter mais de cinco anos no passado. Se você usar qualquer outro formato, os processos TTL ignorarão o item. Se você definir a data de validade como algum momento no futuro em que quiser que o item expire, o item vai expirar após esse período. Por exemplo, digamos que você defina a data de validade como 1724241326 (que é segunda-feira, 21 de agosto de 2024, 11:55:26 (GMT)). O item vai expirar após o horário especificado.

Tópicos

- Criar um item e definir a vida útil
- Atualizar um item e atualizar a vida útil

Criar um item e definir a vida útil

O exemplo a seguir demonstra como calcular o prazo de validade ao criar um item, usando `expireAt` como nome do atributo TTL. Uma declaração de atribuição exibe a hora atual como uma variável. No exemplo, o prazo de validade é calculado como noventa dias a partir do horário atual. A hora é então convertida no formato de época e salva como um tipo de dados inteiro no atributo TTL.

Os exemplos de código a seguir mostram como criar um item com TTL.

Java

SDK para Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
```

```
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
```

```
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
```

```
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
```

```
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };
```

```
    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.
```

```
    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e


# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Para obter detalhes da API, consulte PutItem na Referência da API do AWS SDK para Python (Boto3).

Atualizar um item e atualizar a vida útil

Este exemplo é uma continuação do exemplo da seção anterior. O prazo de validade poderá ser recalculado se o item for atualizado. O exemplo a seguir recalcula o carimbo de data e hora `expireAt` para ser noventa dias a partir da hora atual.

Os exemplos de código a seguir mostram como atualizar a TTL de um item.

Java

SDK para Java 2.x

Atualize a TTL em um item do DynamoDB existente em uma tabela.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
```

```
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";
```

```
export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
```

```
)
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Os exemplos de TTL abordados nesta introdução demonstram um método que garante que somente os itens atualizados recentemente sejam mantidos em uma tabela. Os itens atualizados têm sua vida útil estendida, enquanto os itens não atualizados após a criação expiram e são excluídos sem nenhum custo, reduzindo o armazenamento e mantendo as tabelas limpas.

Trabalhar com itens expirados e vida útil (TTL)

Os itens expirados que estão pendentes de exclusão podem ser filtrados das operações de leitura e de gravação. Isso é útil em situações em que os dados expirados não são mais válidos e não devem ser usados. Se não forem filtrados, continuarão sendo exibidos nas operações de leitura e de gravação até serem excluídos pelo processo em segundo plano.

> **Note**
>
> Esses itens ainda contam em relação aos custos de armazenamento e de leitura até serem excluídos.

As exclusões de TTL podem ser identificadas no DynamoDB Streams, mas somente na região em que a exclusão ocorreu. As exclusões de TTL que são replicadas em regiões da tabela global não podem ser identificadas nos fluxos do DynamoDB nas regiões nas quais a exclusão é replicada.

Filtrar itens expirados das operações de leitura

Em relação a operações de leitura, como Scan e Query, uma expressão de filtro pode filtrar itens expirados que estão pendentes de exclusão. Conforme mostrado no trecho de código a seguir, a expressão de filtro pode filtrar itens em que a hora de TTL é igual ou menor que a hora atual. Por exemplo, o código do SDK para Python inclui uma declaração de atribuição que tem a hora atual como uma variável (`now`) e a converte em `int` para o formato de hora de época.

Os exemplos de código a seguir mostram como consultar itens com TTL.

Java

SDK para Java 2.x

Use uma expressão filtrada na consulta para reunir os itens com TTL em uma tabela do DynamoDB por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            throw e;
        }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Use uma expressão filtrada na consulta para reunir os itens com TTL em uma tabela do DynamoDB por meio do AWS SDK para JavaScript.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
```

```
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Use uma expressão filtrada na consulta para reunir os itens com TTL em uma tabela do DynamoDB por meio do AWS SDK para Python (Boto3).

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
```

```
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Gravar de modo condicional em itens expirados

Uma expressão de condição pode ser usada para evitar gravações em itens expirados. O trecho de código abaixo é uma atualização condicional que confere se o prazo de validade é maior que o horário atual. Se verdadeiro, a operação de gravação continuará.

Os exemplos de código a seguir mostram como atualizar condicionalmente a TTL de um item.

Java

SDK para Java 2.x

Atualize a TTL em um item do DynamoDB existente em uma tabela, com uma condição.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
```

```
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String UPDATED_AT_ATTR = "updatedAt";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
    private static final String CONDITION_EXPRESSION = "attribute_exists(" +
PRIMARY_KEY_ATTR + ")";
    private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
    private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs an UpdateTTLConditional with a default DynamoDB client.
     */
    public UpdateTTLConditional() {
        this.dynamoDbClient = null;
    }

    /**
     * Constructs an UpdateTTLConditional with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());
```

```
        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Atualize a TTL em um item do DynamoDB existente em uma tabela, com uma condição.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
```

```
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Atualize a TTL em um item do DynamoDB existente em uma tabela, com uma condição.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"
```

```
        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Identificar itens excluídos no DynamoDB Streams

O registro de fluxos contém um campo de identidade do usuário `Records[<index>].userIdentity`. Os itens excluídos pelo processo do TTL têm os seguintes campos:

```
Records[<index>].userIdentity.type
```

```
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

O JSON a seguir mostra a parte relevante de um único registro de fluxos.

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

Consultar tabelas no DynamoDB

Você pode usar a operação de API `Query` no Amazon DynamoDB para encontrar itens com base nos valores de chave primária.

Você deve fornecer o nome do atributo de chave de partição e um único valor para esse atributo. A operação `Query` retorna todos os itens com esse valor de chave de partição. Opcionalmente, você pode fornecer um atributo de chave de classificação e usar um operador de comparação para refinar os resultados da pesquisa.

Para obter mais informações sobre como usar `Query`, como a sintaxe de solicitação, parâmetros de resposta e exemplos adicionais, acesse Consulta na Referência da API do Amazon DynamoDB.

Tópicos

- Principais expressões de condição para a operação de consulta no DynamoDB
- Filtrar expressões para a operação de consulta no DynamoDB
- Paginar os resultados de consultas à tabela no DynamoDB
- Outros aspectos do trabalho com a operação de consulta no DynamoDB

Principais expressões de condição para a operação de consulta no DynamoDB

Você pode usar qualquer nome de atributo em uma expressão de condição principal, desde que o primeiro caractere seja `a-z` ou `A-Z` e o restante dos caracteres (a partir do segundo caractere, se houver) seja `a-z`, `A-Z` ou `0-9`. Além disso, o nome do atributo não deve ser uma palavra reservada do DynamoDB. (Para obter uma lista completa dessas palavras reservadas, consulte Palavras reservadas no DynamoDB.) Se um nome de atributo não atender a esses requisitos, defina um nome de atributo de expressão como um espaço reservado. Para obter mais informações, consulte Nomes (aliases) de atributo de expressão no DynamoDB.

Para itens com um determinado valor de chave de partição, o DynamoDB armazena esses itens juntos em ordem classificada por valor de chave de classificação. Em uma operação `Query`, o DynamoDB recupera os itens na ordem classificada e, em seguida, processa os itens usando `KeyConditionExpression` e qualquer `FilterExpression` que estiver presente. Somente então os resultados de `Query` são enviados de volta para o cliente.

Uma operação `Query` sempre retorna um conjunto de resultados. Se não forem encontrados itens correspondentes, o conjunto de resultados estará vazio.

Os resultados de `Query` são sempre classificados pelo valor de chave de classificação. Se o tipo de dados da chave de classificação for `Number`, os resultados serão retornados em ordem numérica. Caso contrário, os resultados serão retornados na ordem de bytes UTF-8. Por padrão, a ordem de classificação é crescente. Para reverter a ordem, defina o parâmetro `ScanIndexForward` como `false`.

Uma única operação `Query` pode recuperar um máximo de 1 MB de dados. Esse limite se aplica antes de qualquer expressão `FilterExpression` ou `ProjectionExpression` ser aplicada aos resultados. Se `LastEvaluatedKey` estiver presente na resposta e não for nula, você deverá paginar o conjunto de resultados (consulte Paginar os resultados de consultas à tabela no DynamoDB).

Exemplos das principais expressões de condição

Para especificar os critérios de pesquisa, você deve usar uma expressão de condição principal – uma string que determina os itens a serem lidos da tabela ou índice.

Você deve especificar o nome e o valor da chave de partição como uma condição de igualdade. Não é possível usar um atributo que não seja de chave em uma expressão de condição de chave.

Opcionalmente, você pode fornecer uma segunda condição para a chave de classificação (se houver). A condição de chave de classificação deve usar um dos seguintes operadores de comparação:

- *a* = *b* - verdadeiro se o atributo *a* for igual ao valor *b*
- *a* < *b*: verdadeiro se *a* for menor que *b*
- *a* <= *b*: verdadeiro se *a* for menor que ou igual a *b*
- *a* > *b*: verdadeiro se *a* for maior que *b*
- *a* >= *b*: verdadeiro se *a* for maior ou igual a *b*
- *a* BETWEEN *b* AND *c* - verdadeiro se *a* for maior ou igual a *b*e menor ou igual a *c*.

A função a seguir também tem suporte:

- begins_with (*a*, *substr*)- verdadeiro se o valor do atributo *a* começa com uma determinada substring.

Os seguintes exemplos da AWS Command Line Interface (AWS CLI) demonstram o uso das expressões de condição de chave. Essas expressões usam espaços reservados (como :name e :sub), em vez de valores reais. Para obter mais informações, consulte Nomes (aliases) de atributo de expressão no DynamoDB e Usar valores de atributos de expressão no DynamoDB.

Example

Consulte a tabela Thread para encontrar um ForumName específico (chave de partição). Todos os itens com esse valor de ForumName são lidos pela consulta, porque a chave de classificação (Subject) não está incluída na KeyConditionExpression.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Consulte a tabela Thread para encontrar um ForumName específico (chave de partição), mas, desta vez, retornar apenas os itens com um determinado Subject (chave de classificação).

```
aws dynamodb query \
```

```
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

Consulte a tabela `Reply` para encontrar um `Id` específico (chave de partição), mas retornar apenas os itens cuja `ReplyDateTime` (chave de classificação) começa com determinados caracteres.

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
}
```

Filtrar expressões para a operação de consulta no DynamoDB

Se você precisar refinar ainda mais os resultados de `Query`, existe a opção de utilizar uma expressão de filtro. Uma expressão de filtro determina quais itens dos resultados de `Query` devem ser retornados para você. Todos os outros resultados serão descartados.

Uma expressão de filtro é aplicada depois que uma operação `Query` é concluída, mas antes que os resultados sejam retornados. Portanto, uma operação `Query` consome a mesma quantidade de capacidade de leitura, independentemente de uma expressão de filtro estar presente.

Uma operação `Query` pode recuperar um máximo de 1 MB de dados. Esse limite se aplica antes de a expressão de filtro ser avaliada.

Uma expressão de filtro não pode conter atributos de chave de partição ou de chave de classificação. É necessário especificar esses atributos na expressão de condição principal, não na de filtro.

A sintaxe de uma expressão de filtro é semelhante à de uma expressão de condição de chave. As expressões de filtro podem usar os mesmos comparadores, funções e operadores lógicos que uma expressão de condição principal. Além disso, as expressões de filtro podem usar o operador diferente de (<>), bem como os operadores `OR`, `CONTAINS`, `IN`, `BEGINS_WITH`, `BETWEEN`, `EXISTS` e `SIZE`. Para obter mais informações, consulte Principais expressões de condição para a operação de consulta no DynamoDB e Sintaxe para expressões de filtro e de condição.

Example

O exemplo da AWS CLI a seguir consulta a tabela `Thread` para encontrar um `ForumName` (chave de partição) e um `Subject` (chave de classificação) específicos. Dos itens que são encontrados, somente os threads de discussão mais populares são retornados – em outras palavras, apenas os threads com mais de um determinado número de `Views`.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject begins_with :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

Os argumentos de `--expression-attribute-values` são armazenados no arquivo `values.json`.

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Observe que `Views` é uma palavra reservada no DynamoDB (consulte Palavras reservadas no DynamoDB), portanto, este exemplo usa #v como um espaço reservado. Para obter mais informações, consulte Nomes (aliases) de atributo de expressão no DynamoDB.

> **Note**
>
> Uma expressão de filtro remove itens do conjunto de resultados de `Query`. Se possível, evite usar `Query` onde você espera recuperar um grande número de itens, mas também precisa descartar a maioria desses itens.

Paginar os resultados de consultas à tabela no DynamoDB

O DynamoDB pagina os resultados das operações `Query`. Com a paginação, os resultados de `Query` são divididos em "páginas" de dados com 1 MB de tamanho (ou menos). Uma aplicação pode processar a primeira página de resultados e, em seguida, a segunda página, e assim por diante.

Uma única operação `Query` retorna apenas um conjunto de resultados que estão dentro do limite de tamanho de 1 MB. Para determinar se há mais resultados e para recuperá-los em uma página por vez, os aplicativos devem fazer o seguinte:

1. Examine o resultado de `Query` de baixo nível:
   - Se o resultado contiver um elemento `LastEvaluatedKey` e ele não for nulo, prossiga para a etapa 2.
   - Se não houver um `LastEvaluatedKey` no resultado, não haverá mais itens a serem recuperados.
2. Crie uma `Query` com a mesma `KeyConditionExpression`. No entanto, desta vez, use o valor de `LastEvaluatedKey` da etapa 1 como o parâmetro `ExclusiveStartKey` na nova solicitação de `Query`.
3. Execute a nova solicitação de `Query`.
4. Vá para a etapa 1.

Em outras palavras, o valor `LastEvaluatedKey` de uma resposta de `Query` deve ser usado como `ExclusiveStartKey` da próxima solicitação de `Query`. Se não houver um elemento `LastEvaluatedKey` em uma resposta de `Query`, então, você recuperou a página de resultados final. Se `LastEvaluatedKey` não está vazio, isso não necessariamente significa que há mais dados no conjunto de resultados. A única maneira de saber quando você atingiu o final do conjunto de resultados é quando `LastEvaluatedKey` estiver vazio.

Você pode usar a AWS CLI para visualizar esse comportamento. A AWS CLI envia repetidamente solicitações `Query` de baixo nível ao DynamoDB até que `LastEvaluatedKey` não esteja mais

presente nos resultados. Considere o seguinte exemplo da AWS CLI que recupera títulos de filmes de um determinado ano.

```
aws dynamodb query --table-name Movies \
    --projection-expression "title" \
    --key-condition-expression "#y = :yyyy" \
    --expression-attribute-names '{"#y":"year"}' \
    --expression-attribute-values '{":yyyy":{"N":"1993"}}' \
    --page-size 5 \
    --debug
```

Normalmente, a AWS CLI lida com a paginação automaticamente. No entanto, neste exemplo, o parâmetro `--page-size` da AWS CLI limita o número de itens por página. O parâmetro de `--debug` imprime as informações de baixo nível sobre solicitações e respostas.

Se você executar o exemplo, a primeira resposta do DynamoDB será semelhante a esta.

```
2017-07-07 11:13:15,603 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":5,"Items":[{"title":{"S":"A Bronx Tale"}},
{"title":{"S":"A Perfect World"}},{"title":{"S":"Addams Family Values"}},
{"title":{"S":"Alive"}},{"title":{"S":"Benny & Joon"}}],
"LastEvaluatedKey":{"year":{"N":"1993"},"title":{"S":"Benny & Joon"}},
"ScannedCount":5}'
```

O `LastEvaluatedKey` na resposta indica que nem todos os itens foram recuperados. A AWS CLI emite outra solicitação de `Query` para o DynamoDB. Essa solicitação e o padrão de resposta continuam, até a resposta final.

```
2017-07-07 11:13:16,291 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"What\'s Eating Gilbert
 Grape"}}],"ScannedCount":1}'
```

A ausência de `LastEvaluatedKey` indica que não há mais itens a serem recuperados.

> **Note**
>
> Os AWS SDKs lidam com as respostas de baixo nível do DynamoDB (incluindo a presença ou a ausência de `LastEvaluatedKey`) e fornecem várias abstrações para paginar os resultados de `Query` Por exemplo, a interface do documento SDK para Java fornece o suporte a `java.util.Iterator` para que você possa abordar um resultado de cada vez.

> Para obter exemplos de código em várias linguagens de programação, consulte o Guia de conceitos básicos do Amazon DynamoDB e a documentação do AWS SDK para sua linguagem.

Você também pode reduzir o tamanho da página limitando o número de itens no conjunto de resultados, com o parâmetro `Limit` da operação `Query`.

Para obter mais informações sobre como realizar consultas com o DynamoDB, veja Consultar tabelas no DynamoDB.

Outros aspectos do trabalho com a operação de consulta no DynamoDB

Esta seção aborda aspectos adicionais da operação de consulta do DynamoDB, incluindo a limitação do tamanho dos resultados, a contagem de itens varridos versus retornados, o monitoramento do consumo da capacidade de leitura e o controle da consistência de leitura.

Limitar o número de itens no conjunto de resultados

Com a operação `Query`, você pode limitar o número de itens lidos. Para fazer isso, defina o parâmetro `Limit` com o número máximo de itens que você deseja.

Por exemplo, suponha que você execute a operação `Query` em uma tabela, com um valor de `Limit` igual a 6 e sem uma expressão de filtro. O resultado de `Query` contém os primeiros seis itens da tabela que correspondem à expressão de condição da chave da solicitação.

Agora suponha que você adicione uma expressão de filtro à operação `Query`. Nesse caso, o DynamoDB lê até seis itens e retorna somente aqueles que correspondem à expressão do filtro. O resultado final de `Query` contém seis itens ou menos, mesmo que mais itens tivessem correspondido à expressão do filtro se o DynamoDB continuasse lendo mais itens.

Contar os itens nos resultados

Além dos itens que correspondem aos seus critérios, a resposta de `Query` contém os elementos a seguir:

- `ScannedCount`: o número de itens que corresponderam à expressão de condição principal, antes que uma expressão de filtro (se alguma) fosse aplicada.
- `Count`: o número de itens que permaneceram depois que uma expressão de filtro (se alguma) foi aplicada.

> **Note**
>
> Se você não usar uma expressão de filtro, `ScannedCount` e `Count` terão o mesmo valor.

Se o tamanho do conjunto de resultados de `Query` for maior que 1 MB, `ScannedCount` e `Count` representarão apenas uma contagem parcial do total de itens. Você precisa executar várias operações `Query` para recuperar todos os resultados (consulte Paginar os resultados de consultas à tabela no DynamoDB).

Cada resposta de `Query` contém os valores de `ScannedCount` e de `Count` dos itens que foram processados pela solicitação de `Query` específica. Para obter os totais gerais de todas as solicitações de `Query`, você pode manter um total em execução de `ScannedCount` e `Count`.

Unidades de capacidade consumidas por consulta

Você pode usar `Query` em qualquer tabela ou índice secundário, desde que forneça o nome do atributo da chave de partição e um valor único para esse atributo. `Query` retorna todos os itens com esse valor de chave de partição. Opcionalmente, você pode fornecer um atributo de chave de classificação e usar um operador de comparação para refinar os resultados da pesquisa. `Query` As operações de API consomem unidades de capacidade de leitura da seguinte forma:

Se você executar uma operação **`Query`**...	O DynamoDB consumirá unidades de capacidade de leitura de...
Tabela	A capacidade de leitura provisionada da tabela.
Índice secundário global	A capacidade de leitura provisionada do índice.
Índice secundário local	A capacidade de leitura provisionada da tabela-base.

Por padrão, uma operação `Query` não retorna quaisquer dados sobre quanta capacidade de leitura ela consome. Entretanto, você pode especificar o parâmetro `ReturnConsumedCapacity` em uma solicitação de `Query` para obter essas informações. A seguir estão as configurações válidas de `ReturnConsumedCapacity`:

- `NONE`: nenhum dado de capacidade consumida é retornado. (Esse é o padrão.)

- `TOTAL`: a resposta inclui o número agregado de unidades de capacidade de leitura consumidas.
- `INDEXES`: a resposta mostra o número agregado de unidades de capacidade de leitura consumidas, junto com a capacidade consumida para cada tabela e índice acessados.

O DynamoDB calcula o número de unidades de capacidade de leitura consumidas com base na quantidade e no tamanho desses itens, não na quantidade de dados exibidos para uma aplicação. Por esse motivo, o número de unidades de capacidade consumidas será o mesmo, independentemente de você solicitar todos os atributos (o comportamento padrão) ou apenas alguns deles (usando uma expressão de projeção). O número também é o mesmo, independentemente de você usar ou não uma expressão de filtro. `Query` consome uma unidade de capacidade mínima de leitura para realizar uma leitura altamente consistente por segundo ou duas leituras finais consistentes por segundo para um item de até 4 KB. Se você precisar ler um item com mais de 4 KB, o DynamoDB precisará de unidades de solicitação de leitura adicionais. Tabelas vazias e tabelas muito grandes que têm uma quantidade esparsa de chaves de partição podem ter algumas RCUs adicionais cobradas além da quantidade de dados consultados. Isso cobre o custo de atender à solicitação `Query`, mesmo que não existam dados.

Consistência de leitura para consulta

Uma operação `Query` executa leituras finais consistentes, por padrão. Isso significa que os resultados de `Query` talvez não reflitam as alterações causadas pelas operações `PutItem` ou `UpdateItem` concluídas recentemente. Para obter mais informações, consulte Consistência de leitura do DynamoDB.

Se você precisar de leituras fortemente consistentes, defina o parâmetro `ConsistentRead` como `true` na solicitação de `Query`.

Verificar tabelas no DynamoDB

Uma operação `Scan` no Amazon DynamoDB lê cada item de uma tabela ou de um índice secundário. Por padrão, uma operação `Scan` retorna todos os atributos de dados de cada item na tabela ou índice. Você pode usar o parâmetro `ProjectionExpression` para que a operação `Scan` retorne apenas alguns dos atributos, em vez de todos eles.

`Scan` sempre retorna um conjunto de resultados. Se não forem encontrados itens correspondentes, o conjunto de resultados estará vazio.

Uma única solicitação de `Scan` pode recuperar um máximo de 1 MB de dados. Opcionalmente, o DynamoDB pode aplicar uma expressão de filtro a esses dados, restringindo os resultados antes que eles sejam retornados ao usuário.

Tópicos

- Expressões de filtro para verificação
- Limitar o número de itens no conjunto de resultados
- Paginar resultados
- Contar os itens nos resultados
- Unidades de capacidade consumidas por verificação
- Consistência de leitura para verificação
- Verificar em paralelo

Expressões de filtro para verificação

Se você precisar refinar ainda mais os resultados de `Scan`, existe a opção de utilizar uma expressão de filtro. Uma expressão de filtro determina quais itens dos resultados de `Scan` devem ser retornados para você. Todos os outros resultados serão descartados.

Uma expressão de filtro é aplicada depois que uma operação `Scan` é concluída, mas antes que os resultados sejam retornados. Portanto, uma operação `Scan` consome a mesma quantidade de capacidade de leitura, independentemente de uma expressão de filtro estar presente.

Uma operação `Scan` pode recuperar um máximo de 1 MB de dados. Esse limite se aplica antes de a expressão de filtro ser avaliada.

Com `Scan`, você pode especificar quaisquer atributos em uma expressão de filtro, incluindo atributos de chave de partição ou de chave de classificação.

A sintaxe de uma expressão de filtro é idêntica à de uma expressão de condição. As expressões de filtro podem usar os mesmos comparadores, funções e operadores lógicos que uma expressão de condição. Consulte Expressões de condição e filtro, operadores e funções no DynamoDB para obter mais informações sobre operadores lógicos.

Example

O exemplo da AWS Command Line Interface (AWS CLI) a seguir verifica a tabela `Thread` e retorna apenas os itens que foram publicados pela última vez por um usuário específico.

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

Limitar o número de itens no conjunto de resultados

A operação `Scan` permite que você limite o número de itens retornados no resultado. Para fazer isso, defina o parâmetro `Limit` como o número máximo de itens que você deseja que a operação `Scan` retorne, antes da avaliação da expressão de filtro.

Por exemplo, suponha que você execute a operação `Scan` em uma tabela, com um valor de `Limit` igual a 6 e sem uma expressão de filtro. O resultado `Scan` contém os primeiros seis itens da tabela.

Agora suponha que você adicione uma expressão de filtro à operação `Scan`. Nesse caso, o DynamoDB aplicará a expressão de filtro aos seis itens que foram retornados e descarta os que não correspondem. O resultado final de `Scan` contém 6 itens ou menos, dependendo do número de itens que foram filtrados.

Paginar resultados

O DynamoDB pagina os resultados das operações `Scan`. Com a paginação, os resultados de `Scan` são divididos em "páginas" de dados com 1 MB de tamanho (ou menos). Uma aplicação pode processar a primeira página de resultados e, em seguida, a segunda página, e assim por diante.

Uma única operação `Scan` retorna apenas um conjunto de resultados que estão dentro do limite de tamanho de 1 MB.

Para determinar se há mais resultados e para recuperá-los em uma página por vez, as aplicações devem fazer o seguinte:

1. Examine o resultado de `Scan` de baixo nível:
   - Se o resultado contiver um elemento `LastEvaluatedKey`, prossiga para a etapa 2.
   - Se não houver um `LastEvaluatedKey` no resultado, não haverá mais itens a serem recuperados.
2. Construa a nova solicitação de `Scan` com os mesmos parâmetros da anterior. No entanto, desta vez, use o valor de `LastEvaluatedKey` da etapa 1 como o parâmetro `ExclusiveStartKey` na nova solicitação de `Scan`.
3. Execute a nova solicitação de `Scan`.

4. Vá para a etapa 1.

Em outras palavras, o valor `LastEvaluatedKey` de uma resposta de `Scan` deve ser usado como `ExclusiveStartKey` da próxima solicitação de `Scan`. Se não houver um elemento `LastEvaluatedKey` em uma resposta de `Scan`, você terá recuperado a página de resultados final. (A ausência de `LastEvaluatedKey` é a única forma de você saber que chegou ao fim do conjunto de resultados.)

Você pode usar a AWS CLI para visualizar esse comportamento. A AWS CLI envia repetidamente solicitações `Scan` de baixo nível ao DynamoDB até que `LastEvaluatedKey` não esteja mais presente nos resultados. Considere o seguinte exemplo da AWS CLI que verifica toda a tabela `Movies`, mas retorna apenas os filmes de um determinado gênero.

```
aws dynamodb scan \
    --table-name Movies \
    --projection-expression "title" \
    --filter-expression 'contains(info.genres,:gen)' \
    --expression-attribute-values '{":gen":{"S":"Sci-Fi"}}' \
    --page-size 100  \
    --debug
```

Normalmente, a AWS CLI lida com a paginação automaticamente. No entanto, neste exemplo, o parâmetro `--page-size` da AWS CLI limita o número de itens por página. O parâmetro de `--debug` imprime as informações de baixo nível sobre solicitações e respostas.

> **Note**
>
> Os resultados da paginação também serão diferentes com base nos parâmetros de entrada que você passar.
>
> - O uso de `aws dynamodb scan --table-name Prices --max-items 1` retorna `NextToken`
> - O uso de `aws dynamodb scan --table-name Prices --limit 1` retorna `LastEvaluatedKey`.
>
> Também esteja ciente de que usar`--starting-token`, em especial, requer o valor `NextToken`.

Se você executar o exemplo, a primeira resposta do DynamoDB será semelhante ao seguinte:

```
2017-07-07 12:19:14,389 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":7,"Items":[{"title":{"S":"Monster on the Campus"}},{"title":{"S":"+1"}},
{"title":{"S":"100 Degrees Below Zero"}},{"title":{"S":"About Time"}},{"title":
{"S":"After Earth"}},
{"title":{"S":"Age of Dinosaurs"}},{"title":{"S":"Cloudy with a Chance of Meatballs
 2"}}],
"LastEvaluatedKey":{"year":{"N":"2013"},"title":{"S":"Curse of
 Chucky"}},"ScannedCount":100}'
```

O `LastEvaluatedKey` na resposta indica que nem todos os itens foram recuperados. A AWS CLI emite outra solicitação de `Scan` para o DynamoDB. Essa solicitação e o padrão de resposta continuam, até a resposta final.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

A ausência de `LastEvaluatedKey` indica que não há mais itens a serem recuperados.

> **Note**
>
> Os AWS SDKs lidam com as respostas de baixo nível do DynamoDB (incluindo a presença ou a ausência de `LastEvaluatedKey`) e fornecem várias abstrações para paginar os resultados de `Scan` Por exemplo, a interface do documento SDK para Java fornece o suporte a `java.util.Iterator` para que você possa abordar um resultado de cada vez.
> Para obter exemplos de código em várias linguagens de programação, consulte o Guia de conceitos básicos do Amazon DynamoDB e a documentação do AWS SDK para sua linguagem.

Contar os itens nos resultados

Além dos itens que correspondem aos seus critérios, a resposta de `Scan` contém os elementos a seguir:

- `ScannedCount`: o número de itens avaliados antes de qualquer `ScanFilter` ser aplicado. Um valor alto de `ScannedCount` com poucos ou nenhum resultado de `Count` indica uma operação

`Scan` ineficiente. Se você não tiver usado um filtro na solicitação, `ScannedCount` será o mesmo que `Count`.

- `Count`: o número de itens que permaneceram depois que uma expressão de filtro (se alguma) foi aplicada.

> **Note**
>
> Se você não usar uma expressão de filtro, `ScannedCount` e `Count` terão o mesmo valor.

Se o tamanho do conjunto de resultados de `Scan` for maior que 1 MB, `ScannedCount` e `Count` representarão apenas uma contagem parcial do total de itens. Você precisa executar várias operações `Scan` para recuperar todos os resultados (consulte Paginar resultados).

Cada resposta de `Scan` contém os valores de `ScannedCount` e de `Count` dos itens que foram processados pela solicitação de `Scan` específica. Para obter os totais gerais de todas as solicitações de `Scan`, você pode manter um total em execução de `ScannedCount` e de `Count`.

Unidades de capacidade consumidas por verificação

Você pode executar `Scan` em qualquer tabela ou índice secundário. As operações `Scan` consomem unidades de capacidade de leitura da seguinte forma.

Se você executar uma operação **`Scan`**...	O DynamoDB consumirá unidades de capacidade de leitura de...
Tabela	A capacidade de leitura provisionada da tabela.
Índice secundário global	A capacidade de leitura provisionada do índice.
Índice secundário local	A capacidade de leitura provisionada da tabela-base.

> **Note**
>
> Atualmente, o acesso entre contas para operações de verificação de índice secundário não é aceito em políticas baseadas em recursos.

Por padrão, uma operação `Scan` não retorna quaisquer dados sobre quanta capacidade de leitura ela consome. Entretanto, você pode especificar o parâmetro `ReturnConsumedCapacity` em uma solicitação de `Scan` para obter essas informações. A seguir estão as configurações válidas de `ReturnConsumedCapacity`:

- `NONE`: nenhum dado de capacidade consumida é retornado. (Esse é o padrão.)
- `TOTAL`: a resposta inclui o número agregado de unidades de capacidade de leitura consumidas.
- `INDEXES`: a resposta mostra o número agregado de unidades de capacidade de leitura consumidas, junto com a capacidade consumida para cada tabela e índice acessados.

O DynamoDB calcula o número de unidades de capacidade de leitura consumidas com base na quantidade e no tamanho desses itens, não na quantidade de dados exibidos para uma aplicação. Por esse motivo, o número de unidades de capacidade consumidas será o mesmo, independentemente de você solicitar todos os atributos (o comportamento padrão) ou apenas alguns deles (usando uma expressão de projeção). O número também é o mesmo, independentemente de você usar ou não uma expressão de filtro. `Scan` consome uma unidade de capacidade mínima de leitura para realizar uma leitura altamente consistente por segundo ou duas leituras finais consistentes por segundo para um item de até 4 KB. Se você precisar ler um item com mais de 4 KB, o DynamoDB precisará de unidades de solicitação de leitura adicionais. Tabelas vazias e tabelas muito grandes que têm uma quantidade esparsa de chaves de partição podem ter algumas RCUs adicionais cobradas além da quantidade de dados verificados. Isso cobre o custo de atender à solicitação `Scan`, mesmo que não existam dados.

Consistência de leitura para verificação

Uma operação `Scan` executa leituras finais consistentes, por padrão. Isso significa que os resultados de `Scan` talvez não reflitam as alterações causadas pelas operações `PutItem` ou `UpdateItem` concluídas recentemente. Para ter mais informações, consulte Consistência de leitura do DynamoDB.

Se você precisar de leituras altamente consistentes, como a hora em que a operação `Scan` começa, defina o parâmetro `ConsistentRead` como `true` na solicitação de `Scan`. Isso garante que todas as operações de gravação que foram concluídas antes de `Scan` ter começado sejam incluídas na resposta de `Scan`.

Pode ser útil configurar `ConsistentRead` como `true` no backup da tabela ou em cenários de replicação, juntamente com o Amazon DynamoDB Streams. Primeiro, você usa `Scan` com

`ConsistentRead` definida como true para obter uma cópia consistente dos dados na tabela. Durante a operação `Scan`, o DynamoDB Streams registra qualquer atividade de gravação adicional que ocorra na tabela. Após a conclusão de `Scan`, você pode aplicar a atividade de gravação do fluxo à tabela.

> **Note**
>
> Uma operação `Scan` com `ConsistentRead` definida como `true` consome duas vezes mais unidades de capacidade de leitura, em comparação com deixar `ConsistentRead` com seu valor padrão (`false`).

Verificar em paralelo

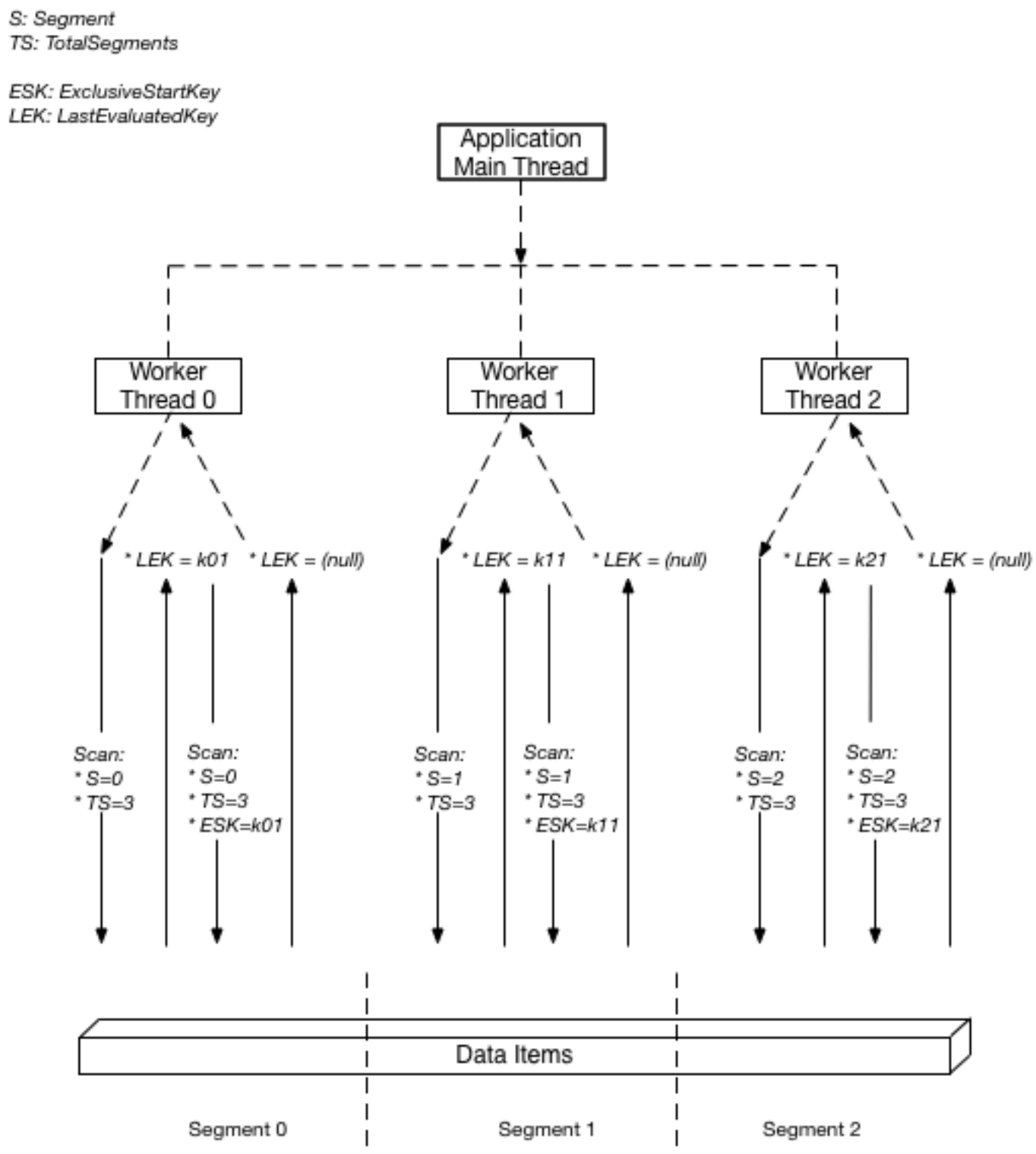
Por padrão, a operação `Scan` processa os dados sequencialmente. O Amazon DynamoDB retorna os dados para a aplicação em incrementos de 1 MB, e uma aplicação realiza operações `Scan` adicionais para recuperar os próximos 1 MB de dados.

Quanto maior a tabela ou índice que está sendo verificado, mais tempo a operação `Scan` levará para ser concluída. Além disso, uma operação `Scan` sequencial talvez nem sempre consiga utilizar totalmente a capacidade de throughput de leitura provisionada: embora o DynamoDB distribua os dados de uma tabela grande entre várias partições físicas, uma operação `Scan` pode ler apenas uma partição de cada vez. Por esse motivo, o throughput de uma operação `Scan` é restrita pelo throughput máximo de uma única partição.

Para solucionar esses problemas, a operação `Scan` pode dividir logicamente uma tabela ou um índice secundário em vários segmentos, com vários operadores da aplicação verificando os segmentos em paralelo. Cada operador pode ser um thread (em linguagens de programação que aceitam multithreading) ou um processo do sistema operacional. Para executar uma verificação em paralelo, cada operador emite sua própria solicitação de `Scan` com os seguintes parâmetros:

- `Segment`: um segmento a ser verificado por um determinado operador. Cada operador deve usar um valor diferente para `Segment`.
- `TotalSegments`: o número total de segmentos para a verificação em paralelo. Esse valor deve ser o mesmo que o número de operadores que sua aplicação usará.

O diagrama a seguir mostra como uma aplicação multithread executa uma operação `Scan` em paralelo com três níveis de paralelismo:



Neste diagrama, a aplicação gera três threads e atribui um número a cada thread. (Segmentos são baseados em zero, portanto, o primeiro número é sempre 0.) Cada thread emite uma solicitação de `Scan`, configurando `Segment` como seu número designado e `TotalSegments` como 3. Cada thread

verifica seu segmento designado, recuperando 1 MB de dados por vez, e retorna os dados para o thread principal da aplicação.

Os valores para `Segment` e `TotalSegments` aplicam-se a solicitações de `Scan` individuais, e você pode usar valores diferentes a qualquer momento. Talvez você precise experimentar com esses valores, e o número de operadores que usa, até que seu aplicativo atinja a melhor performance.

> **Note**
>
> Uma operação Scan em paralelo com um grande número de operadores pode facilmente consumir todo o throughput provisionado da tabela ou índice que está sendo verificado. É melhor evitar tais verificações, se a tabela ou índice também incorrer em uso intensivo de atividades de leitura ou de gravação de outras aplicações.
> Para controlar a quantidade de dados retornados por solicitação, use o parâmetro `Limit`. Isso pode ajudar a evitar situações em que um operador consome todo o throughput provisionado, às custas de todos os outros operadores.

PartiQL: uma linguagem de consultas compatível com SQL para o Amazon DynamoDB

O Amazon DynamoDB oferece suporte a PartiQL, uma linguagem de consultas compatível com SQL, para selecionar, inserir, atualizar e excluir dados no Amazon DynamoDB. Usando PartiQL, você pode interagir facilmente com tabelas do DynamoDB e executar consultas ad hoc usando o AWS Management Console, o NoSQL Workbench, a AWS Command Line Interfacee as APIs do DynamoDB para PartiQL.

As operações PartiQL fornecem as mesmas disponibilidade, latência e performance que as outras operações de plano de dados do DynamoDB.

As seções a seguir descrevem a implementação de PartiQL do DynamoDB.

Tópicos

- O que é PartiQL?
- PartiQL no Amazon DynamoDB
- Conceitos básicos de PartiQL para DynamoDB
- Tipos de dados de PartiQL para DynamoDB
- Instruções em PartiQL para DynamoDB

- Usar funções PartiQL com o DynamoDB
- Operadores aritméticos, comparativos e lógicos de PartiQL para DynamoDB
- Executar transações com PartiQL para DynamoDB
- Executar operações em lote com PartiQL para DynamoDB
- Políticas de segurança do IAM com PartiQL para DynamoDB

O que é PartiQL?

A linguagem PartiQL garante acesso de consultas compatíveis com SQL em vários armazenamentos de dados que contêm dados estruturados, dados semiestruturados e dados aninhados. Ela é amplamente utilizada na Amazon e agora está disponível como parte de muitos serviços da AWS, incluindo o DynamoDB.

Para obter a especificação da PartiQL e um tutorial sobre a linguagem de consulta principal, consulte a Documentação da PartiQL.

> **Note**
>
> - O Amazon DynamoDB oferece suporte a um subconjunto da linguagem de consultas PartiQL.
> - O Amazon DynamoDB não é compatível com o formato de dados Amazon Ion nem com liderais do Amazon Ion.

PartiQL no Amazon DynamoDB

Para executar consultas PartiQL no DynamoDB, você pode usar:

- O console do DynamoDB
- O NoSQL Workbench
- A AWS Command Line Interface (AWS CLI)
- As APIs do DynamoDB

Para obter informações sobre como usar esses métodos para acessar o DynamoDB, consulte Acesso ao DynamoDB.

Conceitos básicos de PartiQL para DynamoDB

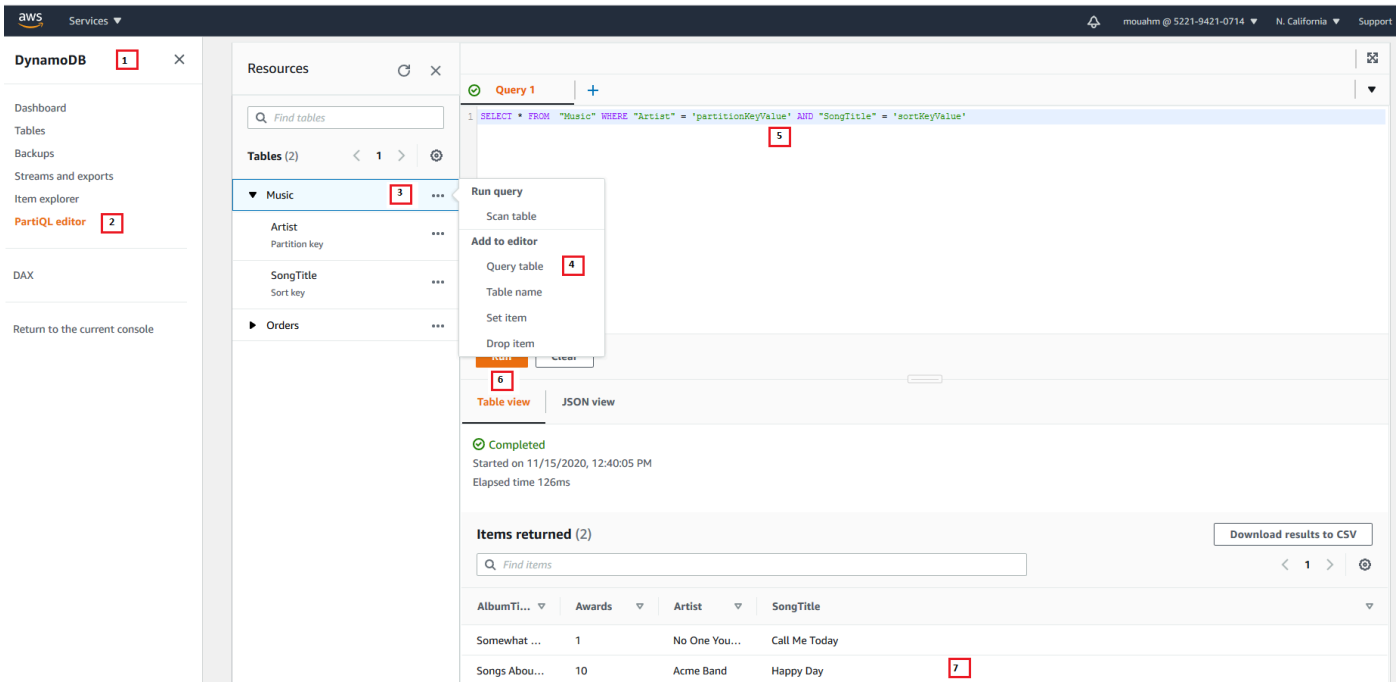
Esta seção descreve como usar PartiQL para DynamoDB via console do Amazon DynamoDB, AWS Command Line Interface (AWS CLI) e APIs do DynamoDB.

Nos exemplos a seguir, a tabela do DynamoDB definida no tutorial Conceitos básicos do DynamoDB é um pré-requisito.

Para obter informações sobre como usar o console do DynamoDB, a AWS Command Line Interface ou as APIs do DynamoDB para acessar o DynamoDB, consulte Acesso ao DynamoDB.

Para baixar e usar o NoSQL Workbench para criar instruções em PartiQL para DynamoDB, escolha PartiQL Operations (Operações PartiQL) no canto superior direito do Criador de operações do NoSQL Workbench para DynamoDB.

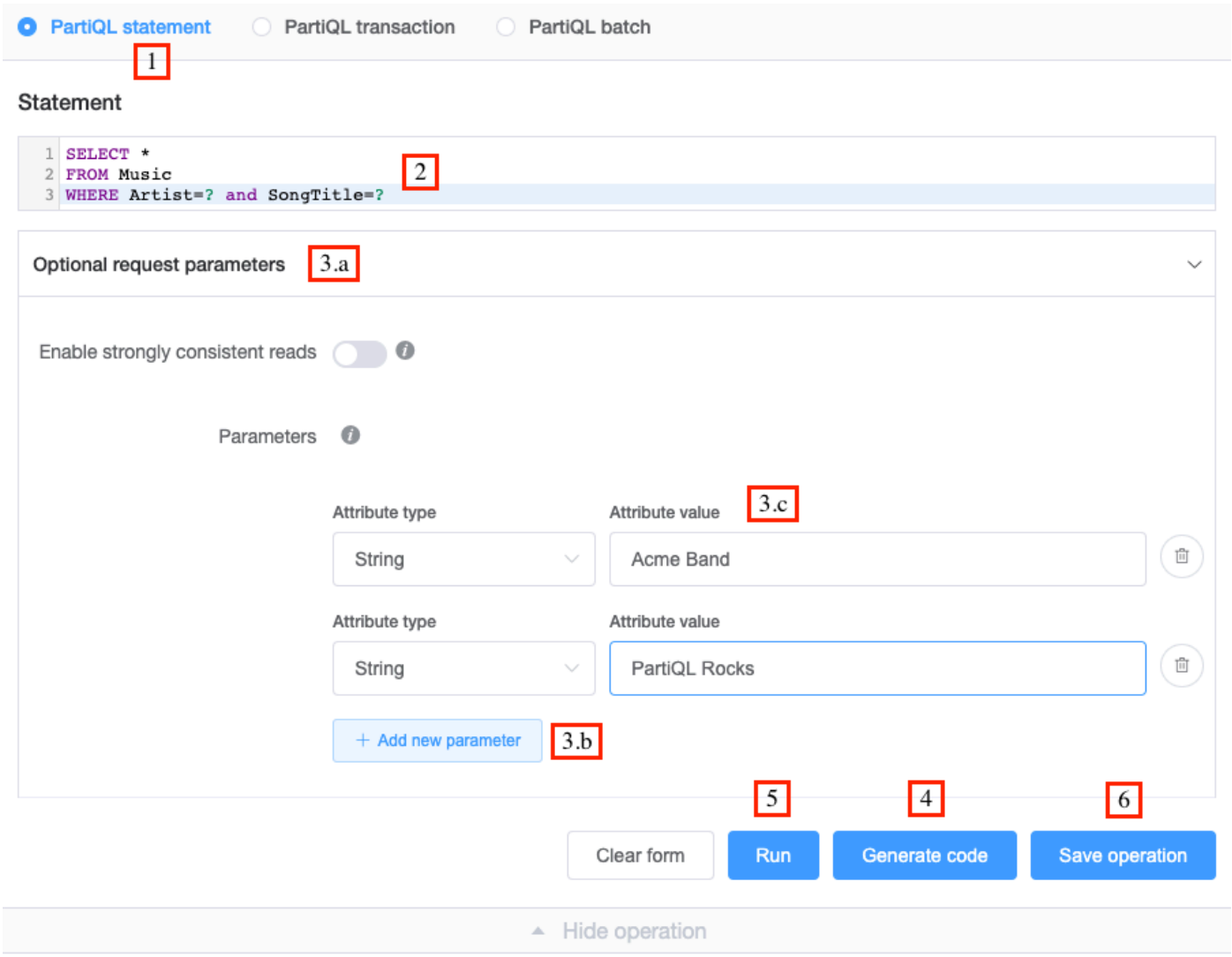
Console



1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione PartiQL editor (Editor de PartiQL).
3. Escolha a tabela Music.
4. Escolha Query table (Consultar tabela). Essa ação gera uma consulta que não resultará em uma varredura de tabela completa.

5. Substitua `partitionKeyValue` pelo valor da string `Acme Band`. Substitua `sortKeyValue` pelo valor da string `Happy Day`.
6. Selecione o botão Run (Executar).
7. Você pode visualizar os resultados da consulta escolhendo a opção Table view (Visualização da tabela ou JSON view (Visualização de JSON).

NoSQL workbench



1. Escolha PartiQL statement (Instrução PartiQL).
2. Insira a seguinte instrução SELECT PartiQL

```
SELECT *
FROM Music
```

```
WHERE Artist=? and SongTitle=?
```

3. Para especificar um valor para os parâmetros `Artist` e `SongTitle`:

   a. Escolha Optional request parameters (Parâmetros de solicitação opcionais).

   b. Escolha Add new parameters (Adicionar novos parâmetros).

   c. Escolha o tipo de atributo string e o valor `Acme Band`.

   d. Repita as etapas b e c e escolha o tipo string e o valor `PartiQL Rocks`.

4. Se desejar gerar código, escolha Generate code (Gerar código).

   Selecione a linguagem desejada nas guias exibidas. Agora você pode copiar esse código e usá-lo na sua aplicação.

5. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).

AWS CLI

1. Crie um item na tabela `Music` usando a instrução PartiQL INSERT.

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. Recupere um item da tabela Music usando a instrução PartiQL SELECT.

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

3. Atualize um item na tabela `Music` usando a instrução PartiQL UPDATE.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   Adicione um valor de lista para um item na tabela `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Remova um valor de lista para um item na tabela `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Adicione um novo membro de mapa para um item na tabela `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Adicione um novo atributo de conjunto de strings para um item na tabela `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Atualize um atributo de conjunto de strings para um item na tabela `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. Exclua um item da tabela `Music` usando a instrução PartiQL DELETE.

```
aws dynamodb execute-statement --statement "DELETE  FROM Music  \
```

```
    WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Java

```
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.model.AttributeValue;
import software.amazon.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.dynamodb.model.ExecuteStatementRequest;
import software.amazon.dynamodb.model.ExecuteStatementResult;
import software.amazon.dynamodb.model.InternalServerErrorException;
import software.amazon.dynamodb.model.ItemCollectionSizeLimitExceededException;
import software.amazon.dynamodb.model.ProvisionedThroughputExceededException;
import software.amazon.dynamodb.model.RequestLimitExceededException;
import software.amazon.dynamodb.model.ResourceNotFoundException;
import software.amazon.dynamodb.model.TransactionConflictException;

public class DynamoDBPartiQGettingStarted {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

        try {
            // Create ExecuteStatementRequest
            ExecuteStatementRequest executeStatementRequest = new
 ExecuteStatementRequest();
            List<AttributeValue> parameters= getPartiQLParameters();

            //Create an item in the Music table using the INSERT PartiQL statement
            processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
 value {'Artist':?,'SongTitle':?}", parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
 statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
 where Artist=? and SongTitle=?", parameters));
```

```
            //Update an item in the Music table using the UPDATE PartiQL statement.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist=? and
SongTitle=?", parameters));

            //Add a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  where Artist=? and
SongTitle=?", parameters));

            //Remove a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2]   where Artist=? and SongTitle=?", parameters));

            //Add a new map member for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

            //Add a new string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

            //update a string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

            //delete an item from the Music Table
            processResults(executeStatementRequest(dynamoDB, "DELETE  FROM Music
where Artist=? and SongTitle=?", parameters));
        } catch (Exception e) {
            handleExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
```

```
   }

   private static List<AttributeValue> getPartiQLParameters() {
       List<AttributeValue> parameters = new ArrayList<AttributeValue>();
       parameters.add(new AttributeValue("Acme Band"));
       parameters.add(new AttributeValue("PartiQL Rocks"));
       return parameters;
   }

   private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters ) {
       ExecuteStatementRequest request = new ExecuteStatementRequest();
       request.setStatement(statement);
       request.setParameters(parameters);
       return client.executeStatement(request);
   }

   private static void processResults(ExecuteStatementResult
executeStatementResult) {
       System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());

   }

   // Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (ConditionalCheckFailedException ccfe) {
           System.out.println("Condition check specified in the operation failed,
review and update the condition " +
                                      "check before retrying. Error: " +
ccfe.getErrorMessage());
       } catch (TransactionConflictException tce) {
           System.out.println("Operation was rejected because there is an ongoing
transaction for the item, generally " +
                                      "safe to retry with exponential back-off.
Error: " + tce.getErrorMessage());
       } catch (ItemCollectionSizeLimitExceededException icslee) {
           System.out.println("An item collection is too large, you\'re using Local
Secondary Index and exceeded " +
```

```
                                        "size limit of items per
partition key. Consider using Global Secondary Index instead. Error: " +
icslee.getErrorMessage());
       } catch (Exception e) {
           handleCommonErrors(e);
       }
   }

   private static void handleCommonErrors(Exception exception) {
       try {
           throw exception;
       } catch (InternalServerErrorException isee) {
           System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
       } catch (RequestLimitExceededException rlee) {
           System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                                        "retrying. Error: " +
rlee.getErrorMessage());
       } catch (ProvisionedThroughputExceededException ptee) {
           System.out.println("Request rate is too high. If you're using a custom
retry strategy make sure to retry with exponential back-off. " +
                                        "Otherwise consider reducing frequency of
requests or increasing provisioned capacity for your table or secondary index.
Error: " +
                                        ptee.getErrorMessage());
       } catch (ResourceNotFoundException rnfe) {
           System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getErrorMessage());
       } catch (AmazonServiceException ase) {
           System.out.println("An AmazonServiceException occurred, indicates that
the request was correctly transmitted to the DynamoDB " +
                                        "service, but for some reason, the service
was not able to process it, and returned an error response instead. Investigate and
" +
                                        "configure retry strategy. Error type: " +
ase.getErrorType() + ". Error message: " + ase.getErrorMessage());
       } catch (AmazonClientException ace) {
           System.out.println("An AmazonClientException occurred, indicates that
the client was unable to get a response from DynamoDB " +
                                        "service, or the client was unable to parse
the response from the service. Investigate and configure retry strategy. "+
                                        "Error: " + ace.getMessage());
       } catch (Exception e) {
```

```
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

Tipos de dados de PartiQL para DynamoDB

A tabela a seguir lista os tipos de dados que você pode usar com a linguagem PartiQL para DynamoDB.

Tipo de dados do DynamoDB	Representação em PartiQL	Observações
`Boolean`	TRUE | FALSE	Não diferencia maiúsculas de minúsculas.
`Binary`	N/D	Suporte oferecido somente via código.
`List`	[valor1, valor2,...]	Não há restrições quanto aos tipos de dados que podem ser armazenados em um tipo List, e os elementos em uma lista não precisam ser do mesmo tipo.
`Map`	{'nome': valor}	Não há restrições quanto aos tipos de dados que podem ser armazenados em um tipo Map, e os elementos em um mapa não precisam ser do mesmo tipo.
`Null`	NULL	Não diferencia maiúsculas de minúsculas.
`Number`	1, 1.0, 1e0	Números podem ser positivos, negativo ou zero. Os números

Tipo de dados do DynamoDB	Representação em PartiQL	Observações
		podem ter uma precisão de até 38 dígitos.
`Number Set`	<<número1, número2>>	Os elementos em um conjunto de números devem ser do tipo Number.
`String Set`	<<'string1', 'string2'>>	Os elementos em um conjunto de strings devem ser do tipo String.
`String`	'valor da string'	Valores de String devem ser especificados entre aspas simples.

Exemplos

A instrução a seguir demonstra como inserir os seguintes tipos de dados: `String`, `Number`, `Map`, `List`, `Number Set` e `String Set`.

```
INSERT INTO TypesTable value {'primarykey':'1',
'NumberType':1,
'MapType' : {'entryname1': 'value', 'entryname2': 4},
'ListType': [1,'stringval'],
'NumberSetType':<<1,34,32,4.5>>,
'StringSetType':<<'stringval','stringval2'>>
}
```

A instrução a seguir demonstra como inserir novos elementos nos tipos `Map`, `List`, `Number Set` e `String Set` e alterar o valor de um tipo `Number`.

```
UPDATE TypesTable
SET NumberType=NumberType + 100
SET MapType.NewMapEntry=[2020, 'stringvalue', 2.4]
SET ListType = LIST_APPEND(ListType, [4, <<'string1', 'string2'>>])
SET NumberSetType= SET_ADD(NumberSetType, <<345, 48.4>>)
SET StringSetType = SET_ADD(StringSetType, <<'stringsetvalue1', 'stringsetvalue2'>>)
WHERE primarykey='1'
```

A instrução a seguir demonstra como remover elementos dos tipos `Map`, `List`, `Number Set` e `String Set` e alterar o valor de um tipo `Number`.

```
UPDATE TypesTable
SET NumberType=NumberType - 1
REMOVE ListType[1]
REMOVE MapType.NewMapEntry
SET NumberSetType = SET_DELETE( NumberSetType, <<345>>)
SET StringSetType = SET_DELETE( StringSetType, <<'stringsetvalue1'>>)
WHERE primarykey='1'
```

Para obter mais informações, consulte Tipos de dados do DynamoDB.

Instruções em PartiQL para DynamoDB

O Amazon DynamoDB oferece suporte às instruções PartiQL a seguir.

> **Note**
>
> O DynamoDB não oferece suporte a todas as instruções PartiQL.
> Esta referência apresenta a sintaxe básica e fornece exemplos de uso de instruções PartiQL que você deve executar manualmente usando o AWS CLI ou APIs.

Linguagem de manipulação de dados (DML) é o conjunto de instruções PartiQL que você usa para gerenciar dados em tabelas do DynamoDB. Você usa instruções DML para adicionar, modificar ou excluir dados em uma tabela.

As seguintes instruções de linguagem DML e consultas são aceitas:

- Instruções Select em PartiQL para DynamoDB
- Instruções Update em PartiQL para DynamoDB
- Instruções Insert em PartiQL para DynamoDB
- Instruções Delete em PartiQL para DynamoDB

Executar transações com PartiQL para DynamoDB e Executar operações em lote com PartiQL para DynamoDB também são compatíveis com a linguagem PartiQL para DynamoDB.

Instruções Select em PartiQL para DynamoDB

Use a instrução `SELECT` para recuperar dados de uma tabela no Amazon DynamoDB.

O uso da declaração `SELECT` poderá gerar uma verificação completa da tabela se uma condição de igualdade ou IN com uma chave de partição não for fornecida na cláusula WHERE. Uma operação de verificação examina todos os itens com os valores solicitados e pode usar o throughput provisionado para uma tabela ou índice grande em uma única operação.

Se desejar evitar a verificação completa da tabela em PartiQL, você pode:

- Criar suas instruções `SELECT` para não resultar em verificações completas de tabela, certificando-se de que a condição da cláusula WHERE seja configurada de acordo.
- Desativar verificações completas de tabela usando a política do IAM especificada em Exemplo: permitir instruções Select e negar instruções de verificação de tabela completa em PartiQL para DynamoDB, conforme descrito no Guia do desenvolvedor do DynamoDB.

Para obter mais informações, consulte Melhores práticas para consulta e verificação de dados no Guia do desenvolvedor do DynamoDB.

Tópicos

- Sintaxe
- Parâmetros
- Exemplos

Sintaxe

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parâmetros

expressão

(Obrigatório) Uma projeção formada a partir do curinga * ou uma lista de projeção de um ou mais nomes de atributos ou caminhos de documentos do conjunto de resultados. Uma expressão

pode consistir em chamadas para Usar funções PartiQL com o DynamoDB ou campos que são modificados por Operadores aritméticos, comparativos e lógicos de PartiQL para DynamoDB .

tabela

(Obrigatório) O nome da tabela a ser consultada.

índice

(Opcional) O nome do índice para consultar.

> **Note**
>
> É necessário adicionar aspas duplas ao nome da tabela e ao nome do índice ao consultar um índice.
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condição

(Opcional) Os critérios de seleção para a consulta.

> **Important**
>
> Para garantir que uma instrução `SELECT` não resulte em uma verificação completa da tabela, a condição da cláusula `WHERE` deverá especificar uma chave de partição. Use o operador de igualdade ou IN.
> Por exemplo, se você tiver uma tabela `Orders` com uma partição `OrderID` e outros atributos não chave, incluindo um `Address`, as seguintes instruções não resultarão em uma varredura completa da tabela:
>
> ```
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100
>
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100 and Address='some address'
>
> SELECT *
> ```

```
FROM "Orders"
WHERE OrderID = 100 or OrderID = 200

SELECT *
FROM "Orders"
WHERE OrderID IN [100, 300, 234]
```

As instruções SELECT a seguir, no entanto, resultarão em uma varredura completa da tabela:

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

chave

(Opcional) Uma chave de hash ou uma chave de classificação a ser usada para ordenar os resultados retornados. A ordem padrão é crescente (ASC). Especifique DESC se desejar que os resultados sejam retornados em ordem decrescente.

> **Note**
>
> Se você omitir a cláusula WHERE, todos os itens da tabela serão recuperados.

Exemplos

A consulta a seguir retorna um item, se houver, da tabela `Orders` mediante a especificação da chave de partição `OrderID` e o uso do operador de igualdade.

```
SELECT OrderID, Total
```

```
FROM "Orders"
WHERE OrderID = 1
```

A consulta a seguir retorna todos os itens na tabela `Orders` que têm uma chave de partição específica, `OrderID`, valores usando o operador OR.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

A consulta a seguir retorna todos os itens na tabela `Orders` que têm uma chave de partição específica, `OrderID`, valores usando o operador IN. Os resultados retornados estão em ordem decrescente, com base no valor de atributo de chave `OrderID`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

A consulta a seguir mostra uma varredura de tabela completa que retorna todos os itens da tabela `Orders`que têm uma `Total` maior que 500, onde `Total` é um atributo não chave.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

A consulta a seguir mostra uma varredura de tabela completa que retorna todos os itens da tabela `Orders` em um intervalo de ordem `Total` específico, usando o operador IN e um atributo não chave `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

A consulta a seguir mostra uma varredura de tabela completa que retorna todos os itens da tabela `Orders` em um intervalo de ordem `Total` específico, usando o operador BETWEEN e um atributo não chave `Total`.

```
SELECT OrderID, Total
```

```
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

A consulta a seguir retorna a primeira data em que um dispositivo firestick foi usado para observar mediante a especificação da chave de partição `CustomerID` e da chave de classificação `MovieID` na condição da cláusula WHERE e usando caminhos de documento na cláusula SELECT.

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

A consulta a seguir mostra uma verificação de tabela completa que retorna a lista de itens em que um dispositivo firestick foi usado pela primeira vez após 24/12/19 usando caminhos de documento na condição da cláusula WHERE.

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

Instruções Update em PartiQL para DynamoDB

Use a instrução `UPDATE` para modificar o valor de um ou mais atributos em um item em uma tabela do Amazon DynamoDB.

> **Note**
>
> Você só pode atualizar um item de cada vez; não é possível emitir uma única instrução PartiQL do DynamoDB para atualizar vários itens. Para obter informações sobre como atualizar vários itens, consulte Executar transações com PartiQL para DynamoDB ou Executar operações em lote com PartiQL para DynamoDB.

Tópicos

- Sintaxe
- Parâmetros
- Valor de retorno
- Exemplos

Sintaxe

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parâmetros

tabela

(Obrigatório) A tabela que contém os dados a serem modificados.

path

(Obrigatório) Um nome de atributo ou caminho de documento a ser criado ou modificado.

data

(Obrigatório) Um valor de atributo ou o resultado de uma operação.

As operações com suporte a serem usadas com SET:

- LIST_APPEND: adiciona um valor a um tipo de lista.
- SET_ADD: adiciona um valor a um conjunto de números ou strings.
- SET_DELETE: remove um valor de um conjunto de números ou strings.

condição

(Obrigatório) Os critérios de seleção para o item a ser modificado. Essa condição deve ser resolvida em um único valor de chave primária.

returnvalues

(Opcional) Use `returnvalues` se desejar obter os atributos de item como eles aparecem antes ou depois de serem atualizados. Os valores válidos são:

- `ALL OLD *`: retorna todos os atributos do item como eles apareciam antes da operação de atualização.
- `MODIFIED OLD *`: retorna somente os atributos atualizados como eles apareciam antes da operação de atualização.
- `ALL NEW *`: retorna todos os atributos do item como eles aparecem após a operação de atualização.

- `MODIFIED NEW *` : retorna somente os atributos atualizados como eles aparecem após a operação `UpdateItem`.

Valor de retorno

Esta instrução não retornará um valor a menos que o parâmetro `returnvalues` seja especificado.

> **Note**
>
> Se a cláusula WHERE da instrução UPDATE não for avaliada como true para nenhum item na tabela do DynamoDB, `ConditionalCheckFailedException` será retornado.

Exemplos

Atualiza um valor de atributo em um item existente. Se o atributo não existir, ele será criado.

A consulta a seguir atualiza um item na tabela `"Music"` adicionando um atributo do tipo Number (`AwardsWon`) e um atributo do tipo Map (`AwardDetail`).

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Você pode adicionar `RETURNING ALL OLD *` para retornar os atributos como eles apareceram antes da operação `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

Isso retorna o seguinte:

```
{
    "Items": [
        {
```

```
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Você pode adicionar `RETURNING ALL NEW *` para retornar os atributos como eles apareceram depois da operação `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

Isso retorna o seguinte:

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
```

```
}
```

A consulta a seguir atualiza um item na tabela `"Music"` anexando-o a uma lista `AwardDetail.Grammys`.

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

A consulta a seguir atualiza um item na tabela `"Music"` removendo-o de uma lista `AwardDetail.Grammys`.

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

A consulta a seguir atualiza um item na tabela `"Music"` adicionando `BillBoard` ao mapa `AwardDetail`.

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

A consulta a seguir atualiza um item na tabela `"Music"` adicionando o atributo de conjunto de strings `BandMembers`.

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

A consulta a seguir atualiza um item na tabela `"Music"` adicionando `newbandmember` ao conjunto de strings `BandMembers`.

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Instruções Delete em PartiQL para DynamoDB

Use a instrução `DELETE` para excluir um item existente da tabela do Amazon DynamoDB.

> **Note**
>
> É possível excluir apenas um item de cada vez. Você não pode emitir uma única instrução PartiQL do DynamoDB para excluir vários itens. Para obter informações sobre como excluir vários itens, consulte Executar transações com PartiQL para DynamoDB ou Executar operações em lote com PartiQL para DynamoDB.

Tópicos

- Sintaxe
- Parâmetros
- Valor de retorno
- Exemplos

Sintaxe

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

Parâmetros

tabela

(Obrigatório) A tabela do DynamoDB que contém o item a ser excluído.

condição

(Obrigatório) Os critérios de seleção para o item a ser excluído; essa condição deve ser resolvida para um único valor de chave primária.

returnvalues

(Opcional) Use `returnvalues` se desejar obter os atributos do item como eles apareciam antes de ser excluídos. Os valores válidos são:

- `ALL OLD *`: o conteúdo do item antigo é retornado.

Valor de retorno

Esta instrução não retornará um valor a menos que o parâmetro `returnvalues` seja especificado.

> **Note**
>
> Se a tabela do DynamoDB não tiver nenhum item com a mesma chave primária que a do item para o qual a instrução DELETE foi emitida, SUCESS será retornado com 0 itens excluídos. Se a tabela tiver um item com a mesma chave primária, mas a condição na cláusula WHERE da instrução DELETE for avaliada como false, `ConditionalCheckFailedException` será retornado.

Exemplos

A consulta a seguir exclui um item da tabela `"Music"`.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

Você pode adicionar o parâmetro `RETURNING ALL OLD *` para retornar os dados que foram excluídos.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

A instrução `Delete` agora retorna o seguinte:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Instruções Insert em PartiQL para DynamoDB

Use a instrução `INSERT` para adicionar um item a uma tabela no Amazon DynamoDB.

> **Note**
>
> Você só pode inserir um item de cada vez; não é possível emitir uma única instrução PartiQL do DynamoDB para inserir vários itens. Para obter informações sobre como inserir vários itens, consulte Executar transações com PartiQL para DynamoDB ou Executar operações em lote com PartiQL para DynamoDB.

Tópicos

- Sintaxe
- Parâmetros
- Valor de retorno
- Exemplos

Sintaxe

Insira um único item.

```
INSERT INTO table VALUE item
```

Parâmetros

tabela

(Obrigatório) A tabela na qual você deseja inserir os dados. A tabela já deve existir.

item

(Obrigatório) Um item válido do DynamoDB representado como uma tupla PartiQL. Você só deve especificar um item, e cada nome de atributo no item diferencia maiúsculas e minúsculas e pode ser indicado com aspas simples (`'...'`) na linguagem PartiQL.

Os valores de string também são denotados com aspas simples (`'...'`) em PartiQL.

Valor de retorno

Esta instrução não retorna nenhum valor.

> **Note**
>
> Se a tabela do DynamoDB já tiver um item com a mesma chave primária da chave primária do item que está sendo inserido, `DuplicateItemException` será retornado.

Exemplos

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

Usar funções PartiQL com o DynamoDB

A linguagem PartiQL no Amazon DynamoDB é compatível com as seguintes variantes integradas de funções padrão SQL.

> **Note**
>
> As funções SQL que não fazem parte desta lista não são aceitas no DynamoDB.

Funções agregadas

- Usar a função SIZE com PartiQL para Amazon DynamoDB

Funções condicionais

- Usar a função EXISTS com PartiQL para DynamoDB
- Usar a função ATTRIBUTE_TYPE com PartiQL para DynamoDB
- Usar a função BEGINS_WITH com PartiQL para DynamoDB
- Usar a função CONTAINS com PartiQL para DynamoDB
- Usar a função MISSING com PartiQL para DynamoDB

Usar a função EXISTS com PartiQL para DynamoDB

Você pode usar EXISTS para executar a mesma função que `ConditionCheck` na API TransactWriteItems. A função EXISTS só pode ser usada em transações.

Sendo fornecido um valor, a função retornará `TRUE` se o valor for uma coleção não vazia. Caso contrário, gera `FALSE`.

> **Note**
>
> Essa função só pode ser usada em operações transacionais.

Sintaxe

```
EXISTS ( statement )
```

Argumentos

instrução

(Obrigatório) A instrução SELECT que a função avalia.

> **Note**
>
> A instrução SELECT deve especificar uma chave primária completa e alguma outra condição.

Tipo de retorno

`bool`

Exemplos

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

Usar a função BEGINS_WITH com PartiQL para DynamoDB

Retorna TRUE quando o atributo especificado começa com uma substring específica.

Sintaxe

```
begins_with(path, value )
```

Argumentos

path

(Obrigatório) O nome do atributo ou o caminho do documento a ser usado.

value

(Obrigatório) A string a ser pesquisada.

Tipo de retorno

bool

Exemplos

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

Usar a função MISSING com PartiQL para DynamoDB

Retorna TRUE quando o item não contém o atributo especificado. Somente operadores de igualdade e desigualdade podem ser usados com esta função.

Sintaxe

```
 attributename IS | IS NOT  MISSING
```

Argumentos

attributename

(Obrigatório) O nome do atributo a ser procurado.

Tipo de retorno

`bool`

Exemplos

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

Usar a função ATTRIBUTE_TYPE com PartiQL para DynamoDB

Retorna TRUE quando o atributo no caminho especificado é de um tipo de dados específico.

Sintaxe

```
attribute_type( attributename, type )
```

Argumentos

attributename

(Obrigatório) O nome do atributo a ser usado.

tipo

(Obrigatório) O tipo de atributo a ser verificado. Para obter uma lista de valores válidos, consulte attribute_type do DynamoDB.

Tipo de retorno

`bool`

Exemplos

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

Usar a função CONTAINS com PartiQL para DynamoDB

Retorna TRUE quando o atributo especificado pelo caminho é um dos seguintes:

- Uma String que contém uma substring específica.
- Um Set que contém um elemento específico dentro do conjunto.

Para obter mais informações, consulte a função contains do DynamoDB.

Sintaxe

```
contains( path, substring )
```

Argumentos

path

(Obrigatório) O nome do atributo ou o caminho do documento a ser usado.

substring

(Obrigatório) A substring do atributo ou o membro do conjunto a ser verificado. Para obter mais informações, consulte a função contains do DynamoDB.

Tipo de retorno

`bool`

Exemplos

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Usar a função SIZE com PartiQL para Amazon DynamoDB

Retorna um número que representa o tamanho de um atributo em bytes. Veja a seguir os tipos de dados válidos para uso com SIZE. Para obter mais informações, consulte a função size do DynamoDB.

Sintaxe

```
size( path)
```

Argumentos

path

(Obrigatório) O nome do atributo ou o caminho do documento.

Para obter os tipos compatíveis, consulte a função size do DynamoDB.

Tipo de retorno

`int`

Exemplos

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

Operadores aritméticos, comparativos e lógicos de PartiQL para DynamoDB

A linguagem PartiQL no Amazon DynamoDB oferece suporte às instruções SQL padrão a seguir.

> **Note**
>
> Os operadores SQL que não fazem parte desta lista não são aceitas no DynamoDB.

Operadores aritméticos

Operador	Descrição
+	Adicionar
-	Subtrair

Operadores de comparação

Operador	Descrição
=	Igual a
<>	Não é igual a
!=	Não é igual a
>	Maior que

Operador	Descrição
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Operadores lógicos

Operador	Descrição
AND	TRUE se todas as condições separadas por AND forem TRUE
BETWEEN	TRUE se o operando estiver dentro do intervalo de comparações. Esse operador inclui os limites inferior e superior dos operandos nos quais você o aplica.
IN	TRUE se o operando for igual a um de uma lista de expressões (no máximo 50 valores de atributo hash ou no máximo 100 valores de atributo que não sejam chaves)
IS	TRUE se o operando for um determinado tipo de dados PartiQL, incluindo NULL ou MISSING
NOT	Reverte o valor de uma determinada expressão booleana
OR	TRUE se qualquer uma das condições separadas por OR for TRUE

Para obter mais informações sobre como usar operadores lógicos, consulte Fazer comparações e Avaliações lógicas.

Executar transações com PartiQL para DynamoDB

Esta seção descreve como usar transações com a linguagem PartiQL para DynamoDB. As transações PartiQL são limitadas a 100 instruções (ações) no total.

Para obter mais informações sobre transações do DynamoDB, consulte Gerenciar fluxos de trabalho complexos com transações do DynamoDB.

> **Note**
>
> A transação inteira deve ser composta por instruções de leitura ou instruções de gravação. Não é permitido misturar os dois em uma transação. A função EXISTS é uma exceção. Você pode usá-la para verificar a condição de atributos específicos do item de uma maneira semelhante a `ConditionCheck` na operação TransactWriteItems da API.

Tópicos

- Sintaxe
- Parâmetros
- Valores de retorno
- Exemplos

Sintaxe

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parâmetros

instrução

(Obrigatório) Uma instrução valida na linguagem PartiQL para DynamoDB.

> **Note**
>
> A transação inteira deve ser composta por instruções de leitura ou instruções de gravação. Não é permitido misturar os dois em uma transação.

parametertype

(Opcional) Um tipo do DynamoDB, se parâmetros foram usados ao especificar a instrução PartiQL.

parametervalue

(Opcional) Um valor de parâmetro, se parâmetros foram usados ao especificar a instrução PartiQL.

Valores de retorno

Esta instrução não retorna nenhum valor para operações de gravação (INSERT, UPDATE ou DELETE). No entanto, ela retorna valores diferentes para operações de leitura (SELECT) com base nas condições especificadas na cláusula WHERE.

> **Note**
>
> Se qualquer uma das operações singleton INSERT, UPDATE ou DELETE retornar um erro, as transações serão canceladas com a exceção `TransactionCanceledException` e o código de motivo de cancelamento incluirá os erros das operações singleton individuais.

Exemplos

O exemplo a seguir executa várias instruções como uma transação.

AWS CLI

1. Salve o seguinte código JSON em um arquivo chamado partiql.json.

```
[
    {
        "Statement": "EXISTS(SELECT * FROM \"Music\" where Artist='No One You
 Know' and SongTitle='Call Me Today' and Awards is  MISSING)"
```

```
    },
    {
        "Statement": "INSERT INTO Music value {'Artist':?,'SongTitle':'?'}",
        "Parameters": [{\"S\": \"Acme Band\"}, {\"S\": \"Best Song\"}]
    },
    {
        "Statement": "UPDATE \"Music\" SET AwardsWon=1 SET
 AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
 SongTitle='PartiQL Rocks'"
    }
]
```

2. Execute o comando a seguir em um prompt de comando.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlTransaction {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create ExecuteTransactionRequest
            ExecuteTransactionRequest executeTransactionRequest =
 createExecuteTransactionRequest();
            ExecuteTransactionResult executeTransactionResult =
 dynamoDB.executeTransaction(executeTransactionRequest);
            System.out.println("ExecuteTransaction successful.");
            // Handle executeTransactionResult

        } catch (Exception e) {
            handleExecuteTransactionErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }
```

```
   private static ExecuteTransactionRequest createExecuteTransactionRequest() {
       ExecuteTransactionRequest request = new ExecuteTransactionRequest();

       // Create statements
       List<ParameterizedStatement> statements = getPartiQLTransactionStatements();

       request.setTransactStatements(statements);
       return request;
   }

   private static List<ParameterizedStatement> getPartiQLTransactionStatements() {
       List<ParameterizedStatement> statements = new
ArrayList<ParameterizedStatement>();

       statements.add(new ParameterizedStatement()
                               .withStatement("EXISTS(SELECT * FROM "Music" where
Artist='No One You Know' and SongTitle='Call Me Today' and Awards is  MISSING)"));

       statements.add(new ParameterizedStatement()
                               .withStatement("INSERT INTO "Music" value
{'Artist':'?','SongTitle':'?'}")
                               .withParameters(new AttributeValue("Acme Band"),new
AttributeValue("Best Song")));

       statements.add(new ParameterizedStatement()
                               .withStatement("UPDATE "Music" SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
SongTitle='PartiQL Rocks'"));

       return statements;
   }

   // Handles errors during ExecuteTransaction execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleExecuteTransactionErrors(Exception exception) {
       try {
           throw exception;
       } catch (TransactionCanceledException tce) {
           System.out.println("Transaction Cancelled, implies a client issue, fix
before retrying. Error: " + tce.getErrorMessage());
       } catch (TransactionInProgressException tipe) {
           System.out.println("The transaction with the given request token is
already in progress, consider changing " +
```

```
                "retry strategy for this type of error. Error: " +
tipe.getErrorMessage());
        } catch (IdempotentParameterMismatchException ipme) {
            System.out.println("Request rejected because it was retried with a
different payload but with a request token that was already used, " +
                "change request token for this payload to be accepted. Error: " +
ipme.getErrorMessage());
        } catch (Exception e) {
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
```

```
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

O exemplo a seguir mostra os diferentes valores de retorno quando o DynamoDB lê itens com diferentes condições especificadas na cláusula WHERE.

AWS CLI

1. Salve o seguinte código JSON em um arquivo chamado partiql.json.

```
[
    // Item exists and projected attribute exists
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One You Know' and
 SongTitle='Call Me Today'"
    },
    // Item exists but projected attributes do not exist
    {
        "Statement": "SELECT non_existent_projected_attribute FROM "Music" WHERE
 Artist='No One You Know' and SongTitle='Call Me Today'"
    },
    // Item does not exist
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One I Know' and
 SongTitle='Call You Today'"
    }
]
```

2. comando a seguir em um prompt de comando.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. A resposta a seguir será retornada:

```
{
    "Responses": [
        // Item exists and projected attribute exists
```

```
        {
            "Item": {
                "Artist":{
                    "S": "No One You Know"
                },
                "SongTitle":{
                    "S": "Call Me Today"
                }
            }
        },
        // Item exists but projected attributes do not exist
        {
            "Item": {}
        },
        // Item does not exist
        {}
    ]
}
```

Executar operações em lote com PartiQL para DynamoDB

Esta seção descreve como usar instruções em lote com a linguagem PartiQL para DynamoDB.

> **Note**
>
> - O lote inteiro deve consistir em instruções de leitura ou instruções de gravação; não é possível misturar ambas em um lote.
> - `BatchExecuteStatement` e `BatchWriteItem` podem executar mais de 25 instruções por lote.
> - O `BatchExecuteStatement` faz uso do `BatchGetItem` que utiliza uma lista de chaves primárias em declarações separadas.

Tópicos

- Sintaxe
- Parâmetros
- Exemplos

Sintaxe

```
[
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#9StkWHYTxm7x2AqSXcrfu7' AND
 sk = 'info'"
  },
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#isC2ChceGbxHgESc4szoTE' AND
 sk = 'info'"
  }
]
```

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parâmetros

instrução

(Obrigatório) Uma instrução valida na linguagem PartiQL para DynamoDB.

> **Note**
>
> - O lote inteiro deve consistir em instruções de leitura ou instruções de gravação; não é possível misturar ambas em um lote.
> - `BatchExecuteStatement` e `BatchWriteItem` podem executar mais de 25 instruções por lote.

parametertype

(Opcional) Um tipo do DynamoDB, se parâmetros foram usados ao especificar a instrução PartiQL.

parametervalue

(Opcional) Um valor de parâmetro, se parâmetros foram usados ao especificar a instrução PartiQL.

Exemplos

AWS CLI

1. Salve o seguinte json em um arquivo chamado partiql.json

```
[
   {
  "Statement": "INSERT INTO Music VALUE {'Artist':?,'SongTitle':?}",
   "Parameters": [{"S": "Acme Band"}, {"S": "Best Song"}]
 },
 {
  "Statement": "UPDATE Music SET AwardsWon=1, AwardDetail={'Grammys':[2020,
 2018]} WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
    }
]
```

2. Execute o comando a seguir em um prompt de comando.

```
aws dynamodb batch-execute-statement  --statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlBatch {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create BatchExecuteStatementRequest
            BatchExecuteStatementRequest batchExecuteStatementRequest =
 createBatchExecuteStatementRequest();
            BatchExecuteStatementResult batchExecuteStatementResult =
 dynamoDB.batchExecuteStatement(batchExecuteStatementRequest);
            System.out.println("BatchExecuteStatement successful.");
```

```
            // Handle batchExecuteStatementResult

        } catch (Exception e) {
            handleBatchExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {

        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static BatchExecuteStatementRequest createBatchExecuteStatementRequest()
{
        BatchExecuteStatementRequest request = new BatchExecuteStatementRequest();

        // Create statements
        List<BatchStatementRequest> statements = getPartiQLBatchStatements();

        request.setStatements(statements);
        return request;
    }

    private static List<BatchStatementRequest> getPartiQLBatchStatements() {
        List<BatchStatementRequest> statements = new
ArrayList<BatchStatementRequest>();

        statements.add(new BatchStatementRequest()
                               .withStatement("INSERT INTO Music value
{'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"));

        statements.add(new BatchStatementRequest()
                               .withStatement("UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist='Acme Band' and SongTitle='PartiQL
Rocks'"));

        return statements;
    }

    // Handles errors during BatchExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleBatchExecuteStatementErrors(Exception exception) {
        try {
```

```
            throw exception;
        } catch (Exception e) {
            // There are no API specific errors to handle for BatchExecuteStatement,
common DynamoDB API errors are handled below
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
retry strategy. Error: " + e.getMessage());
        }
```

```
    }

}
```

Políticas de segurança do IAM com PartiQL para DynamoDB

As seguintes permissões são necessárias:

- Para ler itens usando PartiQL para DynamoDB, é necessário ter a permissão `dynamodb:PartiQLSelect` na tabela ou no índice.
- Para inserir itens usando PartiQL para DynamoDB, é necessário ter a permissão `dynamodb:PartiQLInsert` na tabela ou no índice.
- Para atualizar itens usando PartiQL para DynamoDB, é necessário ter a permissão `dynamodb:PartiQLUpdate` na tabela ou no índice.
- Para excluir itens usando PartiQL para DynamoDB, é necessário ter a permissão `dynamodb:PartiQLDelete` na tabela ou no índice.

Exemplo: permitir todas as instruções PartiQL para DynamoDB (Select/Insert/Update/Delete) em uma tabela

A política do IAM a seguir concede permissões para executar todas as instruções PartiQL para DynamoDB em uma tabela.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
```

```
         ]
      }
   ]
}
```

Exemplo: permitir instruções Select PartiQL para DynamoDB em uma tabela

A política do IAM a seguir concede permissões para executar a instrução `select` em uma tabela específica.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Exemplo: permitir instruções Insert PartiQL para DynamoDB em um índice

A política do IAM a seguir concede permissões para executar a instrução `insert` em um índice específico.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Effect":"Allow",
```

```
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
      }
   ]
}
```

Exemplo: permitir instruções transacionais PartiQL para DynamoDB somente em uma tabela

A política do IAM a seguir concede permissões para executar somente instruções transacionais em uma tabela específica.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      }
   ]
}
```

Exemplo: permitir leituras e gravações não transacionais PartiQL para DynamoDB e bloquear leituras e gravações transacionais PartiQL em uma tabela.

A política do IAM a seguir concede permissões para executar leituras e gravações não transacionais PartiQL para DynamoDB ao mesmo tempo que bloqueia leituras e gravações transacionais PartiQL para DynamoDB.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
```

```
      }
   ]
}
```

Exemplo: permitir instruções Select e negar instruções de verificação de tabela completa em PartiQL para DynamoDB

A política do IAM a seguir concede permissões para executar a instrução `select` em uma tabela específica ao mesmo tempo que bloqueia instruções `select` que resultam em uma verificação de tabela completa.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ],
         "Condition":{
            "Bool":{
               "dynamodb:FullTableScan":[
                  "true"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ]
      }
```

```
    ]
}
```

Trabalhar com itens: Java

Você pode usar a API de documentos do AWS SDK para Java para realizar operações create, read, update e delete (CRUD) típicas nos itens do Amazon DynamoDB em uma tabela.

> **Note**
>
> O SDK for Java também fornece um modelo de persistência de objetos que permite que você mapeie suas classes do lado do cliente para tabelas do DynamoDB. Essa abordagem pode reduzir a quantidade de código que você precisa escrever. Para ter mais informações, consulte Java 1.x: DynamoDBMapper.

Esta seção contém exemplos de Java para executar várias ações de itens da API de documento do Java e vários exemplos funcionais completos.

Tópicos

- Colocar um item
- Obter um item
- Gravação em lote: colocar e excluir vários itens
- Obtenção em lote: obter vários itens
- Atualizar um item
- Excluir um item
- Exemplo: operações CRUD usando a API de documento do AWS SDK para Java
- Exemplo: operações em lote usando a API de documento do AWS SDK para Java
- Exemplo: tratar atributos do tipo binário usando a API de documento do AWS SDK para Java

Colocar um item

O método `putItem` armazena um item em uma tabela. Se o item existe, ele substitui o item inteiro. Em vez de substituir o item inteiro, se você quiser atualizar apenas atributos específicos, use o método `updateItem`. Para ter mais informações, consulte Atualizar um item.

Java v2

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval> <awards>
 <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is placed
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
```

```
                    SongTitleVal - The value of the song title (for example, Happy
Day).
                **Warning** This program will  place an item that you specify into a
table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {
```

```
        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle, AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The request
 id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Java v1

Siga estas etapas:

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `Table` para representar a tabela com a qual você deseja trabalhar.
3. Crie uma instância da classe `Item` para representar o novo item. Você deve especificar a chave primária do novo item e seus atributos.
4. Chame o método `putItem` do objeto `Table`, usando o `Item` que você criou na etapa anterior.

O exemplo de código Java a seguir demonstra as tarefas anteriores. O código grava um novo item na tabela `ProductCatalog`.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

// Build a list of related items
List<Number> relatedItems = new ArrayList<Number>();
relatedItems.add(341);
relatedItems.add(472);
relatedItems.add(649);

//Build a map of product pictures
Map<String, String> pictures = new HashMap<String, String>();
pictures.put("FrontView", "http://example.com/products/123_front.jpg");
pictures.put("RearView", "http://example.com/products/123_rear.jpg");
pictures.put("SideView", "http://example.com/products/123_left_side.jpg");

//Build a map of product reviews
Map<String, List<String>> reviews = new HashMap<String, List<String>>();

List<String> fiveStarReviews = new ArrayList<String>();
fiveStarReviews.add("Excellent! Can't recommend it highly enough!  Buy it!");
fiveStarReviews.add("Do yourself a favor and buy this");
reviews.put("FiveStar", fiveStarReviews);

List<String> oneStarReviews = new ArrayList<String>();
oneStarReviews.add("Terrible product!  Do not buy this.");
reviews.put("OneStar", oneStarReviews);

// Build the item
Item item = new Item()
    .withPrimaryKey("Id", 123)
    .withString("Title", "Bicycle 123")
    .withString("Description", "123 description")
    .withString("BicycleType", "Hybrid")
    .withString("Brand", "Brand-Company C")
    .withNumber("Price", 500)
    .withStringSet("Color",  new HashSet<String>(Arrays.asList("Red", "Black")))
    .withString("ProductCategory", "Bicycle")
```

```
    .withBoolean("InStock", true)
    .withNull("QuantityOnHand")
    .withList("RelatedItems", relatedItems)
    .withMap("Pictures", pictures)
    .withMap("Reviews", reviews);

// Write the item to the table
PutItemOutcome outcome = table.putItem(item);
```

No exemplo anterior, o item tem atributos que são escalares (`String`, `Number`, `Boolean`, `Null`), conjuntos (`String Set`) e tipos de documento (`List`, `Map`).

Especificar parâmetros opcionais

Além dos parâmetros obrigatórios, você também pode especificar parâmetros opcionais para o método `putItem`. Por exemplo, o seguinte exemplo de código Java usa um parâmetro opcional para especificar uma condição para fazer upload do item. Se a condição que você especificar não for atendida, o AWS SDK para Java lançará uma `ConditionalCheckFailedException`. O exemplo de código especifica os seguintes parâmetros opcionais no método `putItem`:

- Uma `ConditionExpression` que define as condições para a solicitação. O código define a condição de que o item existente com a mesma chave primária será substituído somente se tiver um atributo ISBN igual a um valor específico.
- Um mapa para `ExpressionAttributeValues` que é usado na condição. Nesse caso, existe apenas uma substituição obrigatória: o espaço reservado `:val` na expressão de condição é substituído em tempo de execução pelo valor de ISBN real a ser verificado.

O exemplo a seguir adiciona um novo item de livro usando esses parâmetros opcionais.

Example

```
Item item = new Item()
    .withPrimaryKey("Id", 104)
    .withString("Title", "Book 104 Title")
    .withString("ISBN", "444-4444444444")
    .withNumber("Price", 20)
    .withStringSet("Authors",
        new HashSet<String>(Arrays.asList("Author1", "Author2")));

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
```

```
expressionAttributeValues.put(":val", "444-4444444444");

PutItemOutcome outcome = table.putItem(
    item,
    "ISBN = :val", // ConditionExpression parameter
    null,          // ExpressionAttributeNames parameter - we're not using it for this
 example
    expressionAttributeValues);
```

PutItem e documentos JSON

Você pode armazenar um documento JSON como um atributo em uma tabela do DynamoDB. Para fazer isso, use o método `withJSON` de `Item`. Esse método analisa o documento JSON e mapeia cada elemento em um tipo de dados nativo do DynamoDB.

Suponha que você quisesse armazenar o seguinte documento JSON que contém os fornecedores que podem atender a pedidos de um determinado produto.

Example

```
{
    "V01": {
        "Name": "Acme Books",
        "Offices": [ "Seattle" ]
    },
    "V02": {
        "Name": "New Publishers, Inc.",
        "Offices": ["London", "New York"
        ]
    },
    "V03": {
        "Name": "Better Buy Books",
        "Offices": [ "Tokyo", "Los Angeles", "Sydney"
        ]
    }
}
```

Você pode usar o método `withJSON` para armazenar essas informações na tabela `ProductCatalog`, em um atributo `Map` chamado `VendorInfo`. O exemplo de código Java a seguir demonstra como fazer isso.

```
// Convert the document into a String.  Must escape all double-quotes.
```

```
String vendorDocument = "{"
    + "    \"V01\": {"
    + "        \"Name\": \"Acme Books\","
    + "        \"Offices\": [ \"Seattle\" ]"
    + "    },"
    + "    \"V02\": {"
    + "        \"Name\": \"New Publishers, Inc.\","
    + "        \"Offices\": [ \"London\", \"New York\"" + "]" + "},"
    + "    \"V03\": {"
    + "        \"Name\": \"Better Buy Books\","
    +          "\"Offices\": [ \"Tokyo\", \"Los Angeles\", \"Sydney\""
    + "            ]"
    + "        }"
    + "    }";

Item item = new Item()
    .withPrimaryKey("Id", 210)
    .withString("Title", "Book 210 Title")
    .withString("ISBN", "210-2102102102")
    .withNumber("Price", 30)
    .withJSON("VendorInfo", vendorDocument);

PutItemOutcome outcome = table.putItem(item);
```

Obter um item

Para recuperar um único item, use o método `getItem` de um objeto `Table`. Siga estas etapas:

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `Table` para representar a tabela com a qual você deseja trabalhar.
3. Chame o método `getItem` de instância de `Table`. Você deve especificar a chave primária do item que deseja recuperar.

O exemplo de código Java a seguir demonstra as etapas anteriores. O código obtém o item que tem a chave de partição especificada.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");
```

```
Item item = table.getItem("Id", 210);
```

Especificar parâmetros opcionais

Além dos parâmetros obrigatórios, você também pode especificar parâmetros opcionais do método `getItem`. Por exemplo, o seguinte exemplo de código Java usa um método opcional para recuperar somente uma lista específica de atributos e para especificar leituras fortemente consistentes. (Para saber mais sobre a consistência de leitura, consulte Consistência de leitura do DynamoDB.)

Você pode usar uma `ProjectionExpression` para recuperar somente atributos ou elementos específicos, em vez de todo o item. A `ProjectionExpression` pode especificar atributos aninhados ou de alto nível, usando caminhos de documentos. Para ter mais informações, consulte Usar expressões de projeção no DynamoDB.

Os parâmetros do método `getItem` não permitem que você especifique consistência de leitura. No entanto, você pode criar uma `GetItemSpec` que dá acesso total a todas as entradas para a operação `GetItem` de baixo nível. O exemplo de código a seguir cria uma `GetItemSpec` e usa essa especificação como entrada para o método `getItem`.

Example

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 206)
    .withProjectionExpression("Id, Title, RelatedItems[0], Reviews.FiveStar")
    .withConsistentRead(true);

Item item = table.getItem(spec);

System.out.println(item.toJSONPretty());
```

Para imprimir um `Item` em um formato legível, use o método `toJSONPretty`. A saída do exemplo anterior é semelhante à seguinte.

```
{
  "RelatedItems" : [ 341 ],
  "Reviews" : {
    "FiveStar" : [ "Excellent! Can't recommend it highly enough! Buy it!", "Do yourself
 a favor and buy this" ]
  },
  "Id" : 123,
```

```
  "Title" : "20-Bicycle 123"
}
```

GetItem e documentos JSON

Na seção PutItem e documentos JSON, você armazenou um documento JSON em um atributo `Map` chamado `VendorInfo`. Você pode usar o método `getItem` para recuperar o documento inteiro no formato JSON. Ou pode usar a notação do caminho do documento para recuperar apenas alguns dos elementos no documento. O exemplo de código Java a seguir demonstra essas técnicas.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210);

System.out.println("All vendor info:");
spec.withProjectionExpression("VendorInfo");
System.out.println(table.getItem(spec).toJSON());

System.out.println("A single vendor:");
spec.withProjectionExpression("VendorInfo.V03");
System.out.println(table.getItem(spec).toJSON());

System.out.println("First office location for this vendor:");
spec.withProjectionExpression("VendorInfo.V03.Offices[0]");
System.out.println(table.getItem(spec).toJSON());
```

A saída do exemplo anterior é semelhante à seguinte.

```
All vendor info:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]},"V02":{"Name":"New Publishers, Inc.","Offices":["London","New
 York"]},"V01":{"Name":"Acme Books","Offices":["Seattle"]}}}
A single vendor:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]}}}
First office location for a single vendor:
{"VendorInfo":{"V03":{"Offices":["Tokyo"]}}}
```

> **Note**
>
> Você pode usar o método `toJSON` para converter qualquer item (ou seus atributos) em uma string formatada para JSON. O exemplo de código a seguir recupera vários atributos aninhados e de alto nível e imprime os resultados como JSON.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210)
    .withProjectionExpression("VendorInfo.V01, Title, Price");

Item item = table.getItem(spec);
System.out.println(item.toJSON());
```

A saída é semelhante à seguinte.

```
{"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
```

Gravação em lote: colocar e excluir vários itens

Gravação em lote se refere a inserir e excluir vários itens em um lote. O método `batchWriteItem` permite que você insira e exclua vários itens de uma ou mais tabelas em uma única chamada. Veja a seguir as etapas para inserir ou excluir vários itens usando a API de documento do AWS SDK para Java.

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `TableWriteItems` que descreve todas as operações Put e Delete de uma tabela. Se você quiser gravar em várias tabelas com uma única operação de gravação em lote, crie uma instância de `TableWriteItems` por tabela.
3. Chame o método `batchWriteItem` fornecendo os objetos `TableWriteItems` que você criou na etapa anterior.
4. Processe a resposta. Você deve verificar se há itens de solicitação não processados retornados na resposta. Isso poderá acontecer se você atingir a cota de throughput provisionado ou por algum outro erro temporário. Além disso, o DynamoDB limita o tamanho da solicitação e o número de operações que você pode especificar em uma solicitação. Se você exceder esses

limites, o DynamoDB rejeitará a solicitação. Para ter mais informações, consulte Cotas no Amazon DynamoDB.

O exemplo de código Java a seguir demonstra as etapas anteriores. O exemplo executa uma operação `batchWriteItem` em duas tabelas: `Forum` e `Thread`. Os objetos `TableWriteItems` correspondentes definem as seguintes ações:

- Inserir um item na tabela `Forum`.
- Inserir e excluir um item na tabela `Thread`.

O código chama `batchWriteItem` para executar a operação.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableWriteItems forumTableWriteItems = new TableWriteItems("Forum")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("Name", "Amazon RDS")
            .withNumber("Threads", 0));

TableWriteItems threadTableWriteItems = new TableWriteItems("Thread")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("ForumName","Amazon RDS","Subject","Amazon RDS Thread 1")
    .withHashAndRangeKeysToDelete("ForumName","Some partition key value", "Amazon S3",
 "Some sort key value");

BatchWriteItemOutcome outcome = dynamoDB.batchWriteItem(forumTableWriteItems,
 threadTableWriteItems);

// Code for checking unprocessed items is omitted in this example
```

Para obter um exemplo funcional, consulte Exemplo: operação de gravação em lote usando a API de documento do AWS SDK para Java.

Obtenção em lote: obter vários itens

O método `batchGetItem` permite que você recupere vários itens de uma ou mais tabelas. Para recuperar um único item, você pode usar o método `getItem`.

Siga estas etapas:

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `TableKeysAndAttributes` que descreve uma lista de valores de chave primária para recuperar de uma tabela. Se você desejar ler em várias tabelas em uma única operação de aquisição em lote, será necessário criar uma instância de `TableKeysAndAttributes` por tabela.
3. Chame o método `batchGetItem` fornecendo os objetos `TableKeysAndAttributes` que você criou na etapa anterior.

O exemplo de código Java a seguir demonstra as etapas anteriores. O exemplo recupera dois itens da tabela `Forum` e três itens da tabela `Thread`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

    TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
    forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName", "Subject",
    "Amazon DynamoDB","DynamoDB Thread 1",
    "Amazon DynamoDB","DynamoDB Thread 2",
    "Amazon S3","S3 Thread 1");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(
    forumTableKeysAndAttributes, threadTableKeysAndAttributes);

for (String tableName : outcome.getTableItems().keySet()) {
    System.out.println("Items in table " + tableName);
    List<Item> items = outcome.getTableItems().get(tableName);
    for (Item item : items) {
        System.out.println(item);
    }
}
```

Especificar parâmetros opcionais

Além dos parâmetros obrigatórios, você também pode especificar parâmetros opcionais ao usar `batchGetItem`. Por exemplo, você pode fornecer uma `ProjectionExpression` com cada `TableKeysAndAttributes` que você definir. Isso permite que você especifique os atributos que deseja recuperar da tabela.

O exemplo de código a seguir recupera dois itens da tabela `Forum`. O parâmetro `withProjectionExpression` especifica que apenas o atributo `Threads` deve ser recuperado.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

Atualizar um item

O método `updateItem` de um objeto `Table` pode atualizar valores de atributo existentes, adicionar novos atributos ou excluir atributos de um item existente.

O método `updateItem` se comporta da seguinte forma:

- Se não existir um item (não houver nenhum item na tabela com a chave primária especificada), `updateItem` adicionará um novo item à tabela
- Se existir um item, `updateItem` executará a atualização conforme especificado pelo parâmetro `UpdateExpression`.

> **Note**
>
> Também é possível "atualizar" um item usando `putItem`. Por exemplo, se você chamar `putItem` para adicionar um item à tabela, mas já existir um item com a chave primária especificada, `putItem` substituirá o item inteiro. Se houver atributos no item existente que não são especificados na entrada, `putItem` removerá esses atributos do item.

> Em geral, recomendamos que você use `updateItem` sempre que desejar modificar quaisquer atributos de item. O método `updateItem` só modifica os atributos de item que você especifica na entrada, e os outros atributos no item permanecem inalterados.

Siga estas etapas:

1. Crie uma instância da classe `Table` para representar a tabela com a qual você deseja trabalhar.
2. Chame o método `updateTable` de instância de `Table`. Você deve especificar a chave primária do item que deseja recuperar, juntamente com uma `UpdateExpression` que descreve os atributos a serem modificados e como modificá-los.

O exemplo de código Java a seguir demonstra as tarefas anteriores. O código atualiza um item na tabela `ProductCatalog`. Ele adiciona um novo autor ao conjunto de `Authors` e exclui o atributo `ISBN` existente. Ele também reduz o preço por um.

Um mapa `ExpressionAttributeValues` é usado na `UpdateExpression`. Os espaços reservados `:val1` e `:val2` serão substituídos em tempo de execução pelos valores reais de `Authors` e `Price`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#A", "Authors");
expressionAttributeNames.put("#P", "Price");
expressionAttributeNames.put("#I", "ISBN");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1",
    new HashSet<String>(Arrays.asList("Author YY","Author ZZ")));
expressionAttributeValues.put(":val2", 1);   //Price

UpdateItemOutcome outcome =  table.updateItem(
    "Id",          // key attribute name
    101,           // key attribute value
    "add #A :val1 set #P = #P - :val2 remove #I", // UpdateExpression
    expressionAttributeNames,
```

```
    expressionAttributeValues);
```

Especificar parâmetros opcionais

Além dos parâmetros obrigatórios, você também pode especificar parâmetros opcionais para o método `updateItem`, incluindo uma condição que deve ser atendida para que a atualização ocorra. Se a condição que você especificar não for atendida, o AWS SDK para Java lançará uma `ConditionalCheckFailedException`. Por exemplo, o seguinte exemplo de código Java atualiza condicionalmente o preço de item de um livro para 25. Ele especifica uma `ConditionExpression` que declara que o preço deve ser atualizado somente se o preço existente for 20.

Example

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#P", "Price");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1", 25);  // update Price to 25...
expressionAttributeValues.put(":val2", 20);  //...but only if existing Price is 20

UpdateItemOutcome outcome = table.updateItem(
    new PrimaryKey("Id",101),
    "set #P = :val1", // UpdateExpression
    "#P = :val2",     // ConditionExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

Contador atômico

Você pode usar `updateItem` para implementar um contador atômico, onde pode aumentar ou reduzir o valor de um atributo existente sem interferir em outras solicitações de gravação. Para aumentar um contador atômico, use uma `UpdateExpression` com uma ação `set` para adicionar um valor numérico a um atributo existente do tipo `Number`.

O exemplo de código a seguir demonstra isso incrementando o atributo `Quantity` em um. Ele também demonstra o uso do parâmetro `ExpressionAttributeNames` em uma `UpdateExpression`.

```
Table table = dynamoDB.getTable("ProductCatalog");
```

```
Map<String,String> expressionAttributeNames = new HashMap<String,String>();
expressionAttributeNames.put("#p", "PageCount");

Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", 1);

UpdateItemOutcome outcome = table.updateItem(
    "Id", 121,
    "set #p = #p + :val",
    expressionAttributeNames,
    expressionAttributeValues);
```

Excluir um item

O método `deleteItem` exclui um item de uma tabela. É necessário fornecer a chave primária do item que você deseja excluir.

Siga estas etapas:

1. Crie uma instância do cliente `DynamoDB`.
2. Chame o método `deleteItem`, fornecendo a chave do item que você deseja excluir.

O exemplo de código Java a seguir demonstra essas tarefas.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

Especificar parâmetros opcionais

Você pode especificar parâmetros opcionais para `deleteItem`. Por exemplo, o seguinte exemplo de código Java especifica uma `ConditionExpression` que declara que um item de livro em `ProductCatalog` só pode ser excluído se o livro não estiver mais em publicação (o atributo `InPublication` é falso).

Example

```
Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", false);

DeleteItemOutcome outcome = table.deleteItem("Id",103,
    "InPublication = :val",
    null, // ExpressionAttributeNames - not used in this example
    expressionAttributeValues);
```

Exemplo: operações CRUD usando a API de documento do AWS SDK para Java

O exemplo de código a seguir ilustra operações CRUD em um item do Amazon DynamoDB. O exemplo cria um item, o recupera, executa várias atualizações e, por fim, o exclui.

> **Note**
>
> O SDK for Java também fornece um modelo de persistência de objetos que permite que você mapeie suas classes do lado do cliente para tabelas do DynamoDB. Essa abordagem pode reduzir a quantidade de código que você precisa escrever. Para ter mais informações, consulte Java 1.x: DynamoDBMapper.

> **Note**
>
> Este exemplo de código pressupõe que você já carregou dados no DynamoDB para sua conta seguindo as instruções na seção Criar tabelas e carregar dados para exemplos de código no DynamoDB.
> Para obter instruções passo a passo sobre como executar o exemplo a seguir, consulte Exemplos de código Java.

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
```

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DeleteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.UpdateItemOutcome;
import com.amazonaws.services.dynamodbv2.document.spec.DeleteItemSpec;
import com.amazonaws.services.dynamodbv2.document.spec.UpdateItemSpec;
import com.amazonaws.services.dynamodbv2.document.utils.NameMap;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.ReturnValue;

public class DocumentAPIItemCRUDExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ProductCatalog";

    public static void main(String[] args) throws IOException {

        createItems();

        retrieveItem();

        // Perform various updates.
        updateMultipleAttributes();
        updateAddNewAttribute();
        updateExistingAttributeConditionally();

        // Delete the item.
        deleteItem();

    }

    private static void createItems() {

        Table table = dynamoDB.getTable(tableName);
        try {

            Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book
 120 Title")
```

```
                    .withString("ISBN", "120-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author12", "Author22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", false).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 121).withString("Title", "Book 121
Title")
                    .withString("ISBN", "121-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author21", "Author 22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Create items failed.");
            System.err.println(e.getMessage());

        }
    }

    private static void retrieveItem() {
        Table table = dynamoDB.getTable(tableName);

        try {

            Item item = table.getItem("Id", 120, "Id, ISBN, Title, Authors", null);

            System.out.println("Printing item after retrieving it....");
            System.out.println(item.toJSONPretty());

        } catch (Exception e) {
            System.err.println("GetItem failed.");
            System.err.println(e.getMessage());
        }

    }
```

```
    private static void updateAddNewAttribute() {
        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
121)
                    .withUpdateExpression("set #na = :val1").withNameMap(new
NameMap().with("#na", "NewAttribute"))
                    .withValueMap(new ValueMap().withString(":val1", "Some value"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after adding new attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to add new attribute in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void updateMultipleAttributes() {

        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withUpdateExpression("add #a :val1 set #na=:val2")
                    .withNameMap(new NameMap().with("#a", "Authors").with("#na",
"NewAttribute"))
                    .withValueMap(
                            new ValueMap().withStringSet(":val1", "Author YY", "Author
ZZ").withString(":val2",
                                    "someValue"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
```

```
            System.out.println("Printing item after multiple attribute update...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to update multiple attributes in " + tableName);
            System.err.println(e.getMessage());

        }
    }

    private static void updateExistingAttributeConditionally() {

        Table table = dynamoDB.getTable(tableName);

        try {

            // Specify the desired price (25.00) and also the condition (price =
            // 20.00)

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withReturnValues(ReturnValue.ALL_NEW).withUpdateExpression("set #p
= :val1")
                    .withConditionExpression("#p = :val2").withNameMap(new
NameMap().with("#p", "Price"))
                    .withValueMap(new ValueMap().withNumber(":val1",
25).withNumber(":val2", 20));

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after conditional update to new
attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error updating item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void deleteItem() {

        Table table = dynamoDB.getTable(tableName);
```

```
        try {

            DeleteItemSpec deleteItemSpec = new DeleteItemSpec().withPrimaryKey("Id",
 120)
                    .withConditionExpression("#ip = :val").withNameMap(new
 NameMap().with("#ip", "InPublication"))
                    .withValueMap(new ValueMap().withBoolean(":val",
 false)).withReturnValues(ReturnValue.ALL_OLD);

            DeleteItemOutcome outcome = table.deleteItem(deleteItemSpec);

            // Check the response.
            System.out.println("Printing item that was deleted...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error deleting item in " + tableName);
            System.err.println(e.getMessage());
        }
    }
}
```

Exemplo: operações em lote usando a API de documento do AWS SDK para Java

Esta seção fornece exemplos de operações de gravação e aquisição em lote no Amazon DynamoDB usando a API de documentos do AWS SDK para Java.

> **Note**
>
> O SDK for Java também fornece um modelo de persistência de objetos que permite que você mapeie suas classes do lado do cliente para tabelas do DynamoDB. Essa abordagem pode reduzir a quantidade de código que você precisa escrever. Para ter mais informações, consulte Java 1.x: DynamoDBMapper.

Tópicos

- Exemplo: operação de gravação em lote usando a API de documento do AWS SDK para Java
- Exemplo: operação de obtenção em lote usando a API de documento do AWS SDK para Java

Exemplo: operação de gravação em lote usando a API de documento do AWS SDK para Java

O exemplo de código Java a seguir usa o método `batchWriteItem` para executar as operações Put e Delete a seguir:

- Inserir um item na tabela `Forum`.
- Inserir e excluir um item da tabela `Thread`.

Você pode especificar quantas solicitações de inserção e exclusão desejar em uma ou mais tabelas ao criar sua solicitação de gravação em lote. No entanto, `batchWriteItem` limita o tamanho de uma solicitação de gravação em lote e o número de operações Put e Delete em uma única operação. Se a sua solicitação ultrapassar esses limites, ela será rejeitada. Se a sua tabela não tiver throughput provisionado suficiente para servir a essa solicitação, os itens de solicitação não processados serão retornados na resposta.

O exemplo a seguir verifica a resposta para conferir se existem itens de solicitação não processados. Se houver, ele retorna e reenvia a solicitação `batchWriteItem` com itens não processados na solicitação. Se você tiver seguido os exemplos neste guia, já deverá ter criado as tabelas `Forum` e `Thread`. Você também pode criar essas tabelas e fazer upload dos dados de exemplo de forma programática. Para ter mais informações, consulte Criar exemplos de tabelas e carregar dados usando o AWS SDK para Java.

Para obter instruções detalhadas sobre como testar o exemplo a seguir, consulte Exemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchWriteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
```

```
import com.amazonaws.services.dynamodbv2.document.TableWriteItems;
import com.amazonaws.services.dynamodbv2.model.WriteRequest;

public class DocumentAPIBatchWrite {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {

        writeMultipleItemsBatchWrite();

    }

    private static void writeMultipleItemsBatchWrite() {
        try {

            // Add a new item to Forum
            TableWriteItems forumTableWriteItems = new
 TableWriteItems(forumTableName) // Forum
                    .withItemsToPut(new Item().withPrimaryKey("Name", "Amazon
 RDS").withNumber("Threads", 0));

            // Add a new item, and delete an existing item, from Thread
            // This table has a partition key and range key, so need to specify
            // both of them
            TableWriteItems threadTableWriteItems = new
 TableWriteItems(threadTableName)
                    .withItemsToPut(
                            new Item().withPrimaryKey("ForumName", "Amazon RDS",
 "Subject", "Amazon RDS Thread 1")
                                    .withString("Message", "ElastiCache Thread 1
 message")
                                    .withStringSet("Tags", new
 HashSet<String>(Arrays.asList("cache", "in-memory"))))
                    .withHashAndRangeKeysToDelete("ForumName", "Subject", "Amazon S3",
 "S3 Thread 100");

            System.out.println("Making the request.");
            BatchWriteItemOutcome outcome =
 dynamoDB.batchWriteItem(forumTableWriteItems, threadTableWriteItems);
```

```
            do {

                // Check for unprocessed keys which could happen if you exceed
                // provisioned throughput

                Map<String, List<WriteRequest>> unprocessedItems =
 outcome.getUnprocessedItems();

                if (outcome.getUnprocessedItems().size() == 0) {
                    System.out.println("No unprocessed items found");
                } else {
                    System.out.println("Retrieving the unprocessed items");
                    outcome = dynamoDB.batchWriteItemUnprocessed(unprocessedItems);
                }

            } while (outcome.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    }

}
```

Exemplo: operação de obtenção em lote usando a API de documento do AWS SDK para Java

O exemplo de código Java a seguir usa o método `batchGetItem` para recuperar vários itens das tabelas `Forum` e `Thread`. A `BatchGetItemRequest` especifica os nomes de tabela e uma lista de chaves de cada item a ser obtido. O exemplo processa a resposta, imprimindo os itens recuperados.

> **Note**
>
> Este exemplo de código pressupõe que você já carregou dados no DynamoDB para sua conta seguindo as instruções na seção Criar tabelas e carregar dados para exemplos de código no DynamoDB.
> Para obter instruções passo a passo sobre como executar o exemplo a seguir, consulte Exemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchGetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableKeysAndAttributes;
import com.amazonaws.services.dynamodbv2.model.KeysAndAttributes;

public class DocumentAPIBatchGet {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {
        retrieveMultipleItemsBatchGet();
    }

    private static void retrieveMultipleItemsBatchGet() {

        try {

            TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
            // Add a partition key
            forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name", "Amazon S3",
 "Amazon DynamoDB");

            TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
            // Add a partition key and a sort key
            threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName",
 "Subject", "Amazon DynamoDB",
```

```
                    "DynamoDB Thread 1", "Amazon DynamoDB", "DynamoDB Thread 2",
 "Amazon S3", "S3 Thread 1");

            System.out.println("Making the request.");

            BatchGetItemOutcome outcome =
 dynamoDB.batchGetItem(forumTableKeysAndAttributes,
                    threadTableKeysAndAttributes);

            Map<String, KeysAndAttributes> unprocessed = null;

            do {
                for (String tableName : outcome.getTableItems().keySet()) {
                    System.out.println("Items in table " + tableName);
                    List<Item> items = outcome.getTableItems().get(tableName);
                    for (Item item : items) {
                        System.out.println(item.toJSONPretty());
                    }
                }

                // Check for unprocessed keys which could happen if you exceed
                // provisioned
                // throughput or reach the limit on response size.
                unprocessed = outcome.getUnprocessedKeys();

                if (unprocessed.isEmpty()) {
                    System.out.println("No unprocessed keys found");
                } else {
                    System.out.println("Retrieving the unprocessed keys");
                    outcome = dynamoDB.batchGetItemUnprocessed(unprocessed);
                }

            } while (!unprocessed.isEmpty());

        } catch (Exception e) {
            System.err.println("Failed to retrieve items.");
            System.err.println(e.getMessage());
        }

    }

}
```

Exemplo: tratar atributos do tipo binário usando a API de documento do AWS SDK para Java

O exemplo de código Java a seguir ilustra o tratamento de atributos do tipo binário. O exemplo adiciona um item à tabela `Reply`. O item inclui um atributo do tipo binário (`ExtendedMessage`) que armazena dados compactados. Em seguida, o exemplo recupera um item e imprime todos os valores do atributo. Para ilustração, o exemplo usa a classe `GZIPOutputStream` para compactar um stream de exemplo e atribuí-lo ao atributo `ExtendedMessage`. Quando o atributo binário é recuperado, ele é descompactado usando a classe `GZIPInputStream`.

> **Note**
>
> O SDK for Java também fornece um modelo de persistência de objetos que permite que você mapeie suas classes do lado do cliente para tabelas do DynamoDB. Essa abordagem pode reduzir a quantidade de código que você precisa escrever. Para ter mais informações, consulte Java 1.x: DynamoDBMapper.

Se você tiver seguido a seção Criar tabelas e carregar dados para exemplos de código no DynamoDB, já deverá ter criado a tabela `Reply`. Você também pode criar essa tabela de forma programática. Para ter mais informações, consulte Criar exemplos de tabelas e carregar dados usando o AWS SDK para Java.

Para obter instruções detalhadas sobre como testar o exemplo a seguir, consulte Exemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;
```

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.GetItemSpec;

public class DocumentAPIItemBinaryExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";
    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    public static void main(String[] args) throws IOException {
        try {

            // Format the primary key values
            String threadId = "Amazon DynamoDB#DynamoDB Thread 2";

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
            String replyDateTime = dateFormatter.format(new Date());

            // Add a new reply with a binary attribute type
            createItem(threadId, replyDateTime);

            // Retrieve the reply with a binary attribute type
            retrieveItem(threadId, replyDateTime);

            // clean up by deleting the item
            deleteItem(threadId, replyDateTime);
        } catch (Exception e) {
            System.err.println("Error running the binary attribute type example: " +
 e);
            e.printStackTrace(System.err);
        }
    }

    public static void createItem(String threadId, String replyDateTime) throws
 IOException {
```

```
        Table table = dynamoDB.getTable(tableName);

        // Craft a long message
        String messageInput = "Long message to be compressed in a lengthy forum reply";

        // Compress the long message
        ByteBuffer compressedMessage = compressString(messageInput.toString());

        table.putItem(new Item().withPrimaryKey("Id",
threadId).withString("ReplyDateTime", replyDateTime)
                .withString("Message", "Long message
follows").withBinary("ExtendedMessage", compressedMessage)
                .withString("PostedBy", "User A"));
    }

    public static void retrieveItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        GetItemSpec spec = new GetItemSpec().withPrimaryKey("Id", threadId,
"ReplyDateTime", replyDateTime)
                .withConsistentRead(true);

        Item item = table.getItem(spec);

        // Uncompress the reply message and print
        String uncompressed =
uncompressString(ByteBuffer.wrap(item.getBinary("ExtendedMessage")));

        System.out.println("Reply message:\n" + " Id: " + item.getString("Id") + "\n" +
" ReplyDateTime: "
                + item.getString("ReplyDateTime") + "\n" + " PostedBy: " +
item.getString("PostedBy") + "\n"
                + " Message: "
                + item.getString("Message") + "\n" + " ExtendedMessage (uncompressed):
" + uncompressed + "\n");
    }

    public static void deleteItem(String threadId, String replyDateTime) {

        Table table = dynamoDB.getTable(tableName);
        table.deleteItem("Id", threadId, "ReplyDateTime", replyDateTime);
    }
```

```
    private static ByteBuffer compressString(String input) throws IOException {
        // Compress the UTF-8 encoded String into a byte[]
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPOutputStream os = new GZIPOutputStream(baos);
        os.write(input.getBytes("UTF-8"));
        os.close();
        baos.close();
        byte[] compressedBytes = baos.toByteArray();

        // The following code writes the compressed bytes to a ByteBuffer.
        // A simpler way to do this is by simply calling
        // ByteBuffer.wrap(compressedBytes);
        // However, the longer form below shows the importance of resetting the
        // position of the buffer
        // back to the beginning of the buffer if you are writing bytes directly
        // to it, since the SDK
        // will consider only the bytes after the current position when sending
        // data to DynamoDB.
        // Using the "wrap" method automatically resets the position to zero.
        ByteBuffer buffer = ByteBuffer.allocate(compressedBytes.length);
        buffer.put(compressedBytes, 0, compressedBytes.length);
        buffer.position(0); // Important: reset the position of the ByteBuffer
                            // to the beginning
        return buffer;
    }

    private static String uncompressString(ByteBuffer input) throws IOException {
        byte[] bytes = input.array();
        ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPInputStream is = new GZIPInputStream(bais);

        int chunkSize = 1024;
        byte[] buffer = new byte[chunkSize];
        int length = 0;
        while ((length = is.read(buffer, 0, chunkSize)) != -1) {
            baos.write(buffer, 0, length);
        }

        String result = new String(baos.toByteArray(), "UTF-8");

        is.close();
        baos.close();
```

```
        bais.close();

        return result;
    }
}
```

Trabalhar com itens: .NET

Você pode usar a API de baixo nível do AWS SDK para .NET para executar operações típicas create, read, update, delete (CRUD - criação, leitura, atualização e exclusão) em um item de uma tabela. Veja a seguir as etapas comuns que você segue para executar operações CRUD em dados usando a API de baixo nível do .NET:

1. Crie uma instância da classe `AmazonDynamoDBClient` (o cliente).
2. Forneça os parâmetros necessários específicos à operação em um objeto de solicitação correspondente.

   Por exemplo, use o objeto de solicitação `PutItemRequest` ao carregar um item e use o objeto de solicitação `GetItemRequest` ao recuperar um item existente.

   Você pode usar o objeto de solicitação para fornecer tanto parâmetros necessários quanto opcionais.
3. Execute o método apropriado fornecido pelo cliente, transmitindo o objeto de solicitação que você criou na etapa anterior.

   O cliente `AmazonDynamoDBClient` fornece os métodos `PutItem`, `GetItem`, `UpdateItem` e `DeleteItem` para as operações CRUD.

Tópicos

- Colocar um item
- Obter um item
- Atualizar um item
- Contador atômico
- Excluir um item
- Gravação em lote: colocar e excluir vários itens

- Obtenção em lote: obter vários itens
- Exemplo: operações CRUD usando a API de baixo nível AWS SDK para .NET
- Exemplo: operações em lote usando a API de baixo nível do AWS SDK para .NET
- Exemplo: tratar atributos do tipo binário usando a API de baixo nível do AWS SDK para .NET

Colocar um item

O método `PutItem` carrega um item em uma tabela. Se o item existe, ele substitui o item inteiro.

> **Note**
>
> Em vez de substituir o item inteiro, se você quiser atualizar apenas atributos específicos, use o método `UpdateItem`. Para ter mais informações, consulte Atualizar um item.

Veja a seguir as etapas para carregar um item usando a API do SDK do .NET de baixo nível:

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Forneça os parâmetros necessários, criando uma instância da classe `PutItemRequest`.

   Para inserir item, você deve fornecer o nome da tabela e o item.
3. Execute o método `PutItem` fornecendo o objeto `PutItemRequest` que você criou na etapa anterior.

O exemplo de C# a seguir demonstra as etapas anteriores. O exemplo faz upload de um item para a tabela `ProductCatalog`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new PutItemRequest
{
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
      {
```

```
          { "Id", new AttributeValue { N = "201" }},
          { "Title", new AttributeValue { S = "Book 201 Title" }},
          { "ISBN", new AttributeValue { S = "11-11-11-11" }},
          { "Price", new AttributeValue { S = "20.00" }},
          {
            "Authors",
            new AttributeValue
            { SS = new List<string>{"Author1", "Author2"}   }
          }
      }
};
client.PutItem(request);
```

No exemplo anterior, você faz upload de um item de livro que tem os atributos `Id`, `Title`, `ISBN` e `Authors`. Observe que `Id` é um atributo de tipo numérico, e todos os outros atributos são do tipo string. Autores é um conjunto `String`.

Especificar parâmetros opcionais

Você também pode fornecer parâmetros opcionais usando o objeto `PutItemRequest`, conforme mostrado no seguinte exemplo de C#. O exemplo especifica os seguintes parâmetros opcionais:

- `ExpressionAttributeNames`, `ExpressionAttributeValues` e `ConditionExpression` especificam que o item pode ser substituído somente se o item existente tiver o atributo ISBN com um valor específico.
- o parâmetro `ReturnValues` para solicitar o item antigo na resposta.

Example

```
var request = new PutItemRequest
 {
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
               {
                   { "Id", new AttributeValue { N = "104" }},
                   { "Title", new AttributeValue { S = "Book 104  Title" }},
                   { "ISBN", new AttributeValue { S = "444-4444444444" }},
                   { "Authors",
                     new AttributeValue { SS = new List<string>{"Author3"}}}
               },
    // Optional parameters.
```

```
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":isbn",new AttributeValue {S = "444-4444444444"}}
    },
    ConditionExpression = "#I = :isbn"

};
var response = client.PutItem(request);
```

Para obter mais informações, consulte PutItem.

Obter um item

O método `GetItem` recupera um item.

> **Note**
>
> Para recuperar vários itens, você pode usar o método `BatchGetItem`. Para ter mais informações, consulte Obtenção em lote: obter vários itens.

Veja a seguir as etapas para recuperar um item existente usando a API do AWS SDK para .NET de baixo nível.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Forneça os parâmetros necessários, criando uma instância da classe `GetItemRequest`.

   Para obter um item, você deve fornecer o nome da tabela e a chave primária desse item.
3. Execute o método `GetItem` fornecendo o objeto `GetItemRequest` que você criou na etapa anterior.

O exemplo de C# a seguir demonstra as etapas anteriores. O exemplo recupera um item da tabela `ProductCatalog`.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
 };
 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item; // Attribute list in the response.
```

Especificar parâmetros opcionais

Você também pode fornecer parâmetros opcionais usando o objeto `GetItemRequest`, conforme mostrado no seguinte exemplo de C#. O exemplo especifica os parâmetros opcionais a seguir:

- o parâmetro `ProjectionExpression` para especificar os atributos a serem recuperados.
- o parâmetro `ConsistentRead` para realizar uma leitura fortemente consistente. Para saber mais sobre a consistência de leitura, consulte Consistência de leitura do DynamoDB.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
   // Optional parameters.
   ProjectionExpression = "Id, ISBN, Title, Authors",
   ConsistentRead = true
 };

 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
```

```
var attributeMap = result.Item;
```

Para obter mais informações, consulte GetItem.

Atualizar um item

O método `UpdateItem` atualiza um item existente, se estiver presente. É possível usar a operação `UpdateItem` para atualizar valores de atributos existentes, adicionar novos atributos ou excluir atributos da coleção existente. Se o item que tem a chave primária especificada não for encontrado, ela adicionará um novo item.

A operação `UpdateItem` usa as seguintes diretrizes:

- Se o item não existir, o `UpdateItem` adicionará um novo item usando a chave primária especificada na entrada.
- Se o item existir, o `UpdateItem` aplicará as atualizações da seguinte maneira:
  - Substitui os valores de atributos existentes pelos valores na atualização.
  - Se o atributo que você fornecer na entrada não existir, ele adicionará um novo atributo ao item.
  - Se o valor do atributo de entrada for nulo, ele excluirá o atributo, se estiver presente.
  - Se você usar `ADD` para a `Action`, poderá adicionar valores a um conjunto existente (conjunto de strings ou números) ou adicionar (usar um número positivo) ou subtrair (usar um número negativo) matematicamente com base no valor de atributo numérico existente.

> **Note**
>
> A operação `PutItem` também pode realizar uma atualização. Para ter mais informações, consulte Colocar um item. Por exemplo, se você chamar `PutItem` para fazer upload de um item, e a chave primária existir, a operação `PutItem` substituirá o item inteiro. Se houver atributos no item existente e esses atributos não forem especificados na entrada, a operação `PutItem` os excluirá. No entanto, `UpdateItem` só atualiza os atributos de entrada especificados. Outros atributos existentes desse item permanecerão inalterados.

Veja a seguir as etapas para atualizar um item existente usando a API do SDK do .NET de baixo nível:

1. Crie uma instância da classe `AmazonDynamoDBClient`.

2. Forneça os parâmetros necessários, criando uma instância da classe `UpdateItemRequest`.

   Este é o objeto de solicitação em que você descreve todas as atualizações, por exemplo, adiciona atributos, atualizar atributos existentes ou excluir atributos. Para excluir um atributo existente, especifique o nome desse atributo com um valor nulo.
3. Execute o método `UpdateItem` fornecendo o objeto `UpdateItemRequest` que você criou na etapa anterior.

O exemplo de código C# a seguir demonstra as etapas anteriores. O exemplo de código atualiza um item na tabela `ProductCatalog`. Ele adiciona um novo autor à coleção `Authors` e exclui o atributo `ISBN` existente. Ele também reduz o preço por um.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#A", "Authors"},
        {"#P", "Price"},
        {"#NA", "NewAttribute"},
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":auth",new AttributeValue { SS = {"Author YY","Author ZZ"}}},
        {":p",new AttributeValue {N = "1"}},
        {":newattr",new AttributeValue {S = "someValue"}},
    },

    // This expression does the following:
    // 1) Adds two new authors to the list
    // 2) Reduces the price
    // 3) Adds a new attribute to the item
    // 4) Removes the ISBN attribute from the item
    UpdateExpression = "ADD #A :auth SET #P = #P - :p, #NA = :newattr REMOVE #I"
};
```

```
var response = client.UpdateItem(request);
```

Especificar parâmetros opcionais

Você também pode fornecer parâmetros opcionais usando o objeto `UpdateItemRequest`, conforme mostrado no seguinte exemplo de C#. Especifica os parâmetros opcionais a seguir:

- `ExpressionAttributeValues` e `ConditionExpression` para especificar que o preço apenas poderá ser atualizado se o preço existente for 20,00.
- o parâmetro `ReturnValues` para solicitar o item atualizado na resposta.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },

    // Update price only if the current price is 20.00.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#P", "Price"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":newprice",new AttributeValue {N = "22"}},
        {":currprice",new AttributeValue {N = "20"}}
    },
    UpdateExpression = "SET #P = :newprice",
    ConditionExpression = "#P = :currprice",
    TableName = tableName,
    ReturnValues = "ALL_NEW" // Return all the attributes of the updated item.
};

var response = client.UpdateItem(request);
```

Para obter mais informações, consulte UpdateItem.

Contador atômico

Você pode usar `updateItem` para implementar um contador atômico, onde pode aumentar ou reduzir o valor de um atributo existente sem interferir em outras solicitações de gravação. Para atualizar um contador atômico, use `updateItem` com um atributo do tipo `Number` no parâmetro `UpdateExpression` e `ADD` como a `Action`.

O exemplo de código a seguir demonstra isso incrementando o atributo `Quantity` em um.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string, AttributeValue>() { { "Id", new AttributeValue { N =
 "121" } } },
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#Q", "Quantity"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":incr",new AttributeValue {N = "1"}}
    },
    UpdateExpression = "SET #Q = #Q + :incr",
    TableName = tableName
};

var response = client.UpdateItem(request);
```

Excluir um item

O método `DeleteItem` exclui um item de uma tabela.

Veja a seguir as etapas para excluir um item usando a API de SDK do .NET de baixo nível.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Forneça os parâmetros necessários, criando uma instância da classe `DeleteItemRequest`.

   Para excluir um item, o nome da tabela e a chave primária desse item são necessários.
3. Execute o método `DeleteItem` fornecendo o objeto `DeleteItemRequest` que você criou na etapa anterior.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },
};

var response = client.DeleteItem(request);
```

Especificar parâmetros opcionais

Você também pode fornecer parâmetros opcionais usando o objeto `DeleteItemRequest`, conforme mostrado no seguinte exemplo de código C#. Especifica os parâmetros opcionais a seguir:

- `ExpressionAttributeValues` e `ConditionExpression` para especificar que o item de livro apenas poderá ser excluído se não estiver mais em publicação (se o valor do atributo InPublication for false).
- o parâmetro `ReturnValues` para solicitar o item excluído na resposta.

Example

```
var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },

    // Optional parameters.
    ReturnValues = "ALL_OLD",
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#IP", "InPublication"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":inpub",new AttributeValue {BOOL = false}}
```

```
    },
    ConditionExpression = "#IP = :inpub"
};

var response = client.DeleteItem(request);
```

Para obter mais informações, consulte DeleteItem.

Gravação em lote: colocar e excluir vários itens

Gravação em lote se refere a inserir e excluir vários itens em um lote. O método `BatchWriteItem` permite que você insira e exclua vários itens de uma ou mais tabelas em uma única chamada. Veja a seguir as etapas para recuperar vários itens usando a API de SDK do .NET de baixo nível.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Descreva todas as operações de inserção e exclusão, criando uma instância da classe `BatchWriteItemRequest`.
3. Execute o método `BatchWriteItem` fornecendo o objeto `BatchWriteItemRequest` que você criou na etapa anterior.
4. Processe a resposta. Você deve verificar se há itens de solicitação não processados retornados na resposta. Isso poderá acontecer se você atingir a cota de throughput provisionado ou por algum outro erro temporário. Além disso, o DynamoDB limita o tamanho da solicitação e o número de operações que você pode especificar em uma solicitação. Se você exceder esses limites, o DynamoDB rejeitará a solicitação. Para obter mais informações, consulte BatchWriteItem.

O exemplo de código C# a seguir demonstra as etapas anteriores. O exemplo cria uma `BatchWriteItemRequest` para realizar as seguintes operações de gravação:

- Inserir um item na tabela `Forum`.
- Inserir e excluir um item da tabela `Thread`.

Em seguida, o código executa `BatchWriteItem` para realizar uma operação em lote.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";
```

```
var request = new BatchWriteItemRequest
 {
   RequestItems = new Dictionary<string, List<WriteRequest>>
    {
      {
        table1Name, new List<WriteRequest>
        {
          new WriteRequest
          {
             PutRequest = new PutRequest
             {
                Item = new Dictionary<string,AttributeValue>
                {
                  { "Name", new AttributeValue { S = "Amazon S3 forum" } },
                  { "Threads", new AttributeValue { N = "0" }}
                }
             }
          }
        }
      } ,
      {
        table2Name, new List<WriteRequest>
        {
          new WriteRequest
          {
            PutRequest = new PutRequest
            {
               Item = new Dictionary<string,AttributeValue>
               {
                 { "ForumName", new AttributeValue { S = "Amazon S3 forum" } },
                 { "Subject", new AttributeValue { S = "My sample question" } },
                 { "Message", new AttributeValue { S = "Message Text." } },
                 { "KeywordTags", new AttributeValue { SS = new List<string> { "Amazon
 S3", "Bucket" }  } }
               }
            }
          },
          new WriteRequest
          {
             DeleteRequest = new DeleteRequest
             {
                Key = new Dictionary<string,AttributeValue>()
                {
```

```
                    { "ForumName", new AttributeValue { S = "Some forum name" } },
                    { "Subject", new AttributeValue { S = "Some subject" } }
                }
            }
        }
    }
  }
 };
response = client.BatchWriteItem(request);
```

Para obter um exemplo funcional, consulte Exemplo: operações em lote usando a API de baixo nível do AWS SDK para .NET.

Obtenção em lote: obter vários itens

O método `BatchGetItem` permite que você recupere vários itens de uma ou mais tabelas.

> **Note**
>
> Para recuperar um único item, você pode usar o método `GetItem`.

Veja a seguir as etapas para recuperar vários itens usando a API do AWS SDK para .NET de baixo nível.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Forneça os parâmetros necessários, criando uma instância da classe `BatchGetItemRequest`.

   Para recuperar vários itens, o nome da tabela e uma lista de valores de chave primária são necessários.
3. Execute o método `BatchGetItem` fornecendo o objeto `BatchGetItemRequest` que você criou na etapa anterior.
4. Processe a resposta. Você deve verificar se houve chaves não processadas, o que pode acontecer caso você tenha atingido a cota de throughput provisionado ou caso tenha ocorrido algum outro erro temporário.

O exemplo de código C# a seguir demonstra as etapas anteriores. O exemplo recupera itens de duas tabelas, `Forum` e `Thread`. A solicitação especifica dois itens na tabela `Forum` e três itens em

`Thread`. A resposta inclui itens de ambas as tabelas. O código mostra como você pode processar a resposta.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      }
    },
    {
      table2Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 1" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 2" } }
```

```
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "Amazon S3" } },
            { "Subject", new AttributeValue { S = "Amazon S3 Thread 1" } }
          }
        }
      }
    }
  }
};

var response = client.BatchGetItem(request);

// Check the response.
var result = response.BatchGetItemResult;
var responses = result.Responses; // The attribute list in the response.

var table1Results = responses[table1Name];
Console.WriteLine("Items in table {0}" + table1Name);
foreach (var item1 in table1Results.Items)
{
  PrintItem(item1);
}

var table2Results = responses[table2Name];
Console.WriteLine("Items in table {1}" + table2Name);
foreach (var item2 in table2Results.Items)
{
  PrintItem(item2);
}
// Any unprocessed keys? could happen if you exceed ProvisionedThroughput or some other
 error.
Dictionary<string, KeysAndAttributes> unprocessedKeys = result.UnprocessedKeys;
foreach (KeyValuePair<string, KeysAndAttributes> pair in unprocessedKeys)
{
    Console.WriteLine(pair.Key, pair.Value);
}
```

Especificar parâmetros opcionais

Você também pode fornecer parâmetros opcionais usando o objeto `BatchGetItemRequest`, conforme mostrado no seguinte exemplo de código C#. O exemplo recupera dois itens da tabela `Forum`. Ele especifica o parâmetro opcional a seguir:

- o parâmetro `ProjectionExpression` para especificar os atributos a serem recuperados.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      },
      // Optional - name of an attribute to retrieve.
      ProjectionExpression = "Title"
    }
  }
};

var response = client.BatchGetItem(request);
```

Para obter mais informações, consulte BatchGetItem.

Exemplo: operações CRUD usando a API de baixo nível AWS SDK para .NET

O exemplo de código C# a seguir ilustra operações CRUD em um item do Amazon DynamoDB. O exemplo adiciona um item à tabela `ProductCatalog`, recupera-o, executa várias atualizações e, por fim, exclui o item. Se você não criou essa tabela, também pode criá-la de maneira programática. Para ter mais informações, consulte Criar exemplos de tabelas e carregar dados usando o AWS SDK para .NET.

Para obter instruções detalhadas sobre como testar o exemplo a seguir, consulte Exemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelItemCRUDExample
    {
        private static string tableName = "ProductCatalog";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                CreateItem();
                RetrieveItem();

                // Perform various updates.
                UpdateMultipleAttributes();
                UpdateExistingAttributeConditionally();

                // Delete item.
                DeleteItem();
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void CreateItem()
        {
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  }},
                { "Title", new AttributeValue {
                      S = "Book 201 Title"
                  }},
                { "ISBN", new AttributeValue {
                      S = "11-11-11-11"
                  }},
                { "Authors", new AttributeValue {
                      SS = new List<string>{"Author1", "Author2" }
                  }},
                { "Price", new AttributeValue {
                      N = "20.00"
                  }},
                { "Dimensions", new AttributeValue {
                      S = "8.5x11.0x.75"
                  }},
                { "InPublication", new AttributeValue {
                      BOOL = false
                  } }
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem()
        {
            var request = new GetItemRequest
```

```
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ProjectionExpression = "Id, ISBN, Title, Authors",
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");
            PrintItem(attributeList);
        }

        private static void UpdateMultipleAttributes()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                // Perform the following updates:
                // 1) Add two new authors to the list
                // 1) Set a new attribute
                // 2) Remove the ISBN attribute
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#A","Authors"},
                {"#NA","NewAttribute"},
                {"#I","ISBN"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":auth",new AttributeValue {
                     SS = {"Author YY", "Author ZZ"}
                 }},
```

```
                {":new",new AttributeValue {
                     S = "New Value"
                 }}
            },

                UpdateExpression = "ADD #A :auth SET #NA = :new REMOVE #I",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
                                                     // print attributeList.
            Console.WriteLine("\nPrinting item after multiple attribute
update .............");
            PrintItem(attributeList);
        }

        private static void UpdateExistingAttributeConditionally()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#P", "Price"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":newprice",new AttributeValue {
                     N = "22.00"
                 }},
                {":currprice",new AttributeValue {
                     N = "20.00"
                 }}
            },
                // This updates price only if current price is 20.00.
```

```
                UpdateExpression = "SET #P = :newprice",
                ConditionExpression = "#P = :currprice",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
            Console.WriteLine("\nPrinting item after updating price value
conditionally ............");
            PrintItem(attributeList);
        }

        private static void DeleteItem()
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },

                // Return the entire item as it appeared before the update.
                ReturnValues = "ALL_OLD",
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#IP", "InPublication"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":inpub",new AttributeValue {
                     BOOL = false
                 }}
            },
                ConditionExpression = "#IP = :inpub"
            };

            var response = client.DeleteItem(request);
```

```
            // Check the response.
            var attributeList = response.Attributes; // Attribute list in the response.
                                                     // Print item.
            Console.WriteLine("\nPrinting item that was just deleted ............");
            PrintItem(attributeList);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

Exemplo: operações em lote usando a API de baixo nível do AWS SDK para .NET

Tópicos

- Exemplo: operação de gravação em lote usando a API de baixo nível do AWS SDK para .NET
- Exemplo: operação de obtenção em lote usando a API de baixo nível do AWS SDK para .NET

Esta seção fornece exemplos de operações em lote, gravação em lote e aquisição em lote aceitas pelo Amazon DynamoDB.

Exemplo: operação de gravação em lote usando a API de baixo nível do AWS SDK para .NET

O exemplo de código C# a seguir usa o método `BatchWriteItem` para executar as operações de inserção e exclusão a seguir:

- Inserir um item na tabela `Forum`.
- Inserir e excluir um item da tabela `Thread`.

Você pode especificar quantas solicitações de inserção e exclusão desejar em uma ou mais tabelas ao criar sua solicitação de gravação em lote. No entanto, `BatchWriteItem` do DynamoDB limita o tamanho de uma solicitação de gravação em lote e o número de operações Put e Delete em uma única operação. Para obter mais informações, consulte BatchWriteItem. Se a sua solicitação ultrapassar esses limites, ela será rejeitada. Se a sua tabela não tiver throughput provisionado suficiente para servir a essa solicitação, os itens de solicitação não processados serão retornados na resposta.

O exemplo a seguir verifica a resposta para conferir se existem itens de solicitação não processados. Se houver, ele retorna e reenvia a solicitação `BatchWriteItem` com itens não processados na solicitação. Você também pode criar essas tabelas de exemplo e carregar dados de exemplo de forma programática. Para ter mais informações, consulte Criar exemplos de tabelas e carregar dados usando o AWS SDK para .NET.

Para obter instruções detalhadas sobre como testar o exemplo a seguir, consulte Exemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchWrite
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                TestBatchWrite();
```

```
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void TestBatchWrite()
        {
            var request = new BatchWriteItemRequest
            {
                ReturnConsumedCapacity = "TOTAL",
                RequestItems = new Dictionary<string, List<WriteRequest>>
            {
                {
                    table1Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "Name", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Threads", new AttributeValue {
                                          N = "0"
                                      }}
                                }
                            }
                        }
                    }
                },
                {
                    table2Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
```

```
                                    { "ForumName", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "My sample question"
                                      } },
                                    { "Message", new AttributeValue {
                                          S = "Message Text."
                                      } },
                                    { "KeywordTags", new AttributeValue {
                                          SS = new List<string> { "S3", "Bucket" }
                                      } }
                                }
                            }
                        },
                        new WriteRequest
                        {
                            // For the operation to delete an item, if you provide a
primary key value
                            // that does not exist in the table, there is no error, it
is just a no-op.
                            DeleteRequest = new DeleteRequest
                            {
                                Key = new Dictionary<string, AttributeValue>()
                                {
                                    { "ForumName",  new AttributeValue {
                                          S = "Some partition key value"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "Some sort key value"
                                      } }
                                }
                            }
                        }
                    }
                }
            }
            };

            CallBatchWriteTillCompletion(request);
        }

        private static void CallBatchWriteTillCompletion(BatchWriteItemRequest request)
        {
```

```
            BatchWriteItemResponse response;

            int callCount = 0;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchWriteItem(request);
                callCount++;

                // Check the response.

                var tableConsumedCapacities = response.ConsumedCapacity;
                var unprocessed = response.UnprocessedItems;

                Console.WriteLine("Per-table consumed capacity");
                foreach (var tableConsumedCapacity in tableConsumedCapacities)
                {
                    Console.WriteLine("{0} - {1}", tableConsumedCapacity.TableName,
 tableConsumedCapacity.CapacityUnits);
                }

                Console.WriteLine("Unprocessed");
                foreach (var unp in unprocessed)
                {
                    Console.WriteLine("{0} - {1}", unp.Key, unp.Value.Count);
                }
                Console.WriteLine();

                // For the next iteration, the request will have unprocessed items.
                request.RequestItems = unprocessed;
            } while (response.UnprocessedItems.Count > 0);

            Console.WriteLine("Total # of batch write API calls made: {0}", callCount);
        }
    }
}
```

Exemplo: operação de obtenção em lote usando a API de baixo nível do AWS SDK para .NET

O exemplo de código C# a seguir usa o método `BatchGetItem` para recuperar vários itens das tabelas `Forum` e `Thread` no Amazon DynamoDB. A `BatchGetItemRequest` especifica os nomes de tabelas e uma lista de chaves primárias para cada tabela. O exemplo processa a resposta, imprimindo os itens recuperados.

Para obter instruções detalhadas sobre como testar o exemplo a seguir, consulte Exemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchGet
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                RetrieveMultipleItemsBatchGet();

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void RetrieveMultipleItemsBatchGet()
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
```

```
                    {
                        { "Name", new AttributeValue {
                      S = "Amazon DynamoDB"
                  } }
                    },
                    new Dictionary<string, AttributeValue>()
                    {
                        { "Name", new AttributeValue {
                      S = "Amazon S3"
                  } }
                    }
                }
            }},
        {
            table2Name,
            new KeysAndAttributes
            {
                Keys = new List<Dictionary<string, AttributeValue> >()
                {
                    new Dictionary<string, AttributeValue>()
                    {
                        { "ForumName", new AttributeValue {
                              S = "Amazon DynamoDB"
                          } },
                        { "Subject", new AttributeValue {
                              S = "DynamoDB Thread 1"
                          } }
                    },
                    new Dictionary<string, AttributeValue>()
                    {
                        { "ForumName", new AttributeValue {
                              S = "Amazon DynamoDB"
                          } },
                        { "Subject", new AttributeValue {
                              S = "DynamoDB Thread 2"
                          } }
                    },
                    new Dictionary<string, AttributeValue>()
                    {
                        { "ForumName", new AttributeValue {
                              S = "Amazon S3"
                          } },
                        { "Subject", new AttributeValue {
                              S = "S3 Thread 1"
```

```
                                } }
                            }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchGetItem(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
```

```
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

Exemplo: tratar atributos do tipo binário usando a API de baixo nível do AWS SDK para .NET

O exemplo de código C# a seguir ilustra a manipulação de atributos do tipo Binário. O exemplo adiciona um item à tabela `Reply`. O item inclui um atributo do tipo binário (`ExtendedMessage`) que armazena dados compactados. Em seguida, o exemplo recupera um item e imprime todos os valores do atributo. Para ilustração, o exemplo usa a classe `GZipStream` para compactar um stream de exemplo e designá-lo ao atributo `ExtendedMessage`. Em seguida, ele descompacta esse stream ao imprimir o valor do atributo.

Para obter instruções passo a passo sobre como testar o exemplo a seguir, consulte Exemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
```

```
using System.IO;
using System.IO.Compression;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelItemBinaryExample
    {
        private static string tableName = "Reply";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            // Reply table primary key.
            string replyIdPartitionKey = "Amazon DynamoDB#DynamoDB Thread 1";
            string replyDateTimeSortKey = Convert.ToString(DateTime.UtcNow);

            try
            {
                CreateItem(replyIdPartitionKey, replyDateTimeSortKey);
                RetrieveItem(replyIdPartitionKey, replyDateTimeSortKey);
                // Delete item.
                DeleteItem(replyIdPartitionKey, replyDateTimeSortKey);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateItem(string partitionKey, string sortKey)
        {
            MemoryStream compressedMessage = ToGzipMemoryStream("Some long extended
 message to compress.");
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
```

```
                }},
            { "ReplyDateTime", new AttributeValue {
                    S = sortKey
                }},
            { "Subject", new AttributeValue {
                    S = "Binary type "
                }},
            { "Message", new AttributeValue {
                    S = "Some message about the binary type"
                }},
            { "ExtendedMessage", new AttributeValue {
                    B = compressedMessage
                }}
        }
        };
        client.PutItem(request);
    }

    private static void RetrieveItem(string partitionKey, string sortKey)
    {
        var request = new GetItemRequest
        {
            TableName = tableName,
            Key = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                    S = partitionKey
                } },
            { "ReplyDateTime", new AttributeValue {
                    S = sortKey
                } }
        },
            ConsistentRead = true
        };
        var response = client.GetItem(request);

        // Check the response.
        var attributeList = response.Item; // attribute list in the response.
        Console.WriteLine("\nPrinting item after retrieving it ............");

        PrintItem(attributeList);
    }

    private static void DeleteItem(string partitionKey, string sortKey)
```

```
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            }
            };
            var response = client.DeleteItem(request);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]") +
                    (value.B == null ? "" : "B=[" + FromGzipMemoryStream(value.B) +
"]")
                    );
            }
            Console.WriteLine("*********************************************");
        }

        private static MemoryStream ToGzipMemoryStream(string value)
        {
            MemoryStream output = new MemoryStream();
            using (GZipStream zipStream = new GZipStream(output,
CompressionMode.Compress, true))
```

```
            using (StreamWriter writer = new StreamWriter(zipStream))
            {
                writer.Write(value);
            }
            return output;
        }

        private static string FromGzipMemoryStream(MemoryStream stream)
        {
            using (GZipStream zipStream = new GZipStream(stream,
 CompressionMode.Decompress))
            using (StreamReader reader = new StreamReader(zipStream))
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

Melhorar o acesso aos dados com índices secundários no DynamoDB

O Amazon DynamoDB dá acesso rápido a itens em uma tabela ao especificar valores de chave primária. No entanto, muitos aplicativos podem se beneficiar por terem uma ou mais chaves secundárias (ou alternativas) disponíveis, para permitir acesso eficiente aos dados com atributos que não sejam a chave primária. Para resolver isso, você pode criar um ou mais índices secundários em uma tabela e emitir solicitações de `Query` ou de `Scan` nesses índices.

Um índice secundário é uma estrutura de dados que contém um subconjunto de atributos de uma tabela, juntamente com uma chave alternativa para oferecer suporte a operações `Query`. Você pode recuperar dados do índice usando uma `Query`, de maneira muito semelhante ao uso de uma `Query` com uma tabela. Uma tabela pode ter vários índices secundários, o que permite que seus aplicativos tenham acesso a muitos padrões de consulta diferentes.

> **Note**
>
> Você também pode `Scan` um índice, da mesma forma como poderia `Scan` uma tabela. Atualmente, o acesso entre contas para operações de verificação de índice secundário não é aceito em políticas baseadas em recursos.

Cada índice secundário está associado a exatamente uma tabela, da qual ele obtém seus dados. Ela é chamada de tabela base para o índice. Ao criar um índice, você define uma chave alternativa para ele (chave de partição e chave de classificação). Você também define os atributos a serem projetados, ou copiados, da tabela-base para o índice. O DynamoDB copia esses atributos para o índice, juntamente com os atributos de chave primária da tabela-base. Você pode consultar ou verificar o índice como faria com qualquer tabela.

Cada índice secundário é automaticamente mantido pelo DynamoDB. Quando você adiciona, modifica ou exclui itens na tabela base, todos os índices dessa tabela também são atualizados para refletir essas alterações.

O DynamoDB oferece suporte a dois tipos de índices secundários:

- Índice secundário global: um índice com uma chave de partição e uma chave de classificação que podem ser diferentes daquelas contidas na tabela-base. Um índice secundário global é considerado “global” porque as consultas no índice podem abranger todos os dados em uma tabela base, além de todas as partições. O índice secundário global é armazenado em seu próprio espaço de partição longe da tabela-base e é dimensionado separadamente da tabela-base.
- Índice secundário local: um índice que tem a mesma chave de partição que a tabela-base, mas uma chave de classificação diferente. Um índice secundário local é “local” no sentido de que cada partição de um índice secundário local tem a determinação de escopo para uma partição de tabela base com o mesmo valor de chave de partição.

Para ver uma comparação entre índices secundários globais e índices secundários locais, assista a este vídeo.

Faça a escolha certa entre GSI e LSI

Tópicos

- Como usar índices secundários globais no DynamoDB
- Índices secundários locais no DynamoDB

Você deve considerar os requisitos do seu aplicativo ao determinar qual tipo de índice usar. A tabela a seguir mostra as principais diferenças entre um índice secundário global e um índice secundário local.

Característica	Índice secundário global	Índice secundário local
Esquema de chaves	A chave primária de um índice secundário global pode ser simples (chave de partição) ou composta (chave de partição e chave de classificação).	A chave primária de um índice secundário local deve ser composta (chave de partição e chave de classificação).
Atributos de chaves	A chave de partição e a chave de classificação (se presente) do índice podem ser qualquer atributo da tabela base do tipo String, Número ou Binário.	A chave de partição do índice é o mesmo atributo que a chave de partição da tabela base. A chave de classificação pode ser qualquer atributo da tabela base do tipo String, Número ou Binário.
Restrições de tamanho por valor de chave de partição	Não há restrições de tamanho para índices secundários globais.	Para cada valor de chave de partição, o tamanho total de todos os itens indexados deve ser 10 GB ou menos.
Operações de índice online	Você pode criar índices secundários globais ao mesmo tempo que cria uma tabela. Você também pode adicionar um novo índice secundário global a uma tabela ou então excluí-lo. Para obter mais informações, consulte Gerenciar índices secundários globais no DynamoDB.	Você pode criar índices secundários locais ao mesmo tempo que cria uma tabela. Não é possível adicionar um índice secundário local a uma tabela existente nem excluir índices secundários existente s.
Consultas e partições	Um índice secundário global permite que você consulte a tabela inteira, em todas as partições.	Um índice secundário local permite consultar uma única partição, conforme especific

Característica	Índice secundário global	Índice secundário local
		ado pelo valor da chave de partição na consulta.
Consistência de leituras	As consultas em índices secundários globais oferecem suporte somente à consistência final.	Ao consultar um índice secundário local, você pode escolher consistência final ou coerência forte.
Consumo de throughput provisionado	Cada índice secundário global tem suas próprias configurações de throughput provisionado para atividades de leitura e gravação. Consultas ou verificações em um índice secundário global consumem unidades de capacidade do índice, e não da tabela-base. O mesmo vale para atualizações de índices secundários globais devido a gravações de tabelas. Um índice secundário global associado a tabelas globais consome unidades de capacidade de gravação.	Consultas ou verificações em um índice secundário local consomem unidades de capacidade de leitura da tabela-base. Quando você grava em uma tabela, seus índices secundários locais também são atualizados. Essas atualizações consomem unidades de capacidade de gravação da tabela-base. Um índice secundário local associado a tabelas globais consome unidades de capacidade de gravação replicadas.
Atributos projetados	Com consultas ou verificações em um índice secundário global, você pode solicitar somente os atributos que estão projetados no índice. O DynamoDB não busca atributos da tabela.	Se você consultar ou verificar um índice secundário local, poderá solicitar atributos que não estão projetados no índice. O DynamoDB buscará automaticamente os atributos da tabela.

Se quiser criar mais de uma tabela com índices secundários, você deverá fazer isso sequencialmente. Por exemplo, você poderia criar a primeira tabela e esperar até que ela entrasse

no estado `ACTIVE`. Em seguida, você criaria a seguinte tabela e esperaria até que ela entrasse no estado `ACTIVE` e assim por diante. Se você tentar criar mais de uma tabela ao mesmo tempo com um índice secundário, o DynamoDB retornará uma `LimitExceededException`.

Cada índice secundário usa a mesma classe de tabela e modo de capacidade da tabela-base à qual está associado. Para cada índice secundário, é necessário especificar o seguinte:

- O tipo de índice a ser criado: índice secundário global ou índice secundário local.
- Um nome para o índice. As regras de nomenclatura de índices são iguais às de tabelas, conforme listado em Cotas no Amazon DynamoDB. O nome deve ser exclusivo para a tabela-base à qual ele está associado, mas você pode usar o mesmo nome para índices que estão associados a diferentes tabelas-base.
- O esquema de chaves do índice. Cada atributo no esquema de chaves do índice deve ser um atributo de nível superior do tipo `String`. `Number` ou `Binary`. Outros tipos de dados, incluindo documentos e conjuntos, não são permitidos. Outros requisitos para o esquema de chaves dependem do tipo de índice:
  - Para um índice secundário global, a chave de partição pode ser qualquer atributo escalar da tabela-base. Uma chave de classificação é opcional e também pode ser qualquer atributo escalar da tabela-base.
  - Para um índice secundário local, a chave de partição deve igual à chave de partição da tabela-base, e a chave de classificação deve ser um atributo da tabela-base não relacionado a chaves.
- Atributos adicionais, se houver, a serem projetados da tabela-base no índice. Esses atributos são adicionais aos atributos de chave da tabela, que são automaticamente projetados em cada índice. Você pode projetar atributos de qualquer tipo de dados, incluindo escalares, documentos e conjuntos.
- As configurações de throughput provisionado para o índice, se necessário:
  - Para um índice secundário global, você deve especificar configurações de unidades de capacidade de gravação. Essas configurações de throughput provisionado são independentes das configurações da tabela-base.
  - Para um índice secundário local, não é necessário especificar configurações de unidades de capacidade de gravação. Qualquer operação de leitura e gravação em um índice secundário extrai das configurações de throughput provisionado de sua tabela-base.

Para obter a flexibilidade máxima nas consultas, é possível criar até 20 índices secundários globais (cota padrão) e até 5 índices secundários locais por tabela.

Para obter uma listagem detalhada de índices secundários em uma tabela, use a operação `DescribeTable`. `DescribeTable` retorna o nome, o tamanho de armazenamento e as contagens de itens de cada índice secundário na tabela. Esses valores não são atualizados em tempo real, mas aproximadamente a cada seis horas.

Você pode acessar os dados em um índice secundário usando a operação `Query` ou `Scan`. Você deve especificar o nome da tabela-base e o nome do índice que deseja usar, os atributos a serem retornados nos resultados e qualquer expressão de condição a ser aplicada. O DynamoDB pode retornar os resultados em ordem crescente ou decrescente.

Quando uma tabela é excluída, todos os índices associados a ela também são excluídos.

Para ver as práticas recomendadas, consulte Práticas recomendadas para uso de índices secundários no DynamoDB.

Como usar índices secundários globais no DynamoDB

Alguns aplicativos talvez precisem executar muitos tipos de consultas, usando uma variedade de atributos diferentes como critérios de consulta. Para oferecer suporte a esses requisitos, você pode criar um ou mais índices secundários globais e emitir solicitações `Query` para esses índices no Amazon DynamoDB.

Tópicos

- Cenário: Uso de um índice secundário global
- Projeções de atributo
- Ler dados de um índice secundário global
- Sincronização de dados entre tabelas e índices secundários globais
- Classes de tabela com índice secundário global
- Considerações sobre throughput provisionado para índices secundários globais
- Considerações sobre armazenamento para índices secundários globais
- Gerenciar índices secundários globais no DynamoDB
- Detectar e corrigir violações de chave de índice no DynamoDB
- Como trabalhar com índices secundários globais: Java
- Como trabalhar com índices secundários globais: .NET
- Trabalhar com índices secundários globais no DynamoDB usando a AWS CLI

Cenário: Uso de um índice secundário global

Para ilustrar, considere uma tabela chamada `GameScores` que controla os usuários e os placares de um aplicativo de jogo móvel. Cada item em `GameScores` é identificado por uma chave de partição (`UserId`) e por uma chave de classificação (`GameTitle`). O diagrama a seguir mostra como os itens da tabela seriam organizados. (Nem todos os atributos são mostrados.)

GameScores

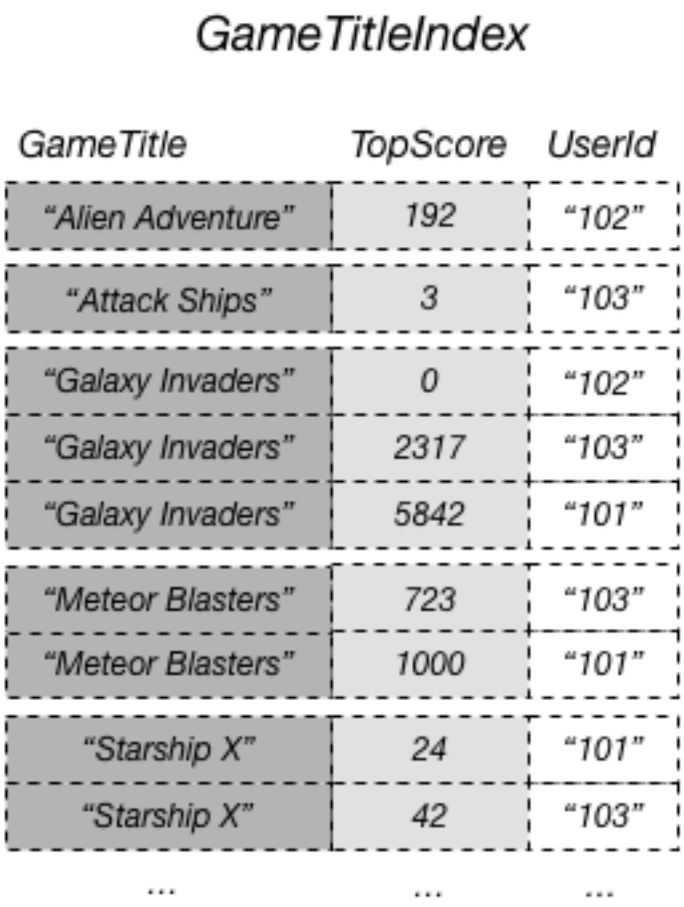
UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	

Agora, vamos supor que você quisesse gravar um aplicativo de placar para exibir as pontuações máximas de cada jogo. Uma consulta que especificasse os atributos chave (`UserId` e `GameTitle`) seria muito eficaz. No entanto, se o aplicativo precisasse recuperar dados de `GameScores` com base no `GameTitle` apenas, ele precisaria usar uma operação `Scan`. À medida que mais itens são adicionados à tabela, as verificações de todos os dados se tornam lentas e ineficientes. Isso dificulta responder a questões como as seguintes:

- Qual é a pontuação máxima já registrada no jogo Meteor Blasters?
- Qual usuário tinha a maior pontuação no Galaxy Invaders?
- Qual era a maior proporção de vitórias versus derrotas?

Para acelerar as consultas em atributos que não são chave, você pode criar um índice secundário global. Um índice secundário global contém uma seleção de atributos da tabela-base, mas eles são organizados por uma chave primária que é diferente daquela da região da tabela. A chave do índice não precisa ter nenhum dos atributos de chaves da tabela. Ela não precisa nem ter o mesmo esquema de chaves que a tabela.

Por exemplo, você poderia criar um índice secundário global chamado `GameTitleIndex` com uma chave de partição `GameTitle` e uma chave de classificação `TopScore`. Os atributos de chave primária da tabela base são sempre projetados em um índice, portanto, o atributo `UserId` também está presente. O diagrama a seguir mostra qual é a aparência do índice `GameTitleIndex`.



Agora você pode consultar `GameTitleIndex` e obter as pontuações do jogo Meteor Blasters com facilidade. Os resultados são ordenados por valores de chave de classificação, `TopScore`. Se você definir o parâmetro `ScanIndexForward` como falso, os resultados serão retornados em ordem decrescente, de modo que a maior pontuação é retornada primeiro.

Cada índice secundário global deve ter uma chave de partição e pode ter uma chave de classificação opcional. O esquema da chave de índice pode ser diferente do esquema da tabela

base. Você poderia ter uma tabela com uma chave primária simples (chave de partição) e criar um índice secundário global com uma chave primária composta (chave de partição e chave de classificação), ou vice versa. Os atributos de chaves do índice podem consistir em quaisquer atributos de nível superior, `String`, `Number` ou `Binary` da tabela base. Outros tipos escalares, tipos de documento e tipos de conjunto não são permitidos.

Você pode projetar outros atributos da tabela-base para o índice, se quisesse. Quando você consultar o índice, o DynamoDB poderá recuperar esses atributos projetados com eficiência. No entanto, as consultas do índice secundário global buscam atributos da tabela-base. Por exemplo, se você consultar `GameTitleIndex` conforme mostrado no diagrama anterior, a consulta poderá não acessar nenhum atributo que não seja de chave além de `TopScore` (embora os atributos de chave `GameTitle` e `UserId` sejam projetados automaticamente).

Em uma tabela do DynamoDB, cada valor de chave deve ser exclusivo. No entanto, os valores de chave em um índice secundário global não precisam ser exclusivos. Para ilustrar, suponhamos que um jogo chamado Comet Quest seja muito difícil. Há vários novos usuários tentando obter uma pontuação acima de zero, mas não conseguem. Veja a seguir alguns dos dados que podem representar isso.

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

Quando esses dados são adicionados à tabela `GameScores`, o DynamoDB os propaga para o `GameTitleIndex`. Se consultarmos o índice usando Comet Quest para `GameTitle` e 0 para `TopScore`, os seguintes dados serão retornados.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Apenas os itens com os valores de chaves especificados aparecem na resposta. Nesse conjunto de dados, os itens não estão em nenhuma ordem específica.

Um índice secundário global controla apenas os itens de dados em que seus atributos de chave realmente existem. Por exemplo, suponha que você tenha adicionado um novo item à tabela `GameScores`, mas forneceu somente os atributos de chave primária necessários.

UserId	GameTitle
400	Comet Quest

Como você não especificou o atributo `TopScore`, o DynamoDB não propagará esse item para `GameTitleIndex`. Portanto, se você tivesse consultado `GameScores` para obter todos os itens do Comet Quest, obteria os quatro itens a seguir.

UserId	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

Uma consulta semelhante em `GameTitleIndex` ainda retornaria três itens, em vez de quatro. Isso acontece porque o item com `TopScore` inexistente não é propagado para o índice.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Projeções de atributo

Uma projeção é o conjunto de atributos que é copiado de uma tabela para um índice secundário. A chave de partição e a chave de classificação da tabela são sempre projetadas no índice; você pode projetar outros atributos para suportar os requisitos de consulta da sua aplicação. Quando você

consulta um índice, o Amazon DynamoDB pode acessar quaisquer atributos na projeção como se estivessem em uma tabela própria.

Quando você cria um índice secundário, é necessário especificar os atributos que serão projetados no índice. O DynamoDB proporciona três opções diferentes para fazer isso:

- KEYS_ONLY: cada item do índice consiste apenas nos valores de chaves de partição e nas chaves de classificação da tabela, além dos valores de chaves do índice. A opção `KEYS_ONLY` resulta no menor índice secundário possível.
- INCLUDE: além dos atributos descritos em `KEYS_ONLY`, o índice secundário incluirá outros atributos não chave que você especificar.
- ALL: o índice secundário inclui todos os atributos da tabela de origem. Como todos os dados da tabela são duplicados no índice, uma projeção `ALL` resulta no maior índice secundário possível.

No diagrama anterior, `GameTitleIndex` tem apenas um atributo projetado: `UserId` Portanto, embora um aplicativo possa determinar eficientemente o `UserId` dos melhores marcadores para cada jogo usando `GameTitle` e `TopScore` nas consultas, ele não pode determinar com eficiência a maior proporção de vitórias versus derrotas dos maiores marcadores. Para fazer isso, ele teria que realizar uma consulta adicional na tabela base para buscar os ganhos e perdas de cada um dos maiores artilheiros. A maneira mais eficiente de oferecer suporte a consultas nesses dados seria projetar esses atributos da tabela-base no índice secundário global, conforme mostrado neste diagrama.

GameTitleIndex

GameTitle	TopScore	UserId	Wins	Losses
"Alien Adventure"	192	"102"	32	192
"Attack Ships"	3	"103"	1	8
"Galaxy Invaders"	0	"102"	0	5
"Galaxy Invaders"	2317	"103"	40	3
"Galaxy Invaders"	5842	"101"	21	72
"Meteor Blasters"	723	"103"	22	12
"Meteor Blasters"	1000	"101"	12	3
"Starship X"	24	"101"	4	9
"Starship X"	42	"103"	4	19
...	...	...	...	...

Como os atributos não são de chave `Wins` e `Losses` são projetados no índice, um aplicativo pode determinar a proporção de vitórias versus derrotas de qualquer jogo ou de qualquer combinação de jogo e ID do usuário.

Ao escolher os atributos para projetar em um índice secundário global, você deve considerar a desvantagem entre os custos de throughput provisionado e os custos de armazenamento:

- Se você precisar acessar apenas alguns atributos com a latência mais baixa possível, considere projetar apenas os atributos em um índice secundário global. Quanto menor o índice, menores serão os custos de armazenamento e de gravação.
- Se sua aplicação acessar frequentemente alguns atributos não chave, considere projetar esses atributos em um índice secundário global. Os custos adicionais de armazenamento do índice secundário global compensarão o custo de executar verificações de tabelas frequentes.
- Se precisar acessar a maioria dos atributos não chave com frequência, você poderá projetar esses atributos, inclusive a tabela-base inteira, em um índice secundário global. Isso dá flexibilidade máxima a você. No entanto, o custo do armazenamento aumentará ou até dobrará.

- Se o aplicativo precisar consultar uma tabela com pouca frequência, mas precisar executar muitas gravações ou atualizações nos dados da tabela, considere projetar `KEYS_ONLY`. O índice secundário global seria de tamanho mínimo, mas ainda estaria disponível quando necessário para a atividade de consulta.

Ler dados de um índice secundário global

Você pode recuperar itens de um índice secundário global usando as operações `Query` e `Scan`. As operações `GetItem` e `BatchGetItem` não podem ser usadas em um índice secundário global.

Como consultar um índice secundário global

Você pode usar a operação `Query` para acessar um ou mais itens em um índice secundário global. A consulta deve especificar o nome da tabela-base e o nome do índice que você deseja usar, os atributos a serem retornados nos resultados de consulta e quaisquer condições que você deseja aplicar. O DynamoDB pode retornar os resultados em ordem crescente ou decrescente.

Considere os seguintes dados retornados de uma `Query` que solicita dados de jogos para um aplicativo de placar.

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

Nesta consulta:

- O DynamoDB acessa GameTitleIndex usando a chave de partição GameTitle para localizar os itens de índice do Meteor Blasters. Todos os itens do índice com essa chave são armazenados lado a lado para rápida recuperação.
- Nesse jogo, o DynamoDB usa o índice para acessar todos os IDs de usuário e as pontuações máximas.

- Os resultados são retornados, classificados em ordem decrescente, pois o parâmetro `ScanIndexForward` está definido como falso.

Como verificar um índice secundário global

Você pode usar a operação `Scan` para recuperar todos os dados de um índice secundário global. Você deve fornecer o nome da tabela-base e o nome de índice na solicitação. Com uma operação `Scan`, o DynamoDB lê todos os dados do índice e os retorna para a aplicação. Você também pode solicitar que apenas alguns dos dados sejam retornados, e que os dados restantes sejam descartados. Para fazer isso, use o parâmetro `FilterExpression` da operação `Scan`. Para obter mais informações, consulte Expressões de filtro para verificação.

Sincronização de dados entre tabelas e índices secundários globais

O DynamoDB sincroniza automaticamente cada índice secundário global com sua tabela-base. Quando uma aplicação grava ou exclui itens em uma tabela, quaisquer índices secundários globais nessa tabela são atualizados de forma assíncrona usando um modelo final consistente. Os aplicativos nunca gravam diretamente em um índice. No entanto, é importante compreender as implicações de como o DynamoDB mantém esses índices.

Índices secundários globais herdam o modo de capacidade leitura/gravação da tabela base. Para obter mais informações, consulte Considerações ao alternar os modos de capacidade no DynamoDB.

Ao criar um índice secundário global, você especifica um ou mais atributos de chave de índice e seus tipos de dados. Isso significa que sempre que você grava um item na tabela-base, os tipos de dados desses atributos devem corresponder aos tipos de dados do esquema de chaves do índice. No caso de `GameTitleIndex`, a chave de partição `GameTitle` no índice é definida como um tipo de dados `String`. A chave de classificação `TopScore` no índice é do tipo `Number`. Se você tentar adicionar um item à tabela `GameScores` e especificar um tipo de dados diferente para `GameTitle` ou `TopScore`, o DynamoDB retornará uma `ValidationException` devido à inconsistência do tipo de dados.

Quando você insere ou exclui itens em uma tabela, os índices secundários globais dessa tabela são atualizados de uma forma eventualmente consistente. As alterações na tabela são propagadas para os índices secundários globais em uma fração de segundo, sob condições normais. No entanto, em alguns cenários improváveis de falha, podem ocorrer atrasos de propagação mais longos. Consequentemente, as aplicações precisam prever e lidar com situações em que uma consulta em um índice secundário global retorna resultados que não estão atualizados.

Se você gravar um item em uma tabela, não será necessário especificar os atributos para qualquer chave de classificação do índice secundário global. Usando `GameTitleIndex` como um exemplo, você não precisa especificar um valor para o atributo `TopScore` para gravar um novo item na tabela `GameScores`. Neste caso, o DynamoDB não grava todos os dados no índice deste item específico.

Os custos das atividades de gravação em uma tabela com muitos índices secundários serão mais altos do que em uma tabela com um número menor de índices. Para obter mais informações, consulte Considerações sobre throughput provisionado para índices secundários globais.

Classes de tabela com índice secundário global

Um índice secundário global sempre usará a mesma classe de tabela que sua tabela-base. Sempre que um novo índice secundário global for adicionado para uma tabela, o novo índice usará a mesma classe de tabela que sua tabela base. Quando a classe de tabela de uma tabela é atualizada, todos os índices secundários globais associados também são atualizados.

Considerações sobre throughput provisionado para índices secundários globais

Ao criar um índice secundário global em uma tabela de modo provisionado, você deve especificar unidades de capacidade de leitura e gravação para a workload esperada no índice. As configurações de throughput provisionado de um índice secundário global são separadas daquelas de sua tabela-base. Uma operação `Query` em um índice secundário global consome unidades de capacidade de leitura do índice, não da tabela-base. Quando você insere ou exclui itens em uma tabela, os índices secundários globais dessa tabela também são atualizados. Essas atualizações de índice consomem unidades de capacidade do índice, não da tabela base.

Por exemplo, se você usar a operação `Query` em um índice secundário global e exceder sua capacidade de leitura provisionada, sua solicitação será limitada. Se você executar atividades de gravação pesadas na tabela, mas um índice secundário global da tabela tiver capacidade de gravação insuficiente, a atividade de gravação na tabela será limitada.

> **Important**
>
> Para evitar o controle de utilização potencial, a capacidade de gravação provisionada para um índice secundário global deve ser igual ou maior que a capacidade de gravação da tabela base, porque as novas atualizações gravarão na tabela base e no índice secundário global.

Para visualizar as configurações de throughput provisionado de um índice secundário global, use a operação `DescribeTable`. Informações detalhadas sobre todos os índices secundários globais da tabela são retornados.

Unidades de capacidade de leitura

Os índices secundários globais oferecem suporte a leituras eventualmente consistentes, cada uma delas consome metade de uma unidade de capacidade de leitura. Isso significa que uma única consulta de índice secundário global pode recuperar até 2 × 4 KB = 8 KB por unidade de capacidade de leitura.

Para consultas de índice secundário global, o DynamoDB calcula a atividade de leitura provisionada da mesma forma que para consultas em tabelas. A única diferença é que o cálculo é baseado no tamanho das entradas de índice, em vez do tamanho do item na tabela-base. O número de unidades de capacidade de leitura é a soma de todos os tamanhos de atributos projetados em todos os itens retornados. O resultado é arredondado para o próximo limite de 4 KB. Para obter mais informações sobre como o DynamoDB calcula a utilização de throughput provisionado, consulte Modo de capacidade provisionada do DynamoDB.

O tamanho máximo dos resultados retornados por uma operação `Query` é 1 MB. Isso inclui os tamanhos de todos os nomes e valores de atributos de todos os itens retornados.

Por exemplo, considere um índice secundário global em que cada item contém 2.000 bytes de dados. Agora vamos supor que você use a operação `Query` nesse índice e que `KeyConditionExpression` da consulta retorne oito itens. O tamanho total dos itens correspondentes é 2.000 bytes × 8 itens = 16.000 bytes. O resultado é arredondado para o próximo limite de 4 KB. Como as consultas do índice secundário global são finais consistentes, o custo total é 0,5 × (16 KB / 4 KB), ou 2 unidades de capacidade de leitura.

Unidades de capacidade de gravação

Quando um item é adicionado, atualizado ou excluído em uma tabela, e um índice secundário global é afetado por isso, o índice secundário global consome unidades de capacidade de gravação provisionadas para a operação. O custo total do throughput provisionado para uma gravação consiste na soma das unidades de capacidade de gravação consumidas pela gravação na tabela base e pela atualização dos índices secundários globais. Se uma gravação em uma tabela não exigir uma atualização do índice secundário global, nenhuma capacidade de gravação será consumida no índice.

Para que uma gravação de tabela seja bem-sucedida, as configurações do throughput provisionado e todos os seus índices secundários globais devem ter capacidade de gravação suficiente para acomodar a gravação. Caso contrário, a gravação na tabela será limitada.

> **Important**
>
> Ao criar um índice secundário global (GSI), as operações de gravação na tabela base poderão sofrer controle de utilização se a atividade do GSI resultante das gravações na tabela base exceder a capacidade de gravação provisionada do GSI. Esse controle de utilização afeta todas as operações de gravação, desde o processo de indexação até a possível interrupção de suas workloads de produção. Para ter mais informações, consulte Problemas de controle de utilização no Amazon DynamoDB.

O custo de gravar um item em um índice secundário global depende de vários fatores:

- Se você gravar um novo item na tabela que define um atributo indexado, ou atualizar um item existente para definir um atributo indexado indefinido anteriormente, uma operação de gravação é necessária para inserir o item no índice.
- Se uma atualização na tabela alterar o valor de um atributo de chave indexado (de A para B), duas gravações serão necessárias, uma para excluir o item anterior do índice e outra gravação para inserir o novo item no índice.
- Se um item estava presente no índice, mas uma gravação na tabela fez com que o atributo indexado fosse excluído, uma gravação é necessária para excluir a projeção do item antigo do índice.
- Se um item não estiver presente no índice antes ou depois que o item é atualizado, não haverá custo de gravação adicionais para o índice.
- Se uma atualização na tabela alterar somente o valor dos atributos projetados no esquema de chaves do índice, mas não alterar o valores de qualquer atributo de chave indexado, uma gravação será necessária para atualizar os valores dos atributos projetados no índice.

Todos esses fatores supõem que o tamanho de cada item no índice seja menor ou igual ao tamanho de item de 1 KB para calcular unidades de capacidade de gravação. Entradas de índice maiores exigirão unidades adicionais de capacidade de gravação. Você pode minimizar os custos de gravação, considerando quais atributos suas consultas precisarão retornar e projetar apenas esses atributos no índice.

Considerações sobre armazenamento para índices secundários globais

Quando uma aplicação grava um item em uma tabela, o DynamoDB copia automaticamente o subconjunto de atributos corretos para qualquer índice secundário global no qual esses atributos devem aparecer. Sua conta da AWS é cobrada pelo armazenamento do item na tabela-base e também pelo armazenamento de atributos em qualquer índice secundário global dessa tabela.

A quantidade de espaço usada por um item do índice é a soma do seguinte:

- O tamanho em bytes da chave primária da tabela-base (chave de partição e chave de classificação)
- O tamanho em bytes do atributo de chave do índice
- O tamanho em bytes dos atributos projetados (se houver)
- 100 bytes de sobrecarga por item de índice

Para estimar os requisitos de armazenamento de um índice secundário global, você pode estimar o tamanho médio de um item no índice e multiplicar pelo número de itens da tabela-base que têm os atributos de chave do índice secundário global.

Se uma tabela contiver um item em que um determinado atributo não está definido, mas está definido como uma chave de partição ou chave de classificação do índice, o DynamoDB não gravará nenhum dado desse item no índice.

Gerenciar índices secundários globais no DynamoDB

Esta seção descreve como criar, modificar e excluir índices secundários globais no Amazon DynamoDB.

Tópicos

- Criar uma tabela com índices secundários globais
- Descrever os índices secundários globais em uma tabela
- Adicionar um índice secundário global a uma tabela
- Exclusão de um índice secundário global
- Modificar um índice secundário global durante a criação

Criar uma tabela com índices secundários globais

Para criar uma tabela com um ou mais índices secundários globais, use a operação `CreateTable` com o parâmetro `GlobalSecondaryIndexes`. Para obter a flexibilidade máxima de consultas, é possível criar até 20 índices secundários globais (cota padrão) por tabela.

Você deve especificar um atributo que não atue como chave de partição do índice. Como opção, você pode especificar outro atributo para a chave de classificação do índice. Não é necessário que nenhum desses atributos de chave seja o mesmo que um atributo de chave na tabela. Por exemplo, na tabela GameScores (consulte Como usar índices secundários globais no DynamoDB), nem `TopScore` nem `TopScoreDateTime` são atributos chave. Você poderia criar um índice secundário global com uma chave de partição `TopScore` e uma chave de classificação `TopScoreDateTime`. Um índice desse tipo pode ser usado para determinar se há uma correlação entre pontuações altas e a hora do dia em que um jogo é jogado.

Cada atributo de chave de índice deve ser um escalar do tipo `String`, `Number` ou `Binary`. (Ele não pode ser um documento ou um conjunto.) Você pode projetar atributos de qualquer tipo de dados em um índice secundário global. Isso inclui escalares, documentos e conjuntos. Para obter uma lista completa de tipos de dados, consulte Tipos de dados.

Se estiver usando o modo provisionado, você deve fornecer configurações de `ProvisionedThroughput` para o índice, formadas por `ReadCapacityUnits` e `WriteCapacityUnits`. Essas configurações de throughput provisionado são distintas daquelas na tabela, mas se comportam de forma semelhante. Para obter mais informações, consulte Considerações sobre throughput provisionado para índices secundários globais.

Índices secundários globais herdam o modo de capacidade leitura/gravação da tabela base. Para obter mais informações, consulte Considerações ao alternar os modos de capacidade no DynamoDB.

> **Note**
>
> As operações de provisionamento e as operações de gravação em andamento compartilham o throughput de gravação dentro do índice secundário. Ao criar um novo GSI, pode ser importante conferir se a escolha de chave de partição está gerando uma distribuição desigual ou estreita de dados ou tráfego nos valores de chave de partição do novo índice. Se isso ocorrer, você pode estar vendo operações de provisionamento e de gravação ocorrendo ao mesmo tempo e restringindo as gravações na tabela base. O serviço toma medidas para minimizar o potencial desse caso, mas não tem insights sobre o formato dos dados do cliente

em relação à chave de partição de índice, à projeção escolhida ou à dispersão da chave primária do índice.
Se você suspeitar que o novo índice secundário global tenha dados estreitos ou distorcidos ou distribuição de tráfego entre valores de chave de partição, considere o seguinte antes de adicionar novos índices a tabelas operacionalmente importantes.

- Talvez seja mais seguro adicionar o índice no momento em que a aplicação está gerando a menor quantidade de tráfego.
- Considere habilitar o CloudWatch Contributor Insights em sua tabela e índices de base. Isso lhe proporcionará um valioso insight sobre sua distribuição de tráfego.
- Para tabelas e índices base no modo de capacidade provisionada, defina a capacidade de gravação provisionada do novo índice para pelo menos o dobro da tabela base. Observe as métricas do CloudWatch `WriteThrottleEvents`, `ThrottledRequests`, `OnlineIndexPercentageProgress`, `OnlineIndexConsumedWriteCapacity` e `OnlineIndexThrottleEvents` em todo o processo. Ajuste a capacidade de gravação provisionada conforme necessário para concluir o provisionamento em tempo razoável, sem efeitos consideráveis de controle de utilização nas operações em andamento.
- Esteja preparado para cancelar a criação do índice, em caso de impacto operacional devido ao controle de utilização de gravação, e caso o aumento de capacidade de gravação provisionada em seu novo GSI não solucione.

Descrever os índices secundários globais em uma tabela

Para ver o status de todos os índices secundários globais em uma tabela, use a operação `DescribeTable`. A parte `GlobalSecondaryIndexes` da resposta mostra todos os índices na tabela, juntamente com o status atual de cada (`IndexStatus`).

O `IndexStatus` para um índice secundário global será um dos seguintes:

- `CREATING`: o índice está sendo criado e ainda não está disponível para uso.
- `ACTIVE`: o índice está pronto para uso e as aplicações podem executar operações `Query` no índice.
- `UPDATING`: as configurações de throughput provisionado do índice estão sendo alteradas.
- `DELETING`: o índice está sendo excluído e não pode mais ser usado.

Quando o DynamoDB tiver terminado de criar um índice secundário global, o status do índice mudará de `CREATING` para `ACTIVE`.

Adicionar um índice secundário global a uma tabela

Para adicionar um índice secundário global a uma tabela existente, use a operação `UpdateTable` com o parâmetro `GlobalSecondaryIndexUpdates`. Você deve fornecer o seguinte:

- Um nome de índice. O nome deve ser exclusivo entre todos os índices na tabela.
- O esquema de chave do índice. É necessário especificar um atributo para a chave da partição de índice, mas existe a opção de especificar outro atributo para a chave de classificação de índice. Não é necessário que nenhum desses atributos de chave seja o mesmo que um atributo de chave na tabela. Os tipos de dados de cada atributo de esquema devem ser escalares: `String`, `Number` ou `Binary`.
- Os atributos a serem projetados da tabela para o índice:
  - `KEYS_ONLY`: cada item do índice consiste apenas nos valores de chaves de partição e nas chaves de classificação da tabela, além dos valores de chaves do índice.
  - `INCLUDE`: além dos atributos descritos em `KEYS_ONLY`, o índice secundário inclui outros atributos não chave que você especificar.
  - `ALL`: o índice inclui todos os atributos da tabela de origem.
- As configurações do throughput provisionado para o índice, formadas por `ReadCapacityUnits` e `WriteCapacityUnits`. Essas configurações de throughput provisionado são distintas daquelas na tabela.

Você pode criar somente um índice secundário global por operação `UpdateTable`.

> **Important**
>
> Ao criar um índice secundário global (GSI), as operações de gravação na tabela base poderão sofrer controle de utilização se a atividade do GSI resultante das gravações na tabela base exceder a capacidade de gravação provisionada do GSI. Esse controle de utilização afeta todas as operações de gravação, desde o processo de indexação até a possível interrupção de suas workloads de produção. Para ter mais informações, consulte Problemas de controle de utilização no Amazon DynamoDB.

Fases da criação de um índice

Quando você adiciona um novo índice secundário global a uma tabela existente, ela continua a estar disponível enquanto o índice está sendo construído. No entanto, o novo índice apenas estará disponível para operações de consulta quando seu status mudar de `CREATING` para `ACTIVE`.

> **Note**
>
> A criação do índice secundário global não usa o Application Auto Scaling. Aumentar a capacidade `MIN` do Application Auto Scaling não diminuirá o tempo de criação do índice secundário global.

Nos bastidores, o DynamoDB constrói o índice em duas fases:

Alocação de recursos

- O DynamoDB aloca os recursos de computação e de armazenamento que são necessários para a construção do índice.

  Durante a fase de alocação de recursos, o atributo `IndexStatus` é `CREATING`, enquanto o atributo `Backfilling` é false. Use a operação `DescribeTable` para recuperar o status de uma tabela e todos os índices secundários dela.

  Enquanto o índice estiver na fase de alocação de recurso, você poderá excluí-lo ou a apagar a tabela principal dele. Você também não pode modificar o throughput provisionado do índice ou da tabela. Não é possível adicionar ou excluir outros índices na tabela. No entanto, você pode modificar o throughput provisionado desses outros índices.

Aterramento

- Para cada item na tabela, o DynamoDB determina qual conjunto de atributos deve ser gravado no índice com base em sua projeção (`KEYS_ONLY`, `INCLUDE` ou `ALL`). Em seguida, ele grava esses atributos no índice. Durante a fase de preenchimento, o DynamoDB rastreia os itens que estão sendo adicionados, excluídos ou atualizados na tabela. Os atributos desses itens também são adicionados, excluídos ou atualizados no índice conforme apropriado.

  Durante a fase de preenchimento, o atributo `IndexStatus` é definido como `CREATING`, e o atributo `Backfilling` é verdadeiro. Use a operação `DescribeTable` para recuperar o status de uma tabela e todos os índices secundários dela.

Enquanto o índice está em processo de aterramento, não é possível excluir a tabela principal. No entanto, ainda é possível excluir o índice ou modificar o throughput provisionado da tabela e de qualquer um dos seus índices secundários globais.

> **Note**
>
> Durante a fase de aterramento, algumas gravações de itens infratores podem ter sucesso, enquanto outras serão rejeitadas. Após o aterramento, todas as gravações de itens que violarem o esquema de chaves do novo índice serão rejeitadas. Recomendamos executar a ferramenta Violation Detector após a conclusão da fase de preenchimento em segundo plano, para detectar e resolver qualquer infração de chave que possa ter ocorrido. Para obter mais informações, consulte Detectar e corrigir violações de chave de índice no DynamoDB.

Enquanto as fases de alocação de recurso e aterramento estão em andamento, o índice permanece no estado `CREATING`. Enquanto isso, o DynamoDB executa operações de leitura na tabela. Você não é cobrado pelas operações de leitura da tabela-base para preencher o índice secundário global. No entanto, você será cobrado pelas operações de gravação para preencher o índice secundário global recém-criado.

Quando a construção do índice estiver concluída, seu status mudará para `ACTIVE`. Você não pode executar `Query` ou `Scan` no índice até que ele esteja no modo `ACTIVE`.

> **Note**
>
> Em alguns casos, o DynamoDB não pode gravar dados da tabela no índice devido a violações de chaves de índice. Isso poderá ocorrer se:
>
> - O tipo de dados de um valor de atributo não corresponder ao tipo de dados de um esquema de chaves de índice.
> - O tamanho de um atributo excede o tamanho máximo de um atributo de chave de índice.
> - Um atributo de chave de índice tem um valor binários ou de string vazio.
>
> Violações de chaves de índice não interferem na criação do índice secundário global. No entanto, quando o índice ficar `ACTIVE`, as chaves infratoras não estarão presentes no índice.

O DynamoDB fornece uma ferramenta autônoma para localizar e resolver esses problemas. Para obter mais informações, consulte Detectar e corrigir violações de chave de índice no DynamoDB.

Adicionar um índice secundário global a uma tabela grande

O tempo necessário para a construção de um índice secundário global depende de vários fatores, entre eles:

- O tamanho da tabela
- O número de itens na tabela que se qualificam para inclusão no índice
- O número de atributos projetados no índice
- A capacidade de gravação provisionada do índice
- A atividade de gravação na tabela principal durante a criação dos índices

Se você estiver adicionando um índice secundário global a uma tabela muito grande, a conclusão do processo de criação poderá ser demorada. Para monitorar o progresso e determinar se o índice tem capacidade de gravação suficiente, consulte as seguintes métricas do Amazon CloudWatch:

- `OnlineIndexPercentageProgress`
- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

Para obter mais informações sobre as métricas do CloudWatch relacionadas ao DynamoDB, consulte Métricas do DynamoDB.

Important

Talvez seja necessário incluir tabelas muito grandes na lista de permissões antes de criar ou atualizar um índice secundário global. Entre em contato com o AWS Support para incluir tabelas na lista de permissões.

> **Note**
>
> Como regra geral, recomendamos definir a capacidade de gravação provisionada do índice como 1,5 vezes maior que a capacidade de gravação da tabela. Essa é uma boa configuração para muitos casos de uso. No entanto, seus requisitos reais podem ser maiores ou menores.

Enquanto um índice está sendo preenchido em segundo plano, o DynamoDB usa a capacidade do sistema interno para fazer leituras na tabela. Isso é feito para minimizar o impacto da criação do índice e garantir que sua tabela não fique sem capacidade de leitura.

No entanto, é possível que o volume de atividades de gravação recebidas exceda a capacidade de gravação provisionada do índice. Este é um cenário de afunilamento, no qual a criação do índice é mais demorada porque a atividade de gravação no índice é limitada. Durante a criação do índice, é recomendável monitorar as métricas do Amazon CloudWatch para o índice a fim de determinar se a capacidade de gravação consumida está excedendo sua capacidade provisionada. Em um cenário de afunilamento, é necessário aumentar a capacidade de gravação provisionada no índice para evitar o controle de utilização de gravações durante a fase de aterramento.

Depois que o índice tiver sido criado, você deverá definir sua capacidade de gravação provisionada para refletir o uso normal de sua aplicação.

Exclusão de um índice secundário global

Se você não precisa mais de um índice secundário global, pode excluí-lo usando a operação `UpdateTable`.

É possível excluir somente um índice secundário global por operação `UpdateTable`.

Enquanto o índice secundário global está sendo excluído, não há efeitos sobre atividades de leitura ou de gravação na tabela principal. Enquanto a exclusão está em andamento, você ainda pode modificar o throughput provisionado em outros índices.

> **Note**
>
> - Quando você exclui uma tabela usando a ação `DeleteTable`, todos os índices secundários globais nessa tabela também são excluídos.
> - Sua conta não será cobrada pela operação de exclusão do índice secundário global.

Modificar um índice secundário global durante a criação

Enquanto um índice está sendo construído, é possível usar a operação `DescribeTable` para determinar em qual fase ele se encontra. A descrição do índice inclui um atributo booleano, `Backfilling`, para indicar se o DynamoDB está carregando o índice com itens da tabela no momento. Se `Backfilling` for verdadeiro, a fase de alocação de recursos estará concluída, e o índice agora estará sendo preenchido.

Durante o aterramento, é possível atualizar os parâmetros de throughput provisionado do índice. Você pode optar por fazer isso para acelerar a construção do índice: é possível aumentar a capacidade de gravação do índice enquanto ele está sendo construído e, em seguida, reduzi-la. Para modificar as configurações de throughput provisionado do índice, use a operação `UpdateTable`. O status do índice muda para `UPDATING`, e `Backfilling` é true até o índice estar pronto para uso.

Durante a fase de aterramento, é possível excluir o índice que está sendo criado. Durante essa fase, não é possível adicionar ou excluir outros índices na tabela.

> **Note**
>
> Para índices que foram criados como parte de uma operação `CreateTable`, o atributo `Backfilling` não aparece na saída de `DescribeTable`. Para obter mais informações, consulte Fases da criação de um índice.

Detectar e corrigir violações de chave de índice no DynamoDB

Durante a fase de preenchimento em segundo plano da criação de um índice secundário global, o Amazon DynamoDB examina cada item na tabela para determinar se ele está qualificado para inclusão no índice. Alguns itens podem não ser qualificados, pois causariam violações de chave de índice. Nesses casos, os itens permanecerão na tabela, mas o índice não terá uma entrada correspondente para eles.

Uma infração na chave do índice ocorre nas seguintes situações:

- Há uma inconsistência de tipo de dados entre um valor de atributo e o tipo de dados do esquema de chaves de índice. Por exemplo, suponha que um dos itens na tabela `GameScores` tivesse um valor `TopScore` do tipo `String`. Se você adicionasse um `TopScore` com uma chave de partição de do tipo `Number`, o item da tabela violaria a chave de índice.

- Um valor de atributo da tabela excede o comprimento máximo para um atributo de chave de índice. O comprimento máximo de uma chave de partição é 2048 bytes, enquanto o comprimento máximo de uma chave de classificação é 1024 bytes. Se qualquer um dos valores de atributo correspondente na tabela exceder esses limites, o item da tabela violará a chave de índice.

> **Note**
>
> Se um valor de atributo binário ou de string for definido para um atributo usado como uma chave de índice, o valor do atributo deverá ter um tamanho maior que zero. Caso contrário, o item da tabela violaria a chave de índice.
> Esta ferramenta não sinaliza esta infração na chave do índice neste momento.

Se uma infração na chave do índice ocorrer, a fase de preenchimento em segundo plano continuará sem interrupção. No entanto, os itens infratores não são incluídos no índice. Concluída a fase de aterramento, todas as gravações em itens que violam o esquema de chaves do novo índice serão rejeitadas.

Para identificar e corrigir os valores de atributos em uma tabela que violam uma chave de índice, use a ferramenta Violation Detector. Para executar o Violation Detector, crie um arquivo de configuração que especifique o nome de uma tabela a ser verificada, os nomes e os tipos de dados da chave de partição do índice secundário global e a chave de classificação, bem como quais ações deverão ser realizadas se violações de chave de índice forem encontradas. O Violation Detector pode ser executado em um destes dois diferentes modos:

- Modo de detecção: detecta violações de chave de índice. Use o modo de detecção para informar os itens na tabela que causariam violações de chaves em um índice secundário global. (Você tem a opção de solicitar que esses itens de tabela infratores sejam excluídos imediatamente quando forem encontrados.) A saída do modo de detecção é gravada em um arquivo, que você pode usar para análise posterior.
- Modo de correção: corrige violações de chaves de índice. No modo de correção, o Violation Detector lê um arquivo de entrada com o mesmo formato que o arquivo de saída do modo de detecção. O modo de correção lê os registros do arquivo de entrada e, para cada registro, ele exclui ou atualiza os itens correspondentes na tabela. (Observe que, se você optar por atualizar os itens, deverá editar o arquivo de entrada e definir os valores apropriados para essas atualizações.)

Baixar e executar o Violation Detector

O Violation Detector está disponível como um arquivo Java executável (arquivo `.jar`) e pode ser executado em computadores Windows, Mac ou Linux. O Violation Detector requer o Java 1.7 (ou posterior) e o Apache Maven.

- Baixar o Violation Detector do GitHub

Siga as instruções no arquivo `README.md` para fazer download e instalar o Violation Detector usando o Maven.

Para iniciar o Violation Detector, acesse o diretório no qual você compilou o arquivo `ViolationDetector.java` e insira o seguinte comando:

```
java -jar ViolationDetector.jar [options]
```

A linha de comando do Violation Detector aceita as seguintes opções:

- `-h | --help`: imprime um resumo de uso e opções do Violation Detector.
- `-p | --configFilePath value`: o nome totalmente qualificado de um arquivo de configuração do Violation Detector. Para ter mais informações, consulte O arquivo de configuração do Violation Detector.
- `-t | --detect value`: detecta violações de chave de índice na tabela e as grava no arquivo de saída do Violation Detector. Se o valor desse parâmetro for definido como `keep`, os itens com violações de chave não serão modificados. Se o valor for definido como `delete`, os itens com violações de chave serão excluídos da tabela.
- `-c | --correct value`: lê violações de chave de índice de um arquivo de entrada e toma ações corretivas nos itens da tabela. Se o valor desse parâmetro for definido como `update`, os itens com violações de chave serão atualizados com valores novos não infratores. Se o valor for definido como `delete`, os itens com violações de chave serão excluídos da tabela.

O arquivo de configuração do Violation Detector

Em tempo de execução, a ferramenta Violation Detector requer um arquivo de configuração Os parâmetros nesse arquivo determinam quais recursos do DynamoDB o Violation Detector pode acessar e quanto throughput provisionado ele pode consumir. A tabela a seguir descreve esses parâmetros.

Nome do parâmetro	Descrição	Obrigatório?

`awsCredentialsFile`	O nome totalmente qualificado de um arquivo que contém suas credenciais da AWS. O arquivo de credenciais deve estar no seguinte formato:
	`accessKey = access_key_id_goes_here`
	`secretKey = secret_key_goes_here`
		Sim

`dynamoDBRegion`	A região da AWS na qual a tabela reside. Por exemplo: `us-west-2` .	Sim
`tableName`	O nome da tabela do DynamoDB a ser verificada.	Sim
`gsiHashKeyName`	O nome da chave de partição do índice.	Sim
`gsiHashKeyType`	O tipo de dados da chave de partição do índice: `String`, `Number` ou `Binary` `S | N | B`	Sim
`gsiRangeKeyName`	O nome da chave de classificação do índice. Não especifique esse parâmetro se o índice tiver apenas uma chave primária simples (chave de partição).	Não

Nome do parâmetro	Descrição	Obrigatório?
`gsiRangeKeyType`	O tipo de dados da chave de classificação do índice: `String`, `Number` ou `Binary` `S | N | B` Não especifique esse parâmetro se o índice tiver apenas uma chave primária simples (chave de partição).	Não
`recordDetails`	Se você deseja gravar os detalhes completos das violações de chaves de índice no arquivo de saída. Se a opção for definida como `true` (o padrão), todas as informações sobre os itens infratores serão relatadas. Se definido como `false`, apenas o número de violações será relatado.	Não
`recordGsiValueInViolationRecord`	Se você deseja gravar os valores das chaves de índice infratoras no arquivo de saída. Se a opção for definida como `true` (padrão), os valores das chaves serão relatados. Se definido como `false`, os valores das chave não serão relatados.	Não

Nome do parâmetro	Descrição	Obrigatório?

`detectionOutputPath`	O caminho completo do arquivo de saída do Violation Detector. Esse parâmetro oferece suporte à gravação em um diretório local ou no Amazon Simple Storage Service (Amazon S3). Veja os exemplos a seguir:
	
	`detectionOutputPath = //local/path/filename.csv`
	
	`detectionOutputPath = s3://bucket/filename.csv`
	
	As informações no arquivo de saída são mostradas no formato CSV (valores separados por vírgula). Se você não definir `detectionOutputPath`, o arquivo de saída se chamará `violation_detection.csv` e será gravado no seu diretório de trabalho atual.
		Não

Nome do parâmetro	Descrição	Obrigatório?
`numOfSegments`	O número de segmentos de verificação paralela a serem usados quando o Violation Detector verificar a tabela. O valor padrão é 1, o que significa que a tabela é verificada de maneira sequencial. Se o valor for 2 ou superior, o Violation Detector dividirá a tabela no número correspondente de segmentos lógicos e em um número igual de threads de verificação. A configuração máxima para `numOfSegments` é 4096. Para tabelas maiores, uma verificação paralela é geralmente mais rápida do que uma verificação sequencial. Além disso, se a tabela for grande o suficiente para abranger várias partições, uma verificação paralela distribui sua atividade de leitura uniformemente entre essas várias partições. Para obter mais informações sobre verificações paralelas no DynamoDB, consulte Verificar em paralelo.	Não

Nome do parâmetro	Descrição	Obrigatório?
`numOfViolations`	O limite superior de violações de chaves de índice a serem gravadas no arquivo de saída. Se definido como `-1` (o padrão), a tabela inteira é verificada. Se definido como um número inteiro positivo, o Violation Detector será interrompido quando encontrar esse número de violações.	Não
`numOfRecords`	O número de itens na tabela a ser verificada. Se definido como -1 (o padrão), a tabela inteira é verificada. Se definido como um número inteiro positivo, o Violation Detector será interrompido após verificar o mesmo número de itens na tabela.	Não
`readWriteIOPSPercent`	Regula a porcentagem de unidades de capacidade de leitura provisionada que são consumidas durante a verificação de tabela. Os valores válidos variam de 1 até `100`. O valor padrão (25) significa que o Violation Detector não consumirá mais de 25% do throughput provisionado da tabela.	Não

Nome do parâmetro	Descrição	Obrigatório?
`correctionInputPath`	O caminho completo do arquivo de entrada de correção do Violation Detector. Se você executar o Violation Detector no modo de correção, o conteúdo desse arquivo será usado para modificar ou excluir os itens de dados na tabela que violam o índice secundário global. O formato do arquivo `correctionInputPath` é o mesmo que o do arquivo `detectionOutputPath` . Isso permite que você processe a saída do modo de detecção como uma entrada no modo de correção.	Não

Nome do parâmetro	Descrição	Obrigatório?

`correctionOutputPath`	O caminho completo do arquivo de saída de correção do Violation Detector. Esse arquivo será criado somente se houver erros de atualizaç ão.
	
	Esse parâmetro oferece suporte à gravação em um diretório local ou no Amazon S3. Veja os exemplos a seguir:
	
	`correctionOutputPa th = //local/path/ filename.csv`
	
	`correctio nOutputPath = s3://bucket/filename. csv`
	
	As informações no arquivo de saída são mostradas no formato CSV. Se você não definir `correctio nOutputPath` , o arquivo de saída se chamará `violation_update_e rrors.csv` e será gravado no seu diretório de trabalho atual.
		Não

Detecção

Para detectar violações de chave de índice, use o Violation Detector com a opção de linha de comando `--detect`. Para mostrar como essa opção funciona, considere a tabela `ProductCatalog`. Veja a seguir uma lista de itens na tabela. Apenas a chave primária (`Id`) e o atributo `Price` são mostrados.

ID (chave primária)	Preço
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Todos os valores para `Price` são do tipo `Number`. No entanto, como o DynamoDB não tem esquema, é possível adicionar um item com um `Price` não numérico. Por exemplo, suponha que adicionemos outro item à tabela `ProductCatalog`:

ID (chave primária)	Preço
999	"Hello"

Agora, a tabela tem um total de nove itens.

Agora você adiciona um novo índice secundário global à tabela: `PriceIndex`. A chave primária desse índice é uma chave de partição, `Price`, do tipo `Number`. Depois que o índice tiver sido construído, ele conterá oito itens, mas a tabela `ProductCatalog` tem nove itens. O motivo dessa

discrepância é que o valor `"Hello"` é do tipo `String`, mas `PriceIndex` tem uma chave primária do tipo `Number`. O valor de `String` viola a chave do índice secundário global e, por isso, não está presente no índice.

Para usar o Violation Detector nesse cenário, você primeiro deve criar um arquivo de configuração como este.

```
# Properties file for violation detection tool configuration.
# Parameters that are not specified will use default values.

awsCredentialsFile = /home/alice/credentials.txt
dynamoDBRegion = us-west-2
tableName = ProductCatalog
gsiHashKeyName = Price
gsiHashKeyType = N
recordDetails = true
recordGsiValueInViolationRecord = true
detectionOutputPath = ./gsi_violation_check.csv
correctionInputPath = ./gsi_violation_check.csv
numOfSegments = 1
readWriteIOPSPercent = 40
```

Em seguida, execute o Violation Detector como no exemplo a seguir.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --detect keep

Violation detection started: sequential scan, Table name: ProductCatalog, GSI name:
 PriceIndex
Progress: Items scanned in total: 9,    Items scanned by this thread: 9,    Violations
 found by this thread: 1, Violations deleted by this thread: 0
Violation detection finished: Records scanned: 9, Violations found: 1, Violations
 deleted: 0, see results at: ./gsi_violation_check.csv
```

Se o parâmetro de configuração `recordDetails` for definido como `true`, o Violation Detector gravará os detalhes de cada infração no arquivo de saída, como no exemplo a seguir.

```
Table Hash Key,GSI Hash Key Value,GSI Hash Key Violation Type,GSI Hash Key Violation
 Description,GSI Hash Key Update Value(FOR USER),Delete Blank Attributes When Updating?
(Y/N)
```

```
999,"{""S"":""Hello""}",Type Violation,Expected: N Found: S,,
```

O arquivo de saída está no formato CSV. A primeira linha do arquivo é um cabeçalho, seguida de um registro por item que viola a chave de índice. Os campos desses registros infratores são os seguintes:

- Table Hash Key (Chave de hash da tabela): o valor da chave de partição do item na tabela.
- Table Range Key (Chave de classificação da tabela): o valor da chave de classificação do item na tabela.
- GSI Hash Key Value (Valor da chave de hash do GSI): o valor da chave de partição do índice secundário global.
- GSI Hash Key Violation Type (Tipo de violação da chave de hash do GSI): `Type Violation` ou `Size Violation`.
- GSI Hash Key Violation Description (Descrição da violação da chave de hash do GSI): a causa da infração.
- GSI Hash Key Update Value(FORUSER) (Valor da atualização da chave de hash do GSI [PARA USUÁRIO]): no modo de correção, um novo valor fornecido pelo usuário para o atributo.
- GSI Hash Key Value (Valor da chave de hash do GSI): o valor da chave de classificação do índice secundário global.
- GSI Range Key Violation Type (Tipo de violação da chave de intervalo do GSI: `Type Violation` ou `Size Violation`.
- GSI Range Key Violation Description (Descrição da violação da chave de intervalo do GSI): a causa da infração.
- GSI Range Key Update Value(FOR USER) (Valor da atualização da chave de intervalo do GSI [PARA USUÁRIO]: no modo de correção, um novo valor fornecido pelo usuário para o atributo.
- Delete Blank Attribute When Updating(Y/N) (Excluir atributo em branco ao atualizar[S/N]): no modo de correção, determina se o item infrator deve ser excluído (Y) ou mantido (N) na tabela, mas apenas se algum dos seguintes campos estiver em branco:
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  Se qualquer um desses campos não estiver em branco, `Delete Blank Attribute When Updating(Y/N)` não terá efeito.

> **Note**
>
> O formato de saída pode variar dependendo do arquivo de configuração e das opções da linha de comando. Por exemplo, se a tabela tiver uma chave primária simples (sem uma chave de classificação), nenhum campo de chave de classificação estará presente na saída. Os registros de infração no arquivo podem não estar na ordem classificada.

Correção

Para corrigir violações de chave de índice, use o Violation Detector com a opção de linha de comando `--correct`. No modo de correção, o Violation Detector lê o arquivo de entrada especificado pelo parâmetro `correctionInputPath`. Esse arquivo tem o mesmo formato que o arquivo `detectionOutputPath` e, portanto, você pode usar a saída da detecção como a entrada para a correção.

O Violation Detector fornece duas maneiras diferentes para corrigir violações de chave de índice:

- Excluir violações: exclua os itens da tabela que possuem valores de atributo infratores.
- Atualizar violações: atualize os itens da tabela substituindo os atributos infratores por valores não infratores.

Em ambos os casos, você pode usar o arquivo de saída do modo de detecção como uma entrada para o modo de correção.

Continuando com o exemplo de `ProductCatalog`, suponha que queiramos excluir o item infrator da tabela. Para fazer isso, use a seguinte linha de comando:

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

Neste ponto, você será solicitado a confirmar se deseja excluir os itens infratores.

```
Are you sure to delete all violations on the table?y/n
y
Confirmed, will delete violations on the table...
Violation correction from file started: Reading records from file: ./
gsi_violation_check.csv, will delete these records from table.
Violation correction from file finished: Violations delete: 1, Violations Update: 0
```

Agora, tanto `ProductCatalog` quanto `PriceIndex` têm o mesmo número de itens.

Como trabalhar com índices secundários globais: Java

Você pode usar a API de documentos do AWS SDK para Java para criar uma tabela do Amazon DynamoDB com um ou mais índices secundários globais na tabela e executar consultas usando os índices.

Veja a seguir as etapas comuns para as operações de tabela.

1. Crie uma instância da classe `DynamoDB`.
2. Forneça os parâmetros obrigatórios e opcionais para a operação, criando os objetos de solicitação correspondentes.
3. Chame o método apropriado fornecido pelo cliente que você criou na etapa anterior.

Tópicos

- Criar uma tabela com um índice secundário global
- Descrever uma tabela com um índice secundário global
- Consultar um índice secundário global
- Exemplo: índices secundários globais que usam a API de documento do AWS SDK para Java

Criar uma tabela com um índice secundário global

Você pode criar índices secundários globais ao mesmo tempo em que cria uma tabela. Para fazer isso, use `CreateTable` e forneça suas especificações para um ou mais índices secundários globais. O exemplo de código Java a seguir cria uma tabela para armazenar informações sobre dados climáticos. A chave de partição é `Location` e a chave de classificação é `Date`. Um índice secundário global chamado `PrecipIndex` permite acesso rápido aos dados de precipitação de vários locais.

Veja a seguir as etapas necessárias para criar uma tabela com um índice secundário global usando a API de documentos do DynamoDB.

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `CreateTableRequest` para fornecer as informações solicitadas.

   Você deve fornecer o nome da tabela, sua chave primária e os valores de throughput provisionado. Para o índice secundário global, você deve fornecer o nome do índice, suas

configurações de throughput provisionado, as definições de atributo da chave de classificação do índice, o esquema de chaves do índice e a projeção do atributo.

3. Chame o método `createTable`, fornecendo o objeto de solicitação como um parâmetro.

O exemplo de código Java a seguir demonstra as etapas anteriores. O código cria uma tabela (`WeatherData`) com um índice secundário global (`PrecipIndex`). A chave de partição do índice é `Date` e a chave de classificação é `Precipitation`. Todos os atributos da tabela estão projetados no índice. Os usuários podem consultar esse índice para obter dados climáticos de uma data específica, opcionalmente, classificar os dados por quantidade de precipitação.

Como `Precipitation` não é um atributo de chave para a tabela, ele não é necessário. No entanto, os itens de `WeatherData` sem `Precipitation` não aparecem no `PrecipIndex`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

// Attribute definitions
ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Location")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Date")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Precipitation")
    .withAttributeType("N"));

// Table key schema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Location")
    .withKeyType(KeyType.HASH));  //Partition key
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.RANGE));  //Sort key

// PrecipIndex
GlobalSecondaryIndex precipIndex = new GlobalSecondaryIndex()
```

```
    .withIndexName("PrecipIndex")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 10)
        .withWriteCapacityUnits((long) 1))
        .withProjection(new Projection().withProjectionType(ProjectionType.ALL));

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();

indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.HASH));  //Partition key
indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Precipitation")
    .withKeyType(KeyType.RANGE));  //Sort key

precipIndex.setKeySchema(indexKeySchema);

CreateTableRequest createTableRequest = new CreateTableRequest()
    .withTableName("WeatherData")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 5)
        .withWriteCapacityUnits((long) 1))
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(tableKeySchema)
    .withGlobalSecondaryIndexes(precipIndex);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Você deve aguardar até que o DynamoDB crie a tabela e defina o status dessa tabela como `ACTIVE`. Depois disso, você poderá começar a inserir itens de dados na tabela.

Descrever uma tabela com um índice secundário global

Para obter mais informações sobre índices secundários globais em uma tabela, use `DescribeTable`. Para cada índice, você pode acessar seu nome, esquema de chaves e atributos projetados.

Veja a seguir as etapas necessárias para acessar informações de índice secundário global em uma tabela.

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `Table` para representar o índice com o qual você deseja trabalhar.

3. Chame o método `describe` no objeto `Table`.

O exemplo de código Java a seguir demonstra as etapas anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
TableDescription tableDesc = table.describe();


Iterator<GlobalSecondaryIndexDescription> gsiIter =
 tableDesc.getGlobalSecondaryIndexes().iterator();
while (gsiIter.hasNext()) {
    GlobalSecondaryIndexDescription gsiDesc = gsiIter.next();
    System.out.println("Info for index "
         + gsiDesc.getIndexName() + ":");

    Iterator<KeySchemaElement> kseIter = gsiDesc.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = gsiDesc.getProjection();
    System.out.println("\tThe projection type is: "
        + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: "
            + projection.getNonKeyAttributes());
    }
}
```

Consultar um índice secundário global

Você pode usar `Query` em um índice secundário global de forma semelhante ao uso de `Query` em uma tabela. Você precisa especificar o nome do índice, os critérios de consulta da chave de partição e da chave de classificação (se houver) do índice, e os atributos que você deseja retornar. Neste exemplo, o índice é `PrecipIndex`, que tem uma chave de partição `Date` e uma chave de classificação `Precipitation`. A consulta de índice retorna todos os dados climáticos de uma data específica, na qual a precipitação é maior que zero.

Veja a seguir as etapas necessárias para consultar um índice secundário global usando a API de documentos do AWS SDK para Java.

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `Table` para representar o índice com o qual você deseja trabalhar.
3. Crie uma instância da classe `Index` para o índice que deseja consultar.
4. Chame o método `query` no objeto `Index`.

O nome do atributo `Date` é uma palavra reservada do DynamoDB. Portanto, use um nome de atributo de expressão como um espaço reservado na `KeyConditionExpression`.

O exemplo de código Java a seguir demonstra as etapas anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
Index index = table.getIndex("PrecipIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("#d = :v_date and Precipitation = :v_precip")
    .withNameMap(new NameMap()
        .with("#d", "Date"))
    .withValueMap(new ValueMap()
        .withString(":v_date","2013-08-10")
        .withNumber(":v_precip",0));

ItemCollection<QueryOutcome> items = index.query(spec);
Iterator<Item> iter = items.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next().toJSONPretty());
}
```

Exemplo: índices secundários globais que usam a API de documento do AWS SDK para Java

O código Java de exemplo a seguir mostra como trabalhar com índices secundários globais. O exemplo cria uma tabela chamada `Issues`, que pode ser usada em um sistema de controle de

bugs simples para desenvolvimento de software. A chave de partição é `IssueId` e a chave de classificação é `Title`. Há três índices secundários globais nessa tabela:

- `CreateDateIndex`: a chave de partição é `CreateDate` e a chave de classificação é `IssueId`. Além das chaves da tabela, os atributos `Description` e `Status` são projetados no índice.
- `TitleIndex`: a chave de partição é `Title` e a chave de classificação é `IssueId`. Nenhum outro atributo além das chaves da tabela são projetos no índice.
- `DueDateIndex`: a chave de partição é `DueDate` e não há chave de classificação. Todos os atributos da tabela estão projetados no índice.

Depois que a tabela `Issues` é criada, o programa carrega a tabela com os dados que representam relatórios de bugs do software. Ele consulta os dados usando os índices secundários globais. Por fim, o programa exclui a tabela `Issues`.

Para obter instruções passo a passo sobre como testar o exemplo a seguir, consulte Exemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.GlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.Projection;
```

```
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class DocumentAPIGlobalSecondaryIndexExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    public static String tableName = "Issues";

    public static void main(String[] args) throws Exception {

        createTable();
        loadData();

        queryIndex("CreateDateIndex");
        queryIndex("TitleIndex");
        queryIndex("DueDateIndex");

        deleteTable(tableName);

    }

    public static void createTable() {

        // Attribute definitions
        ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("IssueId").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Title").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("CreateDate").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("DueDate").withAttributeType("S"));

        // Key schema for table
        ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
        tableKeySchema.add(new
 KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.HASH)); //
 Partition

                    // key
```

```
        tableKeySchema.add(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.RANGE)); // Sort

                    // key

        // Initial provisioned throughput settings for the indexes
        ProvisionedThroughput ptIndex = new
ProvisionedThroughput().withReadCapacityUnits(1L)
                .withWriteCapacityUnits(1L);

        // CreateDateIndex
        GlobalSecondaryIndex createDateIndex = new
GlobalSecondaryIndex().withIndexName("CreateDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("CreateDate").withKeyType(KeyType.HASH), //
Partition

                          // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                 // key
                .withProjection(
                        new
Projection().withProjectionType("INCLUDE").withNonKeyAttributes("Description",
"Status"));

        // TitleIndex
        GlobalSecondaryIndex titleIndex = new
GlobalSecondaryIndex().withIndexName("TitleIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.HASH), // Partition

                     // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                 // key
                .withProjection(new Projection().withProjectionType("KEYS_ONLY"));

        // DueDateIndex
```

```
        GlobalSecondaryIndex dueDateIndex = new
GlobalSecondaryIndex().withIndexName("DueDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("DueDate").withKeyType(KeyType.HASH)) //
Partition

                        // key
                .withProjection(new Projection().withProjectionType("ALL"));

        CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                .withProvisionedThroughput(
                        new ProvisionedThroughput().withReadCapacityUnits((long)
1).withWriteCapacityUnits((long) 1))

.withAttributeDefinitions(attributeDefinitions).withKeySchema(tableKeySchema)
                .withGlobalSecondaryIndexes(createDateIndex, titleIndex, dueDateIndex);

        System.out.println("Creating table " + tableName + "...");
        dynamoDB.createTable(createTableRequest);

        // Wait for table to become active
        System.out.println("Waiting for " + tableName + " to become ACTIVE...");
        try {
            Table table = dynamoDB.getTable(tableName);
            table.waitForActive();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void queryIndex(String indexName) {

        Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
        System.out.print("Querying index " + indexName + "...");

        Index index = table.getIndex(indexName);

        ItemCollection<QueryOutcome> items = null;
```

```
        QuerySpec querySpec = new QuerySpec();

        if (indexName == "CreateDateIndex") {
            System.out.println("Issues filed on 2013-11-01");
            querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
                    .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "TitleIndex") {
            System.out.println("Compilation errors");
            querySpec.withKeyConditionExpression("Title = :v_title and
begins_with(IssueId, :v_issue)")
                    .withValueMap(
                            new ValueMap().withString(":v_title", "Compilation
error").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "DueDateIndex") {
            System.out.println("Items that are due on 2013-11-30");
            querySpec.withKeyConditionExpression("DueDate = :v_date")
                    .withValueMap(new ValueMap().withString(":v_date", "2013-11-30"));
            items = index.query(querySpec);
        } else {
            System.out.println("\nNo valid index name provided");
            return;
        }

        Iterator<Item> iterator = items.iterator();

        System.out.println("Query: printing results...");

        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    public static void deleteTable(String tableName) {

        System.out.println("Deleting table " + tableName + "...");

        Table table = dynamoDB.getTable(tableName);
        table.delete();
```

```
        // Wait for table to be deleted
        System.out.println("Waiting for " + tableName + " to be deleted...");
        try {
            table.waitForDelete();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void loadData() {

        System.out.println("Loading data into table " + tableName + "...");

        // IssueId, Title,
        // Description,
        // CreateDate, LastUpdateDate, DueDate,
        // Priority, Status

        putItem("A-101", "Compilation error", "Can't compile Project X - bad version
number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10", 1, "Assigned");

        putItem("A-102", "Can't read data file", "The main data file is missing, or the
permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30", 2, "In progress");

        putItem("A-103", "Test failure", "Functional test of Project X produces
errors", "2013-11-01", "2013-11-02",
                "2013-11-10", 1, "In progress");

        putItem("A-104", "Compilation error", "Variable 'messageCount' was not
initialized.", "2013-11-15",
                "2013-11-16", "2013-11-30", 3, "Assigned");

        putItem("A-105", "Network issue", "Can't ping IP address 127.0.0.1. Please fix
this.", "2013-11-15",
                "2013-11-16", "2013-11-19", 5, "Assigned");

    }

    public static void putItem(

            String issueId, String title, String description, String createDate, String
lastUpdateDate, String dueDate,
```

```
            Integer priority, String status) {

        Table table = dynamoDB.getTable(tableName);

        Item item = new Item().withPrimaryKey("IssueId", issueId).withString("Title",
 title)
                .withString("Description", description).withString("CreateDate",
 createDate)
                .withString("LastUpdateDate", lastUpdateDate).withString("DueDate",
 dueDate)
                .withNumber("Priority", priority).withString("Status", status);

        table.putItem(item);
    }

}
```

Como trabalhar com índices secundários globais: .NET

Você pode usar a API de baixo nível do AWS SDK para .NET para criar uma tabela do Amazon DynamoDB com um ou mais índices secundários globais na tabela e executar consultas usando os índices. Essas operações são mapeadas nas operações do DynamoDB correspondentes. Para obter mais informações, consulte a Referência de API do Amazon DynamoDB.

Veja a seguir as etapas comuns para operações de tabela usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Forneça os parâmetros obrigatórios e opcionais para a operação, criando os objetos de solicitação correspondentes.

   Por exemplo, crie um objeto `CreateTableRequest` para criar uma tabela e um objeto `QueryRequest` para consultar uma tabela ou um índice.
3. Execute o método apropriado fornecido pelo cliente que você criou na etapa anterior.

Tópicos

- Criar uma tabela com um índice secundário global
- Descrever uma tabela com um índice secundário global
- Consultar um índice secundário global

- Exemplo: índices secundários globais que usam a API de baixo nível do AWS SDK para .NET

Criar uma tabela com um índice secundário global

Você pode criar índices secundários globais ao mesmo tempo em que cria uma tabela. Para fazer isso, use `CreateTable` e forneça suas especificações para um ou mais índices secundários globais. O exemplo de código C#a seguir cria uma tabela para armazenar informações sobre dados climáticos. A chave de partição é `Location` e a chave de classificação é `Date`. Um índice secundário global chamado `PrecipIndex` permite acesso rápido aos dados de precipitação de vários locais.

Veja a seguir as etapas necessárias para criar uma tabela com um índice secundário global usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Crie uma instância da classe `CreateTableRequest` para fornecer as informações solicitadas.

   Você deve fornecer o nome da tabela, sua chave primária e os valores de throughput provisionado. Para o índice secundário global, você deve fornecer o nome do índice, suas configurações de throughput provisionado, as definições de atributo da chave de classificação do índice, o esquema de chaves do índice e a projeção do atributo.
3. Execute o método `CreateTable` fornecendo o objeto de solicitação como um parâmetro.

O exemplo de código C# a seguir demonstra as etapas anteriores. O código cria uma tabela (`WeatherData`) com um índice secundário global (`PrecipIndex`). A chave de partição do índice é `Date` e a chave de classificação é `Precipitation`. Todos os atributos da tabela estão projetados no índice. Os usuários podem consultar esse índice para obter dados climáticos de uma data específica, opcionalmente, classificar os dados por quantidade de precipitação.

Como `Precipitation` não é um atributo de chave para a tabela, ele não é necessário. No entanto, os itens de `WeatherData` sem `Precipitation` não aparecem no `PrecipIndex`.

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

// Attribute definitions
var attributeDefinitions = new List<AttributeDefinition>()
{
    {new AttributeDefinition{
```

```
        AttributeName = "Location",
        AttributeType = "S"}},
    {new AttributeDefinition{
        AttributeName = "Date",
        AttributeType = "S"}},
    {new AttributeDefinition(){
        AttributeName = "Precipitation",
        AttributeType = "N"}
    }
};

// Table key schema
var tableKeySchema = new List<KeySchemaElement>()
{
    {new KeySchemaElement {
        AttributeName = "Location",
        KeyType = "HASH"}},  //Partition key
    {new KeySchemaElement {
        AttributeName = "Date",
        KeyType = "RANGE"}  //Sort key
    }
};

// PrecipIndex
var precipIndex = new GlobalSecondaryIndex
{
    IndexName = "PrecipIndex",
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)10,
        WriteCapacityUnits = (long)1
    },
    Projection = new Projection { ProjectionType = "ALL" }
};

var indexKeySchema = new List<KeySchemaElement> {
    {new KeySchemaElement { AttributeName = "Date", KeyType = "HASH"}},  //Partition
 key
    {new KeySchemaElement{AttributeName = "Precipitation",KeyType = "RANGE"}}  //Sort
 key
};

precipIndex.KeySchema = indexKeySchema;
```

```
CreateTableRequest createTableRequest = new CreateTableRequest
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)5,
        WriteCapacityUnits = (long)1
    },
    AttributeDefinitions = attributeDefinitions,
    KeySchema = tableKeySchema,
    GlobalSecondaryIndexes = { precipIndex }
};

CreateTableResponse response = client.CreateTable(createTableRequest);
Console.WriteLine(response.CreateTableResult.TableDescription.TableName);
Console.WriteLine(response.CreateTableResult.TableDescription.TableStatus);
```

Você deve aguardar até que o DynamoDB crie a tabela e defina o status dessa tabela como `ACTIVE`. Depois disso, você poderá começar a inserir itens de dados na tabela.

Descrever uma tabela com um índice secundário global

Para obter mais informações sobre índices secundários globais em uma tabela, use `DescribeTable`. Para cada índice, você pode acessar seu nome, esquema de chaves e atributos projetados.

Veja a seguir as etapas para acessar informações do índice secundário global para uma tabela usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Execute o método `describeTable` fornecendo o objeto de solicitação como um parâmetro.

   Crie uma instância da classe `DescribeTableRequest` para fornecer as informações solicitadas. Você deve fornecer o nome da tabela.
3.

O exemplo de código C# a seguir demonstra as etapas anteriores.

Example

```
client = new AmazonDynamoDBClient();
```

```
string tableName = "WeatherData";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest
 { TableName = tableName});

List<GlobalSecondaryIndexDescription> globalSecondaryIndexes =
response.DescribeTableResult.Table.GlobalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

foreach (GlobalSecondaryIndexDescription gsiDescription in globalSecondaryIndexes) {
     Console.WriteLine("Info for index " + gsiDescription.IndexName + ":");

     foreach (KeySchemaElement kse in gsiDescription.KeySchema) {
          Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
     }

      Projection projection = gsiDescription.Projection;
      Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

      if (projection.ProjectionType.ToString().Equals("INCLUDE")) {
           Console.WriteLine("\t\tThe non-key projected attributes are: "
                + projection.NonKeyAttributes);
      }
}
```

Consultar um índice secundário global

Você pode usar `Query` em um índice secundário global de forma semelhante ao uso de `Query` em uma tabela. Você precisa especificar o nome do índice, os critérios de consulta da chave de partição e da chave de classificação (se houver) do índice, e os atributos que você deseja retornar. Neste exemplo, o índice é `PrecipIndex`, que tem uma chave de partição `Date` e uma chave de classificação `Precipitation`. A consulta de índice retorna todos os dados climáticos de uma data específica, na qual a precipitação é maior que zero.

Veja a seguir as etapas para consultar um índice secundário global usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Crie uma instância da classe `QueryRequest` para fornecer as informações solicitadas.
3. Execute o método `query` fornecendo o objeto de solicitação como um parâmetro.

O nome do atributo `Date` é uma palavra reservada do DynamoDB. Portanto, use um nome de atributo de expressão como um espaço reservado na `KeyConditionExpression`.

O exemplo de código C# a seguir demonstra as etapas anteriores.

Example

```
client = new AmazonDynamoDBClient();

QueryRequest queryRequest = new QueryRequest
{
    TableName = "WeatherData",
    IndexName = "PrecipIndex",
    KeyConditionExpression = "#dt = :v_date and Precipitation > :v_precip",
    ExpressionAttributeNames = new Dictionary<String, String> {
        {"#dt", "Date"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_date", new AttributeValue { S =  "2013-08-01" }},
        {":v_precip", new AttributeValue { N =  "0" }}
    },
    ScanIndexForward = true
};

var result = client.Query(queryRequest);

var items = result.Items;
foreach (var currentItem in items)
{
    foreach (string attr in currentItem.Keys)
    {
        Console.Write(attr + "---> ");
        if (attr == "Precipitation")
        {
            Console.WriteLine(currentItem[attr].N);
    }
    else
    {
        Console.WriteLine(currentItem[attr].S);
    }

         }
     Console.WriteLine();
```

```
}
```

Exemplo: índices secundários globais que usam a API de baixo nível do AWS SDK para .NET

O código de exemplo Java a seguir mostra como trabalhar com índices secundários globais. O exemplo cria uma tabela chamada `Issues`, que pode ser usada em um sistema de controle de bugs simples para desenvolvimento de software. A chave de partição é `IssueId` e a chave de classificação é `Title`. Há três índices secundários globais nessa tabela:

- `CreateDateIndex`: a chave de partição é `CreateDate` e a chave de classificação é `IssueId`. Além das chaves da tabela, os atributos `Description` e `Status` são projetados no índice.
- `TitleIndex`: a chave de partição é `Title` e a chave de classificação é `IssueId`. Nenhum outro atributo além das chaves da tabela são projetos no índice.
- `DueDateIndex`: a chave de partição é `DueDate` e não há chave de classificação. Todos os atributos da tabela estão projetados no índice.

Depois que a tabela `Issues` é criada, o programa carrega a tabela com os dados que representam relatórios de bugs do software. Ele consulta os dados usando os índices secundários globais. Por fim, o programa exclui a tabela `Issues`.

Para obter instruções detalhadas sobre como testar o exemplo a seguir, consulte Exemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelGlobalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
        public static String tableName = "Issues";

        public static void Main(string[] args)
        {
            CreateTable();
            LoadData();

            QueryIndex("CreateDateIndex");
            QueryIndex("TitleIndex");
            QueryIndex("DueDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press enter");
            Console.Read();
        }

        private static void CreateTable()
        {
            // Attribute definitions
            var attributeDefinitions = new List<AttributeDefinition>()
        {
            {new AttributeDefinition {
                 AttributeName = "IssueId", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "Title", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "CreateDate", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "DueDate", AttributeType = "S"
             }}
        };

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>() {
            {
                new KeySchemaElement {
                    AttributeName= "IssueId",
                    KeyType = "HASH" //Partition key
                }
            },
```

```
            {
                new KeySchemaElement {
                    AttributeName = "Title",
                    KeyType = "RANGE" //Sort key
                }
            }
        };

            // Initial provisioned throughput settings for the indexes
            var ptIndex = new ProvisionedThroughput
            {
                ReadCapacityUnits = 1L,
                WriteCapacityUnits = 1L
            };

            // CreateDateIndex
            var createDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "CreateDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "CreateDate", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "INCLUDE",
                    NonKeyAttributes = {
                    "Description", "Status"
                }
                }
            };

            // TitleIndex
            var titleIndex = new GlobalSecondaryIndex()
            {
                IndexName = "TitleIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
```

```
                AttributeName = "Title", KeyType = "HASH" //Partition key
            },
            new KeySchemaElement {
                AttributeName = "IssueId", KeyType = "RANGE" //Sort key
            }
        },
            Projection = new Projection
            {
                ProjectionType = "KEYS_ONLY"
            }
        };

        // DueDateIndex
        var dueDateIndex = new GlobalSecondaryIndex()
        {
            IndexName = "DueDateIndex",
            ProvisionedThroughput = ptIndex,
            KeySchema = {
            new KeySchemaElement {
                AttributeName = "DueDate",
                KeyType = "HASH" //Partition key
            }
        },
            Projection = new Projection
            {
                ProjectionType = "ALL"
            }
        };



        var createTableRequest = new CreateTableRequest
        {
            TableName = tableName,
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = (long)1,
                WriteCapacityUnits = (long)1
            },
            AttributeDefinitions = attributeDefinitions,
            KeySchema = tableKeySchema,
            GlobalSecondaryIndexes = {
            createDateIndex, titleIndex, dueDateIndex
        }
```

```
            };

            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);

            WaitUntilTableReady(tableName);
        }

        private static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            // IssueId, Title,
            // Description,
            // CreateDate, LastUpdateDate, DueDate,
            // Priority, Status

            putItem("A-101", "Compilation error",
                "Can't compile Project X - bad version number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "Assigned");

            putItem("A-102", "Can't read data file",
                "The main data file is missing, or the permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30",
                2, "In progress");

            putItem("A-103", "Test failure",
                "Functional test of Project X produces errors",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "In progress");

            putItem("A-104", "Compilation error",
                "Variable 'messageCount' was not initialized.",
                "2013-11-15", "2013-11-16", "2013-11-30",
                3, "Assigned");

            putItem("A-105", "Network issue",
                "Can't ping IP address 127.0.0.1. Please fix this.",
                "2013-11-15", "2013-11-16", "2013-11-19",
                5, "Assigned");
        }

        private static void putItem(
```

```
            String issueId, String title,
            String description,
            String createDate, String lastUpdateDate, String dueDate,
            Int32 priority, String status)
        {
            Dictionary<String, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item.Add("IssueId", new AttributeValue
            {
                S = issueId
            });
            item.Add("Title", new AttributeValue
            {
                S = title
            });
            item.Add("Description", new AttributeValue
            {
                S = description
            });
            item.Add("CreateDate", new AttributeValue
            {
                S = createDate
            });
            item.Add("LastUpdateDate", new AttributeValue
            {
                S = lastUpdateDate
            });
            item.Add("DueDate", new AttributeValue
            {
                S = dueDate
            });
            item.Add("Priority", new AttributeValue
            {
                N = priority.ToString()
            });
            item.Add("Status", new AttributeValue
            {
                S = status
            });

            try
            {
                client.PutItem(new PutItemRequest
```

```
                {
                    TableName = tableName,
                    Item = item
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }

        private static void QueryIndex(string indexName)
        {
            Console.WriteLine
                ("\n*****************************************************\n");
            Console.WriteLine("Querying index " + indexName + "...");

            QueryRequest queryRequest = new QueryRequest
            {
                TableName = tableName,
                IndexName = indexName,
                ScanIndexForward = true
            };


            String keyConditionExpression;
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue>();

            if (indexName == "CreateDateIndex")
            {
                Console.WriteLine("Issues filed on 2013-11-01\n");

                keyConditionExpression = "CreateDate = :v_date and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-01"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });
            }
```

```
            else if (indexName == "TitleIndex")
            {
                Console.WriteLine("Compilation errors\n");

                keyConditionExpression = "Title = :v_title and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_title", new AttributeValue
                {
                    S = "Compilation error"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else if (indexName == "DueDateIndex")
            {
                Console.WriteLine("Items that are due on 2013-11-30\n");

                keyConditionExpression = "DueDate = :v_date";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-30"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo valid index name provided");
                return;
            }

            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
```

```
            foreach (string attr in currentItem.Keys)
            {
                if (attr == "Priority")
                {
                    Console.WriteLine(attr + "---> " + currentItem[attr].N);
                }
                else
                {
                    Console.WriteLine(attr + "---> " + currentItem[attr].S);
                }
            }
            Console.WriteLine();
        }
    }

    private static void DeleteTable(string tableName)
    {
        Console.WriteLine("Deleting table " + tableName + "...");
        client.DeleteTable(new DeleteTableRequest
        {
            TableName = tableName
        });
        WaitForTableToBeDeleted(tableName);
    }

    private static void WaitUntilTableReady(string tableName)
    {
        string status = null;
        // Let us wait until table is created. Call DescribeTable.
        do
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
                    TableName = tableName
                });

                Console.WriteLine("Table name: {0}, status: {1}",
                          res.Table.TableName,
                          res.Table.TableStatus);
                status = res.Table.TableStatus;
            }
```

```
                catch (ResourceNotFoundException)
                {
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Trabalhar com índices secundários globais no DynamoDB usando a AWS CLI

Você pode usar a AWS CLI para criar uma tabela do Amazon DynamoDB com um ou mais índices secundários globais na tabela e executar consultas usando os índices.

Tópicos

- Criar uma tabela com um índice secundário global

- Adicionar um índice secundário global a uma tabela existente
- Descrever uma tabela com um índice secundário global
- Consultar um índice secundário global

Criar uma tabela com um índice secundário global

Você pode criar índices secundários globais ao mesmo tempo que cria uma tabela. Para fazer isso, use o parâmetro `create-table` e forneça suas especificações para um ou mais índices secundários globais. O exemplo a seguir cria uma tabela chamada `GameScores` com um índice secundário global chamado `GameTitleIndex`. A tabela-base tem uma chave de partição `UserId` e uma chave de classificação `GameTitle`, permitindo que você encontre a melhor pontuação de um usuário individual para um jogo específico de forma eficiente, enquanto o GSI tem uma chave de partição `GameTitle` e uma chave de classificação `TopScore`, permitindo que você encontre rapidamente a pontuação mais alta geral para um jogo específico.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Você deve aguardar até que o DynamoDB crie a tabela e defina o status dessa tabela como `ACTIVE`. Depois disso, você poderá começar a inserir itens de dados na tabela. Você pode usar describe-table para determinar o status da criação da tabela.

Adicionar um índice secundário global a uma tabela existente

Os índices secundários globais também podem ser adicionados ou modificados após a criação da tabela. Para fazer isso, use o parâmetro `update-table` e forneça suas especificações para um ou mais índices secundários globais. O exemplo a seguir usa o mesmo esquema do exemplo anterior, mas pressupõe que a tabela já foi criada e que adicionaremos o GSI mais tarde.

```
aws dynamodb update-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=TopScore,AttributeType=N  \
    --global-secondary-index-updates \
        "[
            {
                \"Create\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                    \"Projection\":{
                        \"ProjectionType\":\"INCLUDE\",
                        \"NonKeyAttributes\":[\"UserId\"]
                    }
                }
            }
        ]"
```

Descrever uma tabela com um índice secundário global

Para obter mais informações sobre índices secundários globais em uma tabela, use o parâmetro `describe-table`. Para cada índice, você pode acessar seu nome, esquema de chaves e atributos projetados.

```
aws dynamodb describe-table --table-name GameScores
```

Consultar um índice secundário global

Você pode usar a operação `query` em um índice secundário global de forma semelhante ao uso de `query` em uma tabela. Você deve especificar o nome do índice, os critérios de consulta da chave de classificação do índice e os atributos que deseja retornar. Neste exemplo, o índice é `GameTitleIndex` e a chave de classificação do índice é `GameTitle`.

Os únicos atributos retornados são aqueles que foram projetados no índice. É possível modificar essa consulta para selecionar atributos não chave também, mas isso exigiria atividades de busca de tabela que são relativamente caras. Para obter mais informações sobre buscas de tabela, consulte Projeções de atributo.

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

Índices secundários locais no DynamoDB

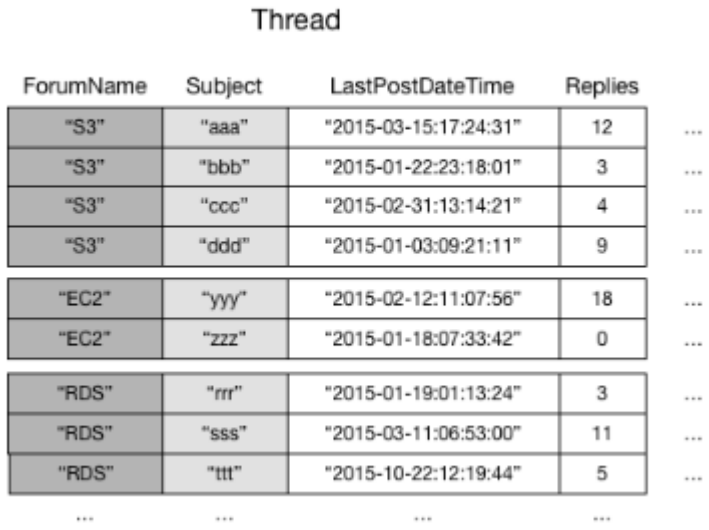
Alguns aplicativos só precisam consultar dados usando a chave primária da tabela base. No entanto, podem haver situações em que uma chave de classificação alternativa seria útil. Para oferecer à sua aplicação opções de chaves de classificação, você pode criar um ou mais índices secundários locais em uma tabela do Amazon DynamoDB e emitir solicitações de `Query` ou `Scan` nesses índices.

Tópicos

- Cenário: Uso de um índice secundário local
- Projeções de atributo
- Criação de um índice secundário local
- Ler dados de um índice secundário local
- Gravações de itens e índices secundários locais
- Considerações sobre throughput provisionado para índices secundários locais
- Considerações sobre armazenamento para índices secundários locais
- Conjuntos de itens em índices secundários locais
- Como trabalhar com índices secundários locais: Java
- Como trabalhar com índices secundários locais: .NET
- Trabalhar com índices secundários locais no DynamoDB usando a AWS CLI

Cenário: Uso de um índice secundário local

Como exemplo, considere a tabela `Thread`. Esta tabela é útil para aplicações como os Fóruns de discussão da AWS. O diagrama a seguir mostra como os itens da tabela seriam organizados. (Nem todos os atributos são mostrados.)



O DynamoDB armazena continuamente todos os itens com o mesmo valor de chave de partição. Neste exemplo, dado um determinado `ForumName`, uma operação `Query` poderia localizar imediatamente todos os threads desse fórum. Em um grupo de itens com o mesmo valor de chave de partição, os itens são classificados pelo valor de chave de classificação. Se a chave de classificação (`Subject`) também for fornecida na consulta, o DynamoDB poderá reduzir os resultados que são retornados. Por exemplo, retornar todos os threads do fórum "S3" em que `Subject` começa com a letra "a".

Algumas solicitações podem exigir padrões mais complexos de acesso aos dados. Por exemplo:

- Quais threads do fórum recebem visualizações e respostas?
- Qual thread em um determinado fórum tem o maior número de mensagens?
- Quantos threads foram publicados em um fórum específico em um determinado período?

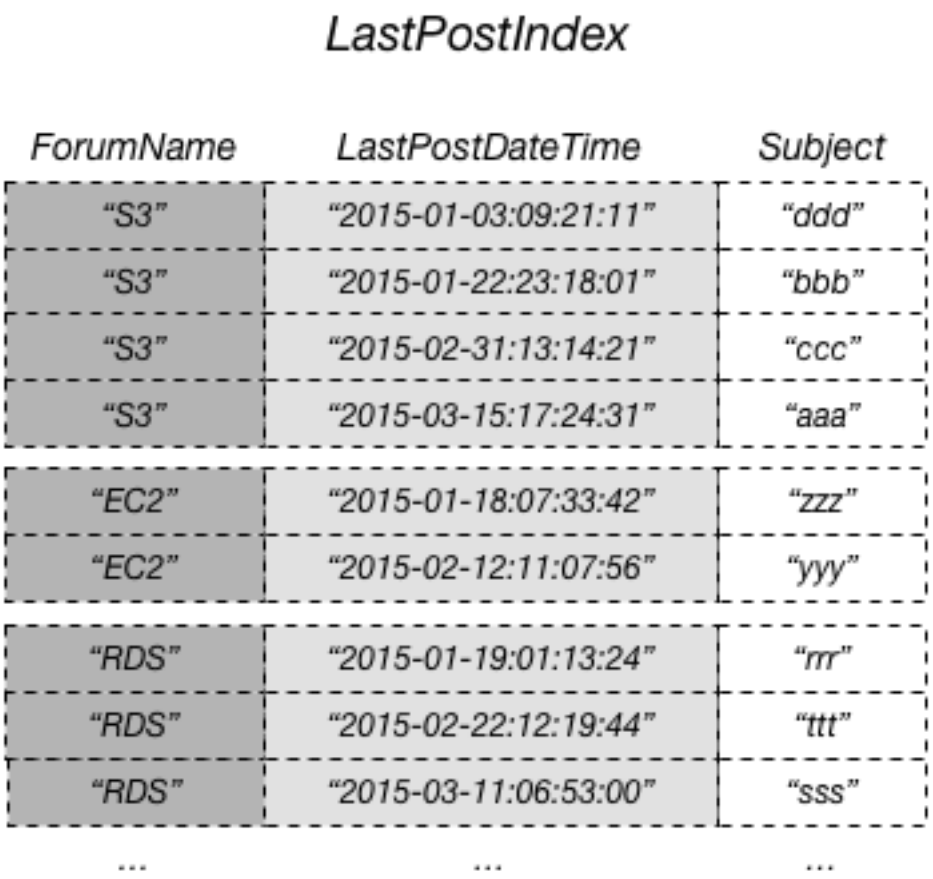
A ação `Query` não seria suficiente para responder a essas perguntas. Em vez disso, você teria de `Scan` toda a tabela. Para uma tabela com milhões de itens, isso poderia consumir uma grande quantidade de throughput provisionado de leitura e levar muito tempo para ser concluído.

No entanto, é possível especificar um ou mais índices secundários em atributos que não são chaves, como `Replies` ou `LastPostDateTime`.

Um índice secundário local mantém uma chave de classificação alternativa para um determinado valor de chave de partição. Um índice secundário local também contém uma cópia de alguns ou de todos os atributos de sua tabela-base. Você especifica quais atributos são projetados no índice secundário local ao criar a tabela. Os dados em um índice secundário local são organizados com a mesma chave de partição que a tabela-base, mas com outra chave de classificação. Isso permite que você acesse os itens de dados de forma eficiente nessa dimensão diferente. Para uma maior flexibilidade de consulta ou verificação, você pode criar até cinco índices secundários locais por tabela.

Suponha que um aplicativo precise encontrar todos os threads que foram publicados nos últimos três meses em um fórum específico. Sem um índice secundário local, a aplicação precisaria realizar `Scan` de toda a tabela `Thread` e descartar todas as publicações que não estivessem dentro do período especificado. Com um índice secundário local, uma operação de `Query` poderia usar `LastPostDateTime` como uma chave de classificação e encontrar os dados rapidamente.

O diagrama a seguir mostra um índice secundário local chamado `LastPostIndex`. Observe que a chave de partição é a mesma que a da tabela `Thread`, mas a chave de classificação é `LastPostDateTime`.



Cada índice secundário local deve atender às seguintes condições:

- A chave de partição é a mesma que a de sua tabela-base.
- A chave de classificação consiste em exatamente um atributo escalar.
- A chave de classificação da tabela-base é projetada no índice, onde ela atua como um atributo que não é chave.

Neste exemplo, a chave de partição é `ForumName` e a chave de classificação do índice secundário local é `LastPostDateTime`. Além disso, o valor da chave de classificação da tabela base (neste exemplo, `Subject`) é projetado no índice, mas não faz parte da chave do índice. Se um aplicativo precisar de uma lista baseada em `ForumName` e `LastPostDateTime`, ele poderá emitir uma solicitação de `Query` com relação a `LastPostIndex`. Os resultados da consulta são classificados por `LastPostDateTime` e podem ser retornados em ordem crescente ou decrescente. A consulta também pode aplicar condições de chave, como retornar apenas os itens que têm uma `LastPostDateTime` dentro de um período específico.

Cada índice secundário local contém automaticamente as chave de partição e de classificação de sua tabela-base, mas você pode opcionalmente projetar atributos não chave no índice. Quando você consultar o índice, o DynamoDB poderá recuperar esses atributos projetados com eficiência. Quando você consulta um índice secundário local, a consulta também pode recuperar atributos que não são projetados no índice. O DynamoDB buscará automaticamente esses atributos na tabela-base, mas com uma latência maior e com custos de throughput provisionado mais altos.

Para qualquer índice secundário local, você pode armazenar até 10 GB de dados por valor de chave de partição distinta. Esta imagem inclui todos os itens na tabela-base, além de todos os itens nos índices, que têm o mesmo valor de chave de partição. Para ter mais informações, consulte Conjuntos de itens em índices secundários locais.

Projeções de atributo

Com `LastPostIndex`, um aplicativo poderia usar `ForumName` e `LastPostDateTime` como critérios de consulta. No entanto, para recuperar qualquer atributo adicional, o DynamoDB deve executar operações de leitura adicionais na tabela `Thread`. Essas leituras extras são conhecidas como buscas e podem aumentar a quantidade total de throughput provisionado necessário para uma consulta.

Suponha que você quisesse preencher uma página da Web com uma lista de todos os threads de "S3" e o número de respostas para cada thread, classificados pela última data/hora de resposta, começando pela resposta mais recente. Para preencher essa lista, você precisaria dos seguintes atributos:

- `Subject`
- `Replies`
- `LastPostDateTime`

A maneira mais eficiente de consultar esses dados e evitar operações de busca seria projetar o atributo `Replies` da tabela no índice secundário local conforme mostrado neste diagrama.

LastPostIndex

ForumName	LastPostDateTime	Subject	Replies
"S3"	"2015-01-03:09:21:11"	"ddd"	9
"S3"	"2015-01-22:23:18:01"	"bbb"	3
"S3"	"2015-02-31:13:14:21"	"ccc"	4
"S3"	"2015-03-15:17:24:31"	"aaa"	12
"EC2"	"2015-01-18:07:33:42"	"zzz"	0
"EC2"	"2015-02-12:11:07:56"	"yyy"	18
"RDS"	"2015-01-19:01:13:24"	"rrr"	3
"RDS"	"2015-02-22:12:19:44"	"ttt"	5
"RDS"	"2015-03-11:06:53:00"	"sss"	11
...	...	...	...

Uma projeção é o conjunto de atributos que é copiado de uma tabela para um índice secundário. A chave de partição e a chave de classificação da tabela são sempre projetadas no índice; você pode projetar outros atributos para suportar os requisitos de consulta da sua aplicação. Quando você consulta um índice, o Amazon DynamoDB pode acessar quaisquer atributos na projeção como se estivessem em uma tabela própria.

Quando você cria um índice secundário, é necessário especificar os atributos que serão projetados no índice. O DynamoDB proporciona três opções diferentes para fazer isso:

- KEYS_ONLY: cada item do índice consiste apenas nos valores de chaves de partição e nas chaves de classificação da tabela, além dos valores de chaves do índice. A opção `KEYS_ONLY` resulta no menor índice secundário possível.
- INCLUDE: além dos atributos descritos em `KEYS_ONLY`, o índice secundário incluirá outros atributos não chave que você especificar.
- ALL: o índice secundário inclui todos os atributos da tabela de origem. Como todos os dados da tabela são duplicados no índice, uma projeção `ALL` resulta no maior índice secundário possível.

No diagrama anterior, o atributo `Replies` que não é chave é projetado em `LastPostIndex`. Um aplicativo pode consultar `LastPostIndex` em vez da tabela `Thread` completa para preencher uma página da Web com `Subject`, `Replies` e `LastPostDateTime`. Se quaisquer outros atributos que não são chaves forem solicitados, o DynamoDB precisará buscar esses atributos na tabela `Thread`.

Do ponto de vista de um aplicativo, buscar atributos adicionais da tabela-base é automático e transparente, portanto, não há necessidade de reescrever qualquer lógica de aplicativo. No entanto, essa busca pode reduzir significativamente a vantagem de performance proporcionada pelo uso de um índice secundário local.

Ao escolher os atributos para projetar em um índice secundário local, você deve considerar a desvantagem entre os custos de throughput provisionado e os custos de armazenamento:

- Se você precisar acessar apenas alguns atributos com a latência mais baixa possível, considere projetar apenas os atributos em um índice secundário local. Quanto menor o índice, menores serão os custos de armazenamento e de gravação. Se houver atributos que você precisa buscar ocasionalmente, o custo de throughput provisionado pode ultrapassar o custo por um prazo mais longo do armazenamento desses atributos.
- Se sua aplicação acessar frequentemente alguns atributos não chave, considere projetar esses atributos em um índice secundário local. Os custos adicionais de armazenamento do índice secundário local compensarão o custo de executar verificações de tabelas frequentes.
- Se precisar acessar a maioria dos atributos não chave com frequência, você poderá projetar esses atributos, inclusive a tabela-base inteira, em um índice secundário local. Isso fornece flexibilidade máxima e menor consumo de throughput provisionado, porque nenhuma busca será necessária. No entanto, o custo do armazenamento deve aumentar ou até dobrar se você estiver projetando todos os atributos.
- Se o seu aplicativo precisa consultar uma tabela com pouca frequência, mas deve realizar muitas gravações ou atualizações nos dados na tabela, pense em projetar KEYS_ONLY. O índice secundário local seria de tamanho mínimo, mas ainda estaria disponível quando necessário para a atividade de consulta.

Criação de um índice secundário local

Para criar um ou mais índices secundários locais, use o parâmetro `LocalSecondaryIndexes` da operação `CreateTable`. Os índices secundários locais em uma tabela são criados quando a tabela é criada. Quando você exclui uma tabela, todos os índices secundários locais da tabela também são excluídos.

Você deve especificar um atributo não chave para atuar como a chave de classificação do índice secundário local. O atributo escolhido deve ser de um tipo escalar como `String`, `Number` ou `Binary` Outros tipos escalares, tipos de documento e tipos de conjunto não são permitidos. Para obter uma lista completa de tipos de dados, consulte Tipos de dados.

> **Important**
>
> Para tabelas com índices secundários locais, há um limite de tamanho de 10 GB para cada valor de chave de partição. Uma tabela com índices secundários locais pode armazenar qualquer número de itens, desde que o tamanho total de qualquer valor de chave de partição não exceda 10 GB. Para ter mais informações, consulte Limite de tamanho de conjunto de itens.

Você pode projetar atributos de qualquer tipo de dados em um índice secundário local. Isso inclui escalares, documentos e conjuntos. Para obter uma lista completa de tipos de dados, consulte Tipos de dados.

Ler dados de um índice secundário local

Você pode recuperar itens de um índice secundário local usando as operações `Query` e `Scan`. As operações `GetItem` e `BatchGetItem` não podem ser usadas em um índice secundário local.

Como consultar um índice secundário local

Em uma tabela do DynamoDB, o valor da chave de partição combinada e o valor da chave de classificação de cada item devem ser exclusivos. No entanto, em um índice secundário local, o valor da chave de classificação não precisa ser exclusivo para um determinado valor de chave de partição. Se houver vários itens no índice secundário local com o mesmo valor de chave de classificação, uma operação `Query` retornará todos os itens que têm o mesmo valor de chave de partição. Na resposta, os itens correspondentes não são retornados em uma ordem específica.

Você pode consultar um índice secundário local usando leituras fortemente consistentes ou finais consistentes. Para especificar qual tipo de consistência você deseja, use o parâmetro `ConsistentRead` da operação `Query`. Uma leitura fortemente consistente de um índice secundário local sempre retorna os valores atualizados mais recentes. Se a consulta precisar buscar atributos adicionais na tabela base, esses atributos serão consistentes com relação ao índice.

Example

Considere os seguintes dados retornados de uma `Query` que solicita dados dos threads de discussão em um determinado fórum.

```
{
    "TableName": "Thread",
    "IndexName": "LastPostIndex",
    "ConsistentRead": false,
    "ProjectionExpression": "Subject, LastPostDateTime, Replies, Tags",
    "KeyConditionExpression":
        "ForumName = :v_forum and LastPostDateTime between :v_start and :v_end",
    "ExpressionAttributeValues": {
        ":v_start": {"S": "2015-08-31T00:00:00.000Z"},
        ":v_end": {"S": "2015-11-31T00:00:00.000Z"},
        ":v_forum": {"S": "EC2"}
    }
}
```

Nesta consulta:

- O DynamoDB acessa `LastPostIndex` usando a chave de partição `ForumName` para localizar os itens do índice para o "EC2". Todos os itens do índice com essa chave são armazenados lado a lado para rápida recuperação.
- Neste fórum, o DynamoDB usa o índice para pesquisar as chaves que correspondem à condição `LastPostDateTime` especificada.
- Como o atributo `Replies` é projetado no índice, o DynamoDB pode recuperar esse atributo sem consumir nenhum throughput provisionado adicional.
- O atributo `Tags` não é projetado no índice, portanto, o DynamoDB deve acessar a tabela `Thread` e buscar esse atributo.
- Os resultados são retornados, classificados por `LastPostDateTime`. As entradas de índice são classificadas por valor de chave de partição e, em seguida, pelo valor de chave de classificação, e `Query` retorna-as na ordem em que são armazenadas. (Você pode usar o parâmetro `ScanIndexForward` para retornar os resultados em ordem decrescente.)

Como o atributo `Tags` não é projetado no índice secundário local, o DynamoDB deve consumir unidades de capacidade de leitura adicionais para buscar esse atributo na tabela-base. Se for necessário executar essa consulta com frequência, projete `Tags` no `LastPostIndex` para evitar a

busca na tabela base. No entanto, se for necessário acessar `Tags` apenas ocasionalmente, o custo do armazenamento adicional para a projeção de `Tags` no índice pode não valer a pena.

Verificação de um índice secundário local

Você pode usar `Scan` para recuperar todos os dados de um índice secundário local. Você deve fornecer o nome da tabela-base e o nome de índice na solicitação. Com uma operação `Scan`, o DynamoDB lê todos os dados do índice e os retorna para a aplicação. Você também pode solicitar que apenas alguns dos dados sejam retornados, e que os dados restantes sejam descartados. Para fazer isso, use o parâmetro `FilterExpression` da API `Scan`. Para ter mais informações, consulte Expressões de filtro para verificação.

Gravações de itens e índices secundários locais

O DynamoDB mantém automaticamente todos os índices secundários locais sincronizados com suas respectivas tabelas base. Os aplicativos nunca gravam diretamente em um índice. No entanto, é importante compreender as implicações de como o DynamoDB mantém esses índices.

Ao criar um índice secundário local, você especifica um atributo para servir como a chave de classificação do índice. Você também especifica um tipo de dados desse atributo. Isso significa que sempre que você grava um item na tabela-base, se o item define um atributo de chave do índice, seu tipo deve corresponder ao tipo de dados do esquema de chaves do índice. No caso de `LastPostIndex`, a chave de classificação de `LastPostDateTime` no índice é definida como um tipo de dados `String`. Se você tentar adicionar um item à tabela `Thread` e especificar um tipo de dados diferente para `LastPostDateTime` (como `Number`), o DynamoDB retornará uma `ValidationException` devido à inconsistência do tipo de dados.

Não há necessidade de um relacionamento de um para um entre os itens em uma tabela-base e os itens em um índice secundário local. Na verdade, esse comportamento pode ser vantajoso para muitas aplicações.

Os custos das atividades de gravação em uma tabela com muitos índices secundários serão mais altos do que em uma tabela com um número menor de índices. Para ter mais informações, consulte Considerações sobre throughput provisionado para índices secundários locais.

> **Important**
>
> Para tabelas com índices secundários locais, há um limite de tamanho de 10 GB para cada valor de chave de partição. Uma tabela com índices secundários locais pode armazenar qualquer número de itens, desde que o tamanho total de qualquer valor de chave de partição

não exceda 10 GB. Para ter mais informações, consulte Limite de tamanho de conjunto de itens.

Considerações sobre throughput provisionado para índices secundários locais

Ao criar uma tabela no DynamoDB, você provisiona unidades de capacidade de leitura e gravação para a workload esperada na tabela. Essa workload inclui a atividade de leitura e gravação nos índices secundários locais da tabela.

Para visualizar as taxas atuais da capacidade de throughput provisionado, acesse Preços do Amazon DynamoDB.

Unidades de capacidade de leitura

Quando você consulta um índice secundário local, o número de unidades de capacidade de leitura consumidas depende de como os dados são acessados.

Assim como ocorre com as consultas de tabela, uma consulta de índice pode usar leituras fortemente consistentes ou eventualmente consistentes, dependendo do valor de `ConsistentRead`. Uma leitura fortemente consistente consome uma unidade de capacidade de leitura, uma leitura eventualmente consistente consome apenas metade disso. Assim, escolhendo leituras eventualmente consistentes, você pode reduzir seus encargos de unidade de capacidade de leitura.

Para consultas do índice que solicitam apenas chaves de índice e atributos projetados, o DynamoDB calcula a atividade de leitura provisionada da mesma forma que para consultas em tabelas. A única diferença é que o cálculo é baseado no tamanho das entradas de índice, em vez do tamanho do item na tabela-base. O número de unidades de capacidade de leitura é a soma de todos os tamanhos de atributos projetados em todos os itens retornados; o resultado é, então, arredondado para o próximo limite de 4 KB. Para obter mais informações sobre como o DynamoDB calcula a utilização de throughput provisionado, consulte Modo de capacidade provisionada do DynamoDB.

Para consultas de índice que leem atributos que não estão projetados no índice secundário local, o DynamoDB precisa buscar esses atributos na tabela-base, além de ler os atributos projetados no índice. Essas buscas ocorrem quando você inclui quaisquer atributos não projetados nos parâmetros `Select` ou `ProjectionExpression` da operação `Query`. A busca causa latência adicional nas respostas da consulta, e também incorre em um custo mais alto de throughput provisionado: além das leituras do índice secundário local descritas acima, você é cobrado pelas unidades de capacidade de leitura de cada item buscado na tabela-base. Essa cobrança é para ler cada item inteiro da tabela, não apenas os atributos solicitados.

O tamanho máximo dos resultados retornados por uma operação `Query` é 1 MB. Isso inclui os tamanhos de todos os nomes e valores de atributos de todos os itens retornados. No entanto, se uma consulta em um índice secundário local fizer o DynamoDB buscar atributos de item na tabela-base, o tamanho máximo dos dados no resultado poderá ser menor. Neste caso, o tamanho do resultado é a soma de:

- O tamanho dos itens correspondentes no índice, arredondado para os próximos 4 KB.
- O tamanho de cada item correspondente na tabela-base, com cada item individualmente arredondado para os próximos 4 KB.

Usando esta fórmula, o tamanho máximo dos resultados retornados por uma operação Query ainda é 1 MB.

Por exemplo, considere uma tabela na qual o tamanho de cada item é 300 bytes. Há um índice secundário local nessa tabela, mas apenas 200 bytes de cada item são projetados no índice. Agora, suponha que você use a operação `Query` nesse índice, que a consulta requer buscas de tabela para cada item e que a consulta retorne 4 itens. O DynamoDB soma o seguinte:

- O tamanho dos itens correspondentes no índice: 200 bytes × 4 itens = 800 bytes; isso é, então, arredondado para 4 KB.
- O tamanho de cada item correspondente na tabela-base: (300 bytes, arredondados para 4 KB) × 4 itens = 16 KB.

O tamanho total dos dados no resultado é, portanto, 20 KB.

Unidades de capacidade de gravação

Quando um item é adicionado, atualizado ou excluído de uma tabela, a atualização dos índices secundários locais consome unidades de capacidade de gravação provisionadas para a tabela. O custo total do throughput provisionado para uma gravação é a soma das unidades de capacidade de gravação consumidas pela gravação na tabela e aquelas consumidas pela atualização dos índices secundários locais.

O custo de gravar um item em um índice secundário local depende de vários fatores:

- Se você gravar um novo item na tabela que define um atributo indexado, ou atualizar um item existente para definir um atributo indexado indefinido anteriormente, uma operação de gravação é necessária para inserir o item no índice.

- Se uma atualização na tabela alterar o valor de um atributo de chave indexado (de A para B), duas gravações serão necessárias, uma para excluir o item anterior do índice e outra gravação para inserir o novo item no índice.
- Se um item estava presente no índice, mas uma gravação na tabela fez com que o atributo indexado fosse excluído, uma gravação é necessária para excluir a projeção do item antigo do índice.
- Se um item não estiver presente no índice antes ou depois que o item é atualizado, não haverá custo de gravação adicionais para o índice.

Todos esses fatores supõem que o tamanho de cada item no índice seja menor ou igual ao tamanho de item de 1 KB para calcular unidades de capacidade de gravação. Entradas de índice maiores exigirão unidades adicionais de capacidade de gravação. Você pode minimizar os custos de gravação considerando de quais atributos suas consultas precisam para retornar e projetar apenas esses atributos no índice.

Considerações sobre armazenamento para índices secundários locais

Quando uma aplicação grava um item em uma tabela, o DynamoDB copia automaticamente o subconjunto correto de atributos em todos os índices secundários locais nos quais esses atributos devem aparecer. Sua conta da AWS é cobrada pelo armazenamento do item na tabela-base e também pelo armazenamento dos atributos em todos os índices secundários locais nessa tabela.

A quantidade de espaço usada por um item do índice é a soma do seguinte:

- O tamanho em bytes da chave primária da tabela-base (chave de partição e chave de classificação)
- O tamanho em bytes do atributo de chave do índice
- O tamanho em bytes dos atributos projetados (se houver)
- 100 bytes de sobrecarga por item de índice

Para estimar os requisitos de armazenamento de um índice secundário local, você pode estimar o tamanho médio de um item no índice e multiplicar pelo número de itens no índice.

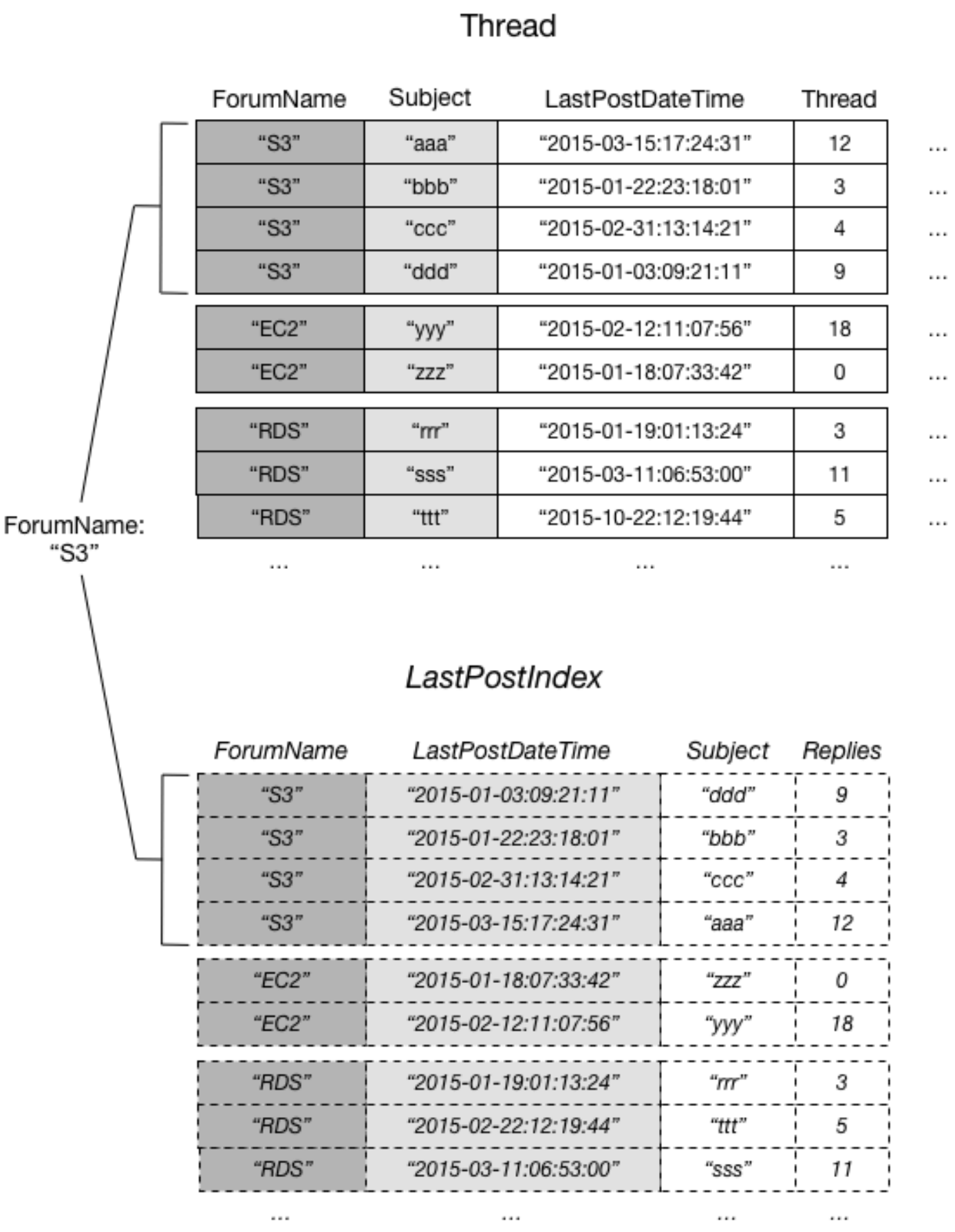
Se uma tabela contiver um item em que um determinado atributo não está definido, mas esse atributo estiver definido como uma chave de classificação do índice, o DynamoDB não gravará nenhum dado desse item no índice.

Conjuntos de itens em índices secundários locais

> **Note**
>
> A seção refere-se apenas a tabelas que têm índices secundários locais.

No DynamoDB, uma coleção de itens é qualquer grupo de itens que têm o mesmo valor de chave de partição em uma tabela e todos os seus índices secundários locais. Nos exemplos usados nesta seção, a chave de partição da tabela `Thread` é `ForumName`, e a chave de partição de `LastPostIndex` também é `ForumName`. Todos os itens da tabela e do índice com o mesmo `ForumName` fazem parte da mesma coleção de itens. Por exemplo, na tabela `Thread` e no índice secundário local `LastPostIndex`, existe uma coleção de itens para o fórum do `EC2` e uma coleção de itens diferente para o fórum do `RDS`.

O seguinte diagrama mostra a coleção de itens do fórum do `S3`.


Thread
ForumName
Subject
LastPostDateTime
Thread
"S3"
"aaa"
"2015-03-15:17:24:31"
12
"S3"
"bbb"
"2015-01-22:23:18:01"
3
"S3"
"ccc"
"2015-02-31:13:14:21"
4
"S3"
"ddd"
"2015-01-03:09:21:11"
9
"EC2"
"yyy"
"2015-02-12:11:07:56"
18
"EC2"
"zzz"
"2015-01-18:07:33:42"
0
"RDS"
"rrr"
"2015-01-19:01:13:24"
3
"RDS"
"sss"
"2015-03-11:06:53:00"
11
"RDS"
"ttt"
"2015-10-22:12:19:44"
5
ForumName:
"S3"
LastPostIndex
ForumName
LastPostDateTime
Subject
Replies
"S3"
"2015-01-03:09:21:11"
"ddd"
9
"S3"
"2015-01-22:23:18:01"
"bbb"
3
"S3"
"2015-02-31:13:14:21"
"ccc"
4
"S3"
"2015-03-15:17:24:31"
"aaa"
12
"EC2"
"2015-01-18:07:33:42"
"zzz"
0
"EC2"
"2015-02-12:11:07:56"
"yyy"
18
"RDS"
"2015-01-19:01:13:24"
"rrr"
3
"RDS"
"2015-02-22:12:19:44"
"ttt"
5
"RDS"
"2015-03-11:06:53:00"
"sss"
11

Neste diagrama, a coleção de itens consiste em todos os itens em `Thread` e `LastPostIndex` em que o valor da chave da partição `ForumName` é "S3". Se houver outros índices secundários locais na tabela, todos os itens nesses índices com `ForumName` igual a "S3" também farão parte da coleção de itens.

Você pode usar qualquer uma das operações a seguir no DynamoDB para retornar informações sobre coleções de itens:

- `BatchWriteItem`
- `DeleteItem`
- `PutItem`
- `UpdateItem`
- `TransactWriteItems`

Cada uma dessas operações é compatível com o parâmetro `ReturnItemCollectionMetrics`. Ao definir esse parâmetro como `SIZE`, você pode exibir informações sobre o tamanho de cada coleção de itens no índice.

Example

Este é um exemplo da saída de uma operação `UpdateItem` na tabela `Thread`, com `ReturnItemCollectionMetrics` definido como `SIZE`. O item que foi atualizado tinha um valor `ForumName` de "EC2", portanto, a saída inclui informações sobre essa coleção de itens.

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

O objeto `SizeEstimateRangeGB` mostra que o tamanho dessa coleção de itens está entre 0 e 1 GB. O DynamoDB atualiza periodicamente essa estimativa de tamanho, portanto, os números podem ser diferentes na próxima vez em que o item é modificado.

Limite de tamanho de conjunto de itens

O tamanho máximo de qualquer coleção de itens para uma tabela com um ou mais índices secundários locais é 10 GB. Isso não se aplica a coleções de itens em tabelas sem índices secundários locais e também não se aplica a coleções de itens em índices secundários globais. Apenas as tabelas que têm um ou mais índices secundários locais são afetadas.

Se uma coleção de itens exceder o limite de 10 GB, o DynamoDB retornará uma `ItemCollectionSizeLimitExceededException`, e você não poderá adicionar mais itens à coleção de itens ou aumentar os tamanhos dos itens que estão na coleção de itens. (As operações de leitura e gravação que diminuem o tamanho da coleção de itens ainda são permitidas.) Você ainda pode adicionar itens a outras coleções de itens.

Para reduzir o tamanho de uma coleção de itens, você pode executar uma das seguintes ações:

- Excluir todos os itens desnecessários com o valor da chave de partição em questão. Quando você exclui esses itens da tabela-base, o DynamoDB também remove todas as entradas do índice que têm o mesmo valor de chave de partição.
- Atualize os itens, removendo atributos ou reduzindo o tamanho dos atributos. Se esses atributos forem projetados em qualquer índice secundário local, o DynamoDB também reduzirá o tamanho das entradas do índice correspondentes.
- Crie uma nova tabela com a mesma chave de partição e chave de classificação e, em seguida, mova os itens da tabela antiga para a nova tabela. Essa pode ser uma boa abordagem, se uma tabela tiver dados históricos que são acessados com pouca frequência. Você também pode considerar arquivar esses dados históricos no Amazon Simple Storage Service (Amazon S3).

Quando o tamanho total da coleção de itens tornar-se inferior a 10 GB, você poderá adicionar itens novamente com o mesmo valor de chave de partição.

Recomendamos como uma melhor prática que você instrumente seu aplicativo para monitorar os tamanhos de suas coleções de itens. Uma maneira de fazer isso é definir o parâmetro `ReturnItemCollectionMetrics` como `SIZE` sempre que você usar `BatchWriteItem`, `DeleteItem`, `PutItem` ou `UpdateItem`. Seu aplicativo deve examinar o objeto `ReturnItemCollectionMetrics` na saída e gerar uma mensagem de erro sempre que uma coleção de itens exceder um limite definido pelo usuário (por exemplo, 8 GB). Configurar um limite menor que 10 GB oferece um sistema de aviso antecipado para que você saiba que uma coleção de itens está se aproximando do limite a tempo de resolver o problema.

Conjuntos de itens e partições

Em uma tabela com um ou mais índices secundários locais, uma coleção de itens é armazenada em uma partição. O tamanho total dessa coleção de itens é limitado à capacidade dessa partição: 10 GB. Para uma aplicação em que o modelo de dados inclui coleções de itens de tamanho ilimitado ou na qual a expectativa é de que algumas coleções de itens aumentem além de 10 GB no futuro, pense em usar um índice secundário global.

Você deve criar seus aplicativos para que os dados da tabela sejam distribuídos uniformemente entre diferentes valores de chave de partição. Para tabelas com índices secundários locais, seus aplicativos não devem criar pontos de atividade de leitura e gravação em uma única coleção de itens em uma única partição.

Como trabalhar com índices secundários locais: Java

Você pode usar a API de documentos do AWS SDK para Java para criar uma tabela do Amazon DynamoDB com um ou mais índices secundários locais na tabela e executar consultas usando os índices.

Veja a seguir as etapas comuns para operações de tabela usando a API de documento do AWS SDK para Java.

1. Crie uma instância da classe `DynamoDB`.
2. Forneça os parâmetros obrigatórios e opcionais para a operação, criando os objetos de solicitação correspondentes.
3. Chame o método apropriado fornecido pelo cliente que você criou na etapa anterior.

Tópicos

- Criar uma tabela com um índice secundário local
- Descrever uma tabela com um índice secundário local
- Consultar um índice secundário local
- Exemplo: índices secundários locais que usam a API de documentos do Java

Criar uma tabela com um índice secundário local

Os índices secundários locais devem ser criados ao mesmo tempo que uma tabela é criada. Para fazer isso, use o método `createTable` e forneça suas especificações para um ou mais índices secundários locais. O exemplo de código Java a seguir cria uma tabela para armazenar

informações sobre músicas em uma coleção de músicas. A chave de partição é `Artist` e a chave de classificação é `SongTitle`. Um índice secundário, `AlbumTitleIndex`, facilita consultas por título de álbum.

Veja a seguir as etapas necessárias para criar uma tabela com um índice secundário local usando a API de documentos do DynamoDB.

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `CreateTableRequest` para fornecer as informações solicitadas.

   Você deve fornecer o nome da tabela, sua chave primária e os valores de throughput provisionado. Para o índice secundário local, você deve fornecer o nome do índice, o nome e o tipo de dados da chave de classificação do índice, o esquema de chave para o índice e a projeção de atributos.
3. Chame o método `createTable`, fornecendo o objeto de solicitação como um parâmetro.

O exemplo de código Java a seguir demonstra as etapas anteriores. O código cria uma tabela (`Music`) com um índice secundário no atributo `AlbumTitle`. A chave de partição de tabela e a chave de classificação, bem como a chave de classificação de índice, são os únicos atributos projetados para o índice.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName);

//ProvisionedThroughput
createTableRequest.setProvisionedThroughput(new
 ProvisionedThroughput().withReadCapacityUnits((long)5).withWriteCapacityUnits((long)5));

//AttributeDefinitions
ArrayList<AttributeDefinition> attributeDefinitions= new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Artist").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("SongTitle").withAttributeType("S"));
```

```
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("AlbumTitle").withAttributeType("S"));

createTableRequest.setAttributeDefinitions(attributeDefinitions);

//KeySchema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE));  //Sort
 key

createTableRequest.setKeySchema(tableKeySchema);

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("AlbumTitle").withKeyType(KeyType.RANGE));  //
Sort key

Projection projection = new Projection().withProjectionType(ProjectionType.INCLUDE);
ArrayList<String> nonKeyAttributes = new ArrayList<String>();
nonKeyAttributes.add("Genre");
nonKeyAttributes.add("Year");
projection.setNonKeyAttributes(nonKeyAttributes);

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()

 .withIndexName("AlbumTitleIndex").withKeySchema(indexKeySchema).withProjection(projection);

ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
 ArrayList<LocalSecondaryIndex>();
localSecondaryIndexes.add(localSecondaryIndex);
createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Você deve aguardar até que o DynamoDB crie a tabela e defina o status dessa tabela como `ACTIVE`. Depois disso, você poderá começar a inserir itens de dados na tabela.

Descrever uma tabela com um índice secundário local

Para obter mais informações sobre índices secundários locais em uma tabela, use o método `describeTable`. Para cada índice, você pode acessar seu nome, esquema de chaves e atributos projetados.

Veja a seguir as etapas para acessar informações do índice secundário local de uma tabela usando a API de documento do AWS SDK para Java.

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `Table`. Você deve fornecer o nome da tabela.
3. Chame o método `describeTable` no objeto `Table`.

O exemplo de código Java a seguir demonstra as etapas anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);

TableDescription tableDescription = table.describe();

List<LocalSecondaryIndexDescription> localSecondaryIndexes
    = tableDescription.getLocalSecondaryIndexes();

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

Iterator<LocalSecondaryIndexDescription> lsiIter = localSecondaryIndexes.iterator();
while (lsiIter.hasNext()) {

    LocalSecondaryIndexDescription lsiDescription = lsiIter.next();
    System.out.println("Info for index " + lsiDescription.getIndexName() + ":");
    Iterator<KeySchemaElement> kseIter = lsiDescription.getKeySchema().iterator();
```

```
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = lsiDescription.getProjection();
    System.out.println("\tThe projection type is: " + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: " +
 projection.getNonKeyAttributes());
    }
}
```

Consultar um índice secundário local

Você pode usar a operação `Query` em um índice secundário local de forma semelhante ao uso de `Query` em uma tabela. Você deve especificar o nome do índice, os critérios de consulta da chave de classificação do índice e os atributos que deseja retornar. Neste exemplo, o índice é `AlbumTitleIndex` e a chave de classificação do índice é `AlbumTitle`.

Os únicos atributos retornados são aqueles que foram projetados no índice. É possível modificar essa consulta para selecionar atributos não chave também, mas isso exigiria atividades de busca de tabela que são relativamente caras. Para obter mais informações sobre buscas de tabela, consulte Projeções de atributo.

Veja a seguir as etapas para consultar um índice secundário local usando a API de documento do AWS SDK para Java.

1. Crie uma instância da classe `DynamoDB`.
2. Crie uma instância da classe `Table`. Você deve fornecer o nome da tabela.
3. Crie uma instância da classe `Index`. Você deve fornecer o nome do índice.
4. Chame o método `query` da classe `Index`.

O exemplo de código Java a seguir demonstra as etapas anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);
```

```
String tableName = "Music";

Table table = dynamoDB.getTable(tableName);
Index index = table.getIndex("AlbumTitleIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"));

ItemCollection<QueryOutcome> items = index.query(spec);

Iterator<Item> itemsIter = items.iterator();

while (itemsIter.hasNext()) {
    Item item = itemsIter.next();
    System.out.println(item.toJSONPretty());
}
```

Exemplo: índices secundários locais que usam a API de documentos do Java

O exemplo de código Java a seguir mostra como trabalhar com índices secundários locais no Amazon DynamoDB. O exemplo cria uma tabela chamada `CustomerOrders` com uma chave de partição `CustomerId` e uma chave de classificação `OrderId`. Há dois índices secundários locais nessa tabela:

- `OrderCreationDateIndex`: a chave de classificação é `OrderCreationDate`, e os seguintes atributos são projetados no índice:
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex`: a chave de classificação é `IsOpen`, e todos os atributos da tabela estão projetados no índice.

Depois que a tabela `CustomerOrders` é criada, o programa carrega a tabela com os dados que representam pedidos de clientes. Ele então consulta os dados usando os índices secundários globais. Por fim, o programa exclui a tabela `CustomerOrders`.

Para obter instruções detalhadas sobre como testar o exemplo a seguir, consulte Exemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.PutItemOutcome;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ReturnConsumedCapacity;
import com.amazonaws.services.dynamodbv2.model.Select;

public class DocumentAPILocalSecondaryIndexExample {

        static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        static DynamoDB dynamoDB = new DynamoDB(client);

        public static String tableName = "CustomerOrders";

        public static void main(String[] args) throws Exception {

                createTable();
                loadData();
```

```
                query(null);
                query("IsOpenIndex");
                query("OrderCreationDateIndex");

                deleteTable(tableName);

        }

        public static void createTable() {

                CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                                .withProvisionedThroughput(
                                                new
ProvisionedThroughput().withReadCapacityUnits((long) 1)

.withWriteCapacityUnits((long) 1));

                // Attribute definitions for table partition and sort keys
                ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("CustomerId").withAttributeType("S"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("OrderId").withAttributeType("N"));

                // Attribute definition for index primary key attributes
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("OrderCreationDate")
                                                .withAttributeType("N"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IsOpen").withAttributeType("N"));

                createTableRequest.setAttributeDefinitions(attributeDefinitions);

                // Key schema for table
                ArrayList<KeySchemaElement> tableKeySchema = new
ArrayList<KeySchemaElement>();
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition
```

```
                                // key
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("OrderId").withKeyType(KeyType.RANGE)); // Sort

                               // key

                createTableRequest.setKeySchema(tableKeySchema);

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();

                // OrderCreationDateIndex
                LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
                                .withIndexName("OrderCreationDateIndex");

                // Key schema for OrderCreationDateIndex
                ArrayList<KeySchemaElement> indexKeySchema = new
ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                               // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("OrderCreationDate")
                                .withKeyType(KeyType.RANGE)); // Sort
                                                              // key

                orderCreationDateIndex.setKeySchema(indexKeySchema);

                // Projection (with list of projected attributes) for
                // OrderCreationDateIndex
                Projection projection = new
Projection().withProjectionType(ProjectionType.INCLUDE);
                ArrayList<String> nonKeyAttributes = new ArrayList<String>();
                nonKeyAttributes.add("ProductCategory");
                nonKeyAttributes.add("ProductName");
                projection.setNonKeyAttributes(nonKeyAttributes);

                orderCreationDateIndex.setProjection(projection);

                localSecondaryIndexes.add(orderCreationDateIndex);
```

```
                // IsOpenIndex
                LocalSecondaryIndex isOpenIndex = new
LocalSecondaryIndex().withIndexName("IsOpenIndex");

                // Key schema for IsOpenIndex
                indexKeySchema = new ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("IsOpen").withKeyType(KeyType.RANGE)); // Sort

                            // key

                // Projection (all attributes) for IsOpenIndex
                projection = new Projection().withProjectionType(ProjectionType.ALL);

                isOpenIndex.setKeySchema(indexKeySchema);
                isOpenIndex.setProjection(projection);

                localSecondaryIndexes.add(isOpenIndex);

                // Add index definitions to CreateTable request
                createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

                System.out.println("Creating table " + tableName + "...");
                System.out.println(dynamoDB.createTable(createTableRequest));

                // Wait for table to become active
                System.out.println("Waiting for " + tableName + " to become
ACTIVE...");
                try {
                        Table table = dynamoDB.getTable(tableName);
                        table.waitForActive();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void query(String indexName) {

                Table table = dynamoDB.getTable(tableName);
```

```
System.out.println("\n***********************************************************\n");
                System.out.println("Querying table " + tableName + "...");

                QuerySpec querySpec = new
QuerySpec().withConsistentRead(true).withScanIndexForward(true)

.withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

                if (indexName == "IsOpenIndex") {

                        System.out.println("\nUsing index: '" + indexName + "': Bob's
orders that are open.");
                        System.out.println("Only a user-specified list of attributes
are returned\n");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid
and IsOpen = :v_isopen")
                                        .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com")
                                                        .withNumber(":v_isopen", 1));

                        querySpec.withProjectionExpression(
                                        "OrderCreationDate, ProductCategory,
ProductName, OrderStatus");

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else if (indexName == "OrderCreationDateIndex") {
                        System.out.println("\nUsing index: '" + indexName
                                        + "': Bob's orders that were placed after
01/31/2015.");
                        System.out.println("Only the projected attributes are returned
\n");
                        Index index = table.getIndex(indexName);
```

```
                    querySpec.withKeyConditionExpression(
                                    "CustomerId = :v_custid and OrderCreationDate
>= :v_orddate")
                                    .withValueMap(
                                                    new
ValueMap().withString(":v_custid", "bob@example.com")

.withNumber(":v_orddate",

20150131));

                    querySpec.withSelect(Select.ALL_PROJECTED_ATTRIBUTES);

                    ItemCollection<QueryOutcome> items = index.query(querySpec);
                    Iterator<Item> iterator = items.iterator();

                    System.out.println("Query: printing results...");

                    while (iterator.hasNext()) {
                            System.out.println(iterator.next().toJSONPretty());
                    }

            } else {
                    System.out.println("\nNo index: All of Bob's orders, by
OrderId:\n");

                    querySpec.withKeyConditionExpression("CustomerId = :v_custid")
                                    .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com"));

                    ItemCollection<QueryOutcome> items = table.query(querySpec);
                    Iterator<Item> iterator = items.iterator();

                    System.out.println("Query: printing results...");

                    while (iterator.hasNext()) {
                            System.out.println(iterator.next().toJSONPretty());
                    }

            }

    }
```

```
        public static void deleteTable(String tableName) {

                Table table = dynamoDB.getTable(tableName);
                System.out.println("Deleting table " + tableName + "...");
                table.delete();

                // Wait for table to be deleted
                System.out.println("Waiting for " + tableName + " to be deleted...");
                try {
                        table.waitForDelete();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void loadData() {

                Table table = dynamoDB.getTable(tableName);

                System.out.println("Loading data into table " + tableName + "...");

                Item item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 1)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150101)
                                .withString("ProductCategory", "Book")
                                .withString("ProductName", "The Great Outdoors")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                PutItemOutcome putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 2)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150221)
                                .withString("ProductCategory", "Bike")
                                .withString("ProductName", "Super Mountain")
                                .withString("OrderStatus", "ORDER RECEIVED");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);
```

```
                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 3)
                                .// no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150304).withString("ProductCategory", "Music")
                                .withString("ProductName", "A Quiet
Interlude").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "176493");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 1)
                                .// no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150111).withString("ProductCategory", "Movie")
                                .withString("ProductName", "Calm Before The Storm")
                                .withString("OrderStatus", "SHIPPING DELAY")
                                .withString("ShipmentTrackingId", "859323");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 2)
                                .// no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150124).withString("ProductCategory", "Music")
                                .withString("ProductName", "E-Z
Listening").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "756943");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 3)
                                .// no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150221).withString("ProductCategory", "Music")
                                .withString("ProductName", "Symphony
9").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "645193");

                putItemOutcome = table.putItem(item);
```

```
                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 4)
                                 .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150222)
                                 .withString("ProductCategory", "Hardware")
                                 .withString("ProductName", "Extra Heavy Hammer")
                                 .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 5)
                                 /* no IsOpen attribute */
                                 .withNumber("OrderCreationDate",
20150309).withString("ProductCategory", "Book")
                                 .withString("ProductName", "How To
Cook").withString("OrderStatus", "IN TRANSIT")
                                 .withString("ShipmentTrackingId", "440185");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 6)
                                 // no IsOpen attribute
                                 .withNumber("OrderCreationDate",
20150318).withString("ProductCategory", "Luggage")
                                 .withString("ProductName", "Really Big
Suitcase").withString("OrderStatus", "DELIVERED")
                                 .withString("ShipmentTrackingId", "893927");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 7)
                                 /* no IsOpen attribute */
                                 .withNumber("OrderCreationDate",
20150324).withString("ProductCategory", "Golf")
                                 .withString("ProductName", "PGA Pro
II").withString("OrderStatus", "OUT FOR DELIVERY")
                                 .withString("ShipmentTrackingId", "383283");

                putItemOutcome = table.putItem(item);
                assert putItemOutcome != null;
```

```
        }

}
```

Como trabalhar com índices secundários locais: .NET

Tópicos

- Criar uma tabela com um índice secundário local
- Descrever uma tabela com um índice secundário local
- Consultar um índice secundário local
- Exemplo: índices secundários locais que usam a API de baixo nível do AWS SDK para .NET

Você pode usar a API de baixo nível do AWS SDK para .NET para criar uma tabela do Amazon DynamoDB com um ou mais índices secundários locais na tabela e executar consultas usando os índices. Essas operações são mapeadas nas ações correspondentes da API de baixo nível do DynamoDB. Para ter mais informações, consulte Exemplos de código .NET.

Veja a seguir as etapas comuns para operações de tabela usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Forneça os parâmetros obrigatórios e opcionais para a operação, criando os objetos de solicitação correspondentes.

   Por exemplo, crie um objeto `CreateTableRequest` para criar uma tabela e crie um objeto `QueryRequest` para consultar uma tabela ou um índice.
3. Execute o método apropriado fornecido pelo cliente que você criou na etapa anterior.

Criar uma tabela com um índice secundário local

Os índices secundários locais devem ser criados ao mesmo tempo que uma tabela é criada. Para fazer isso, use `CreateTable` e forneça suas especificações para um ou mais índices secundários locais. O exemplo de código C# a seguir cria uma tabela para armazenar informações sobre músicas em uma coleção de músicas. A chave de partição é `Artist` e a chave de classificação é `SongTitle`. Um índice secundário, `AlbumTitleIndex`, facilita consultas por título de álbum.

Veja a seguir as etapas necessárias para criar uma tabela com um índice secundário local usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Crie uma instância da classe `CreateTableRequest` para fornecer as informações solicitadas.

   Você deve fornecer o nome da tabela, sua chave primária e os valores de throughput provisionado. Para o índice secundário local, você deve fornecer o nome do índice, o nome e o tipo de dados da chave de classificação do índice, o esquema de chave para o índice e a projeção de atributos.
3. Execute o método `CreateTable` fornecendo o objeto de solicitação como um parâmetro.

O exemplo de código C# a seguir demonstra as etapas anteriores. O código cria uma tabela (`Music`) com um índice secundário no atributo `AlbumTitle`. A chave de partição de tabela e a chave de classificação, bem como a chave de classificação de índice, são os únicos atributos projetados para o índice.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

CreateTableRequest createTableRequest = new CreateTableRequest()
{
    TableName = tableName
};

//ProvisionedThroughput
createTableRequest.ProvisionedThroughput = new ProvisionedThroughput()
{
    ReadCapacityUnits = (long)5,
    WriteCapacityUnits = (long)5
};

//AttributeDefinitions
List<AttributeDefinition> attributeDefinitions = new List<AttributeDefinition>();

attributeDefinitions.Add(new AttributeDefinition()
{
    AttributeName = "Artist",
    AttributeType = "S"
});
```

```
attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "SongTitle",
     AttributeType = "S"
 });

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "AlbumTitle",
     AttributeType = "S"
 });

createTableRequest.AttributeDefinitions = attributeDefinitions;

//KeySchema
List<KeySchemaElement> tableKeySchema = new List<KeySchemaElement>();

tableKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
tableKeySchema.Add(new KeySchemaElement() { AttributeName = "SongTitle", KeyType =
 "RANGE" });  //Sort key

createTableRequest.KeySchema = tableKeySchema;

List<KeySchemaElement> indexKeySchema = new List<KeySchemaElement>();
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "AlbumTitle", KeyType =
 "RANGE" });  //Sort key

Projection projection = new Projection() { ProjectionType = "INCLUDE" };

List<string> nonKeyAttributes = new List<string>();
nonKeyAttributes.Add("Genre");
nonKeyAttributes.Add("Year");
projection.NonKeyAttributes = nonKeyAttributes;

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()
{
    IndexName = "AlbumTitleIndex",
    KeySchema = indexKeySchema,
    Projection = projection
};
```

```
List<LocalSecondaryIndex> localSecondaryIndexes = new List<LocalSecondaryIndex>();
localSecondaryIndexes.Add(localSecondaryIndex);
createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

CreateTableResponse result = client.CreateTable(createTableRequest);
Console.WriteLine(result.CreateTableResult.TableDescription.TableName);
Console.WriteLine(result.CreateTableResult.TableDescription.TableStatus);
```

Você deve aguardar até que o DynamoDB crie a tabela e defina o status dessa tabela como `ACTIVE`. Depois disso, você poderá começar a inserir itens de dados na tabela.

Descrever uma tabela com um índice secundário local

Para obter mais informações sobre índices secundários locais em uma tabela, use a API `DescribeTable`. Para cada índice, você pode acessar seu nome, esquema de chaves e atributos projetados.

Veja a seguir as etapas para acessar informações do índice secundário local para uma tabela usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Crie uma instância da classe `DescribeTableRequest` para fornecer as informações solicitadas. Você deve fornecer o nome da tabela.
3. Execute o método `describeTable` fornecendo o objeto de solicitação como um parâmetro.
4.

O exemplo de código C# a seguir demonstra as etapas anteriores.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest()
 { TableName = tableName });
List<LocalSecondaryIndexDescription> localSecondaryIndexes =
    response.DescribeTableResult.Table.LocalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.
```

```
foreach (LocalSecondaryIndexDescription lsiDescription in localSecondaryIndexes)
{
    Console.WriteLine("Info for index " + lsiDescription.IndexName + ":");

    foreach (KeySchemaElement kse in lsiDescription.KeySchema)
    {
        Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
    }

    Projection projection = lsiDescription.Projection;

    Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

    if (projection.ProjectionType.ToString().Equals("INCLUDE"))
    {
        Console.WriteLine("\t\tThe non-key projected attributes are:");

        foreach (String s in projection.NonKeyAttributes)
        {
            Console.WriteLine("\t\t" + s);
        }

    }
}
```

Consultar um índice secundário local

Você pode usar `Query` em um índice secundário local de forma semelhante ao uso de `Query` em uma tabela. Você deve especificar o nome do índice, os critérios de consulta da chave de classificação do índice e os atributos que deseja retornar. Neste exemplo, o índice é `AlbumTitleIndex` e a chave de classificação do índice é `AlbumTitle`.

Os únicos atributos retornados são aqueles que foram projetados no índice. É possível modificar essa consulta para selecionar atributos não chave também, mas isso exigiria atividades de busca de tabela que são relativamente caras. Para obter mais informações sobre buscas de tabela, consulte Projeções de atributo

Veja a seguir as etapas para consultar um índice secundário local usando a API de baixo nível do .NET.

1. Crie uma instância da classe `AmazonDynamoDBClient`.
2. Crie uma instância da classe `QueryRequest` para fornecer as informações solicitadas.

3. Execute o método `query` fornecendo o objeto de solicitação como um parâmetro.

O exemplo de código C# a seguir demonstra as etapas anteriores.

Example

```
QueryRequest queryRequest = new QueryRequest
{
    TableName = "Music",
    IndexName = "AlbumTitleIndex",
    Select = "ALL_ATTRIBUTES",
    ScanIndexForward = true,
    KeyConditionExpression = "Artist = :v_artist and AlbumTitle = :v_title",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":v_artist",new AttributeValue {S = "Acme Band"}},
        {":v_title",new AttributeValue {S = "Songs About Life"}}
    },
};

QueryResponse response = client.Query(queryRequest);

foreach (var attribs in response.Items)
{
    foreach (var attrib in attribs)
    {
        Console.WriteLine(attrib.Key + " ---> " + attrib.Value.S);
    }
    Console.WriteLine();
}
```

Exemplo: índices secundários locais que usam a API de baixo nível do AWS SDK para .NET

O exemplo de código C# a seguir mostra como trabalhar com índices secundários locais no Amazon DynamoDB. O exemplo cria uma tabela chamada `CustomerOrders` com uma chave de partição `CustomerId` e uma chave de classificação `OrderId`. Há dois índices secundários locais nessa tabela:

- `OrderCreationDateIndex`: a chave de classificação é `OrderCreationDate`, e os seguintes atributos são projetados no índice:

  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex`: a chave de classificação é `IsOpen`, e todos os atributos da tabela estão projetados no índice.

Depois que a tabela `CustomerOrders` é criada, o programa carrega a tabela com os dados que representam pedidos de clientes. Ele então consulta os dados usando os índices secundários globais. Por fim, o programa exclui a tabela `CustomerOrders`.

Para obter instruções passo a passo sobre como testar o exemplo a seguir, consulte Exemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelLocalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "CustomerOrders";

        static void Main(string[] args)
        {
            try
            {
                CreateTable();
                LoadData();
```

```
            Query(null);
            Query("IsOpenIndex");
            Query("OrderCreationDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }
        catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void CreateTable()
    {
        var createTableRequest =
            new CreateTableRequest()
            {
                TableName = tableName,
                ProvisionedThroughput =
                new ProvisionedThroughput()
                {
                    ReadCapacityUnits = (long)1,
                    WriteCapacityUnits = (long)1
                }
            };

        var attributeDefinitions = new List<AttributeDefinition>()
    {
        // Attribute definitions for table primary key
        { new AttributeDefinition() {
              AttributeName = "CustomerId", AttributeType = "S"
          } },
        { new AttributeDefinition() {
              AttributeName = "OrderId", AttributeType = "N"
          } },
        // Attribute definitions for index primary key
        { new AttributeDefinition() {
              AttributeName = "OrderCreationDate", AttributeType = "N"
          } },
        { new AttributeDefinition() {
              AttributeName = "IsOpen", AttributeType = "N"
          }}
```

```
        };

            createTableRequest.AttributeDefinitions = attributeDefinitions;

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderId", KeyType = "RANGE"
              } }                                                   //Sort key
        };

            createTableRequest.KeySchema = tableKeySchema;

            var localSecondaryIndexes = new List<LocalSecondaryIndex>();

            // OrderCreationDateIndex
            LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
            {
                IndexName = "OrderCreationDateIndex"
            };

            // Key schema for OrderCreationDateIndex
            var indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                      //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderCreationDate", KeyType = "RANGE"
              } }                                                                 //Sort key
        };

            orderCreationDateIndex.KeySchema = indexKeySchema;

            // Projection (with list of projected attributes) for
            // OrderCreationDateIndex
            var projection = new Projection()
            {
                ProjectionType = "INCLUDE"
            };
```

```
        var nonKeyAttributes = new List<string>()
    {
        "ProductCategory",
        "ProductName"
    };
        projection.NonKeyAttributes = nonKeyAttributes;

        orderCreationDateIndex.Projection = projection;

        localSecondaryIndexes.Add(orderCreationDateIndex);

        // IsOpenIndex
        LocalSecondaryIndex isOpenIndex
            = new LocalSecondaryIndex()
            {
                IndexName = "IsOpenIndex"
            };

        // Key schema for IsOpenIndex
        indexKeySchema = new List<KeySchemaElement>()
    {
        { new KeySchemaElement() {
              AttributeName = "CustomerId", KeyType = "HASH"
          }},                                                    //Partition key
        { new KeySchemaElement() {
              AttributeName = "IsOpen", KeyType = "RANGE"
          }}                                                  //Sort key
    };

        // Projection (all attributes) for IsOpenIndex
        projection = new Projection()
        {
            ProjectionType = "ALL"
        };

        isOpenIndex.KeySchema = indexKeySchema;
        isOpenIndex.Projection = projection;

        localSecondaryIndexes.Add(isOpenIndex);

        // Add index definitions to CreateTable request
        createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;
```

```
            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);
            WaitUntilTableReady(tableName);
        }

        public static void Query(string indexName)
        {

Console.WriteLine("\n*****************************************************\n");
            Console.WriteLine("Querying table " + tableName + "...");

            QueryRequest queryRequest = new QueryRequest()
            {
                TableName = tableName,
                ConsistentRead = true,
                ScanIndexForward = true,
                ReturnConsumedCapacity = "TOTAL"
            };


            String keyConditionExpression = "CustomerId = :v_customerId";
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue> {
            {":v_customerId", new AttributeValue {
                 S = "bob@example.com"
             }}
        };


            if (indexName == "IsOpenIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that are open.");
                Console.WriteLine("Only a user-specified list of attributes are
returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and IsOpen = :v_isOpen";
                expressionAttributeValues.Add(":v_isOpen", new AttributeValue
                {
                    N = "1"
                });

                // ProjectionExpression
```

```
                queryRequest.ProjectionExpression = "OrderCreationDate,
ProductCategory, ProductName, OrderStatus";
            }
            else if (indexName == "OrderCreationDateIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that were placed after 01/31/2013.");
                Console.WriteLine("Only the projected attributes are returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and OrderCreationDate > :v_Date";
                expressionAttributeValues.Add(":v_Date", new AttributeValue
                {
                    N = "20130131"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo index: All of Bob's orders, by OrderId:\n");
            }
            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "OrderId" || attr == "IsOpen"
                        || attr == "OrderCreationDate")
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].N);
                    }
                    else
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].S);
                    }
                }
                Console.WriteLine();
            }
```

```
            Console.WriteLine("\nConsumed capacity: " +
result.ConsumedCapacity.CapacityUnits + "\n");
        }

        private static void DeleteTable(string tableName)
        {
            Console.WriteLine("Deleting table " + tableName + "...");
            client.DeleteTable(new DeleteTableRequest()
            {
                TableName = tableName
            });
            WaitForTableToBeDeleted(tableName);
        }

        public static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            Dictionary<string, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130101"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Book"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "The Great Outdoors"
```

```
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "PACKING ITEMS"
            };
            /* no ShipmentTrackingId attribute */
            PutItemRequest putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Bike"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Super Mountain"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "ORDER RECEIVED"
            };
            /* no ShipmentTrackingId attribute */
```

```
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "3"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130304"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "A Quiet Interlude"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "IN TRANSIT"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "176493"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
```

```
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130111"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Movie"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Calm Before The Storm"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "SHIPPING DELAY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "859323"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
```

```
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130124"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "E-Z Listening"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "756943"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "3"
            };
            /* no IsOpen attribute */
```

```
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Symphony 9"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "645193"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "4"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130222"
```

```
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Hardware"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Extra Heavy Hammer"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "PACKING ITEMS"
            };
            /* no ShipmentTrackingId attribute */
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "5"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130309"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Book"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "How To Cook"
            };
```

```
            item["OrderStatus"] = new AttributeValue
            {
                S = "IN TRANSIT"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "440185"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "6"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130318"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Luggage"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Really Big Suitcase"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
```

```
            S = "893927"
        };
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "7"
        };
        /* no IsOpen attribute */
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130324"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Golf"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "PGA Pro II"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "OUT FOR DELIVERY"
        };
        item["ShipmentTrackingId"] = new AttributeValue
        {
            S = "383283"
        };
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
```

```
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);
    }

    private static void WaitUntilTableReady(string tableName)
    {
        string status = null;
        // Let us wait until table is created. Call DescribeTable.
        do
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
                    TableName = tableName
                });

                Console.WriteLine("Table name: {0}, status: {1}",
                          res.Table.TableName,
                          res.Table.TableStatus);
                status = res.Table.TableStatus;
            }
            catch (ResourceNotFoundException)
            {
                // DescribeTable is eventually consistent. So you might
                // get resource not found. So we handle the potential exception.
            }
        } while (status != "ACTIVE");
    }

    private static void WaitForTableToBeDeleted(string tableName)
    {
        bool tablePresent = true;

        while (tablePresent)
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
                    TableName = tableName
```

```
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Trabalhar com índices secundários locais no DynamoDB usando a AWS CLI

Você pode usar a AWS CLI para criar uma tabela do Amazon DynamoDB com um ou mais índices secundários locais, descrever os índices na tabela e realizar consultas usando os índices.

Tópicos

- Criar uma tabela com um índice secundário local
- Descrever uma tabela com um índice secundário local
- Consultar um índice secundário local

Criar uma tabela com um índice secundário local

Os índices secundários locais devem ser criados ao mesmo tempo que uma tabela é criada. Para fazer isso, use o parâmetro `create-table` e forneça as especificações para um ou mais índices secundários locais. O exemplo a seguir cria uma tabela (`Music`) para armazenar informações sobre músicas em uma coleção de músicas. A chave de partição é `Artist` e a chave de classificação é `SongTitle`. Um índice secundário, `AlbumTitleIndex`, no atributo `AlbumTitle` facilita consultas por título de álbum.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S  \
```

```
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[{\"IndexName\": \"AlbumTitleIndex\",
        \"KeySchema\":[{\"AttributeName\":\"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"RANGE\"}],
        \"Projection\":{\"ProjectionType\":\"INCLUDE\",  \"NonKeyAttributes\":[\"Genre
\", \"Year\"]}}]"
```

Você deve aguardar até que o DynamoDB crie a tabela e defina o status dessa tabela como `ACTIVE`. Depois disso, você poderá começar a inserir itens de dados na tabela. Você pode usar describe-table para determinar o status da criação da tabela.

Descrever uma tabela com um índice secundário local

Para obter mais informações sobre índices secundários locais em uma tabela, use o parâmetro `describe-table`. Para cada índice, você pode acessar seu nome, esquema de chaves e atributos projetados.

```
aws dynamodb describe-table --table-name Music
```

Consultar um índice secundário local

Você pode usar a operação `query` em um índice secundário local de modo semelhante a como você `query` em uma tabela. Você deve especificar o nome do índice, os critérios de consulta da chave de classificação do índice e os atributos que deseja retornar. Neste exemplo, o índice é `AlbumTitleIndex` e a chave de classificação do índice é `AlbumTitle`.

Os únicos atributos retornados são aqueles que foram projetados no índice. É possível modificar essa consulta para selecionar atributos não chave também, mas isso exigiria atividades de busca de tabela que são relativamente caras. Para obter mais informações sobre buscas de tabela, consulte Projeções de atributo.

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v_artist and AlbumTitle = :v_title" \
    --expression-attribute-values  '{":v_artist":{"S":"Acme Band"},":v_title":
{"S":"Songs About Life"} }'
```

Gerenciar fluxos de trabalho complexos com transações do DynamoDB

As Amazon DynamoDB Transactions simplificam a experiência do desenvolvedor ao fazer alterações de tudo ou nada em vários itens dentro e entre tabelas. As transações fornecem atomicidade, consistência, isolamento e durabilidade (ACID) no DynamoDB, ajudando você a manter a correção dos dados em suas aplicações.

Você pode usar as APIs de leitura e gravação transacionais do DynamoDB para gerenciar fluxos de trabalho empresariais complexos que precisão de adição, atualização ou exclusão de vários itens como uma única operação tudo ou nada. Por exemplo, um desenvolvedor de videogames pode garantir que os perfis dos jogadores sejam atualizados corretamente quando eles trocam itens em um jogo ou fazem compras dentro do jogo.

Com a API de gravação de transações, você pode agrupar várias ações `Put`, `Update`, `Delete` e `ConditionCheck`. Você pode enviar as ações como uma única operação `TransactWriteItems` que é bem-sucedida ou falha como uma unidade. O mesmo é verdade para várias ações `Get`, que você pode agrupar e enviar como uma única operação `TransactGetItems`.

Não há custo adicional para habilitar transações para suas tabelas do DynamoDB. Você paga apenas pelas leituras ou gravações que fazem parte de sua transação. O DynamoDB realiza duas leituras subjacentes ou gravações de cada item na transação: uma para preparar a transação e outra para enviar a transação. Essas duas operações de leitura/gravação subjacentes são visíveis nas métricas do Amazon CloudWatch.

Para começar a usar as transações do DynamoDB, baixe o AWS SDK mais recente ou da AWS Command Line Interface (AWS CLI). Em seguida, siga o Exemplo de transações do DynamoDB.

As seções a seguir fornecem uma visão geral detalhada das APIs de transação e como você pode usá-las no DynamoDB.

Tópicos

- Amazon DynamoDB Transactions: como funciona
- Usar o IAM com transações do DynamoDB
- Exemplo de transações do DynamoDB

Amazon DynamoDB Transactions: como funciona

Com as Amazon DynamoDB Transactions, você pode agrupar várias ações juntas e enviá-las como uma única operação tudo ou nada `TransactWriteItems` ou `TransactGetItems`. As seções a seguir descrevem operações de API, gerenciamento de capacidade, práticas recomendadas e outros detalhes sobre o uso de operações transacionais no DynamoDB.

Tópicos

- API TransactWriteItems
- API TransactGetItems
- Níveis de isolamento para transações do DynamoDB
- Tratamento de conflitos em transações com o DynamoDB
- Usar APIs transacionais no DynamoDB Accelerator (DAX)
- Gerenciar capacidade para transações
- Práticas recomendadas para transações
- Usar APIs Transacionais com tabelas globais
- Transações do DynamoDB comparadas com biblioteca de clientes de transações do AWSLabs

API TransactWriteItems

`TransactWriteItems` é uma operação de gravação síncrona e idempotente que agrupa até 100 ações de gravação em uma única operação tudo ou nada. Essas ações podem ter como destino até 100 itens distintos em uma ou mais tabelas do DynamoDB na mesma conta e região da AWS. O tamanho agregado dos itens na transação não podem exceder 4 MB. As ações são concluídas de forma atômica para que todas sejam bem-sucedidas ou nenhuma tenha êxito.

> **Note**
>
> - Uma operação `TransactWriteItems` difere de uma operação `BatchWriteItem` em que todas as ações contidas devem ser concluídas com êxito, ou nenhuma alteração foi feita. Com uma operação `BatchWriteItem`, é possível que apenas as ações no lote sejam bem-sucedidas enquanto as outras não.
> - As transações não podem ser realizadas usando índices.

Você não pode visar o mesmo item com várias operações dentro da mesma transação. Por exemplo, você não pode executar uma ação `ConditionCheck` e também uma `Update` no mesmo item na mesma transação.

Você pode adicionar os tipos a seguir de ações a uma transação:

- `Put`: inicia uma operação `PutItem` para criar um novo item ou substituir um item antigo com um item novo, de forma condicional ou sem especificar qualquer condição.
- `Update`: inicia uma operação `UpdateItem` para editar um atributo de item existente ou adicionar um novo item à tabela se ele já não existir. Use essa ação para adicionar, excluir ou atualizar atributos ou um item existente de forma condicional ou sem uma condição.
- `Delete`: inicia uma operação `DeleteItem` para excluir um único item em uma tabela identificada por essa chave principal.
- `ConditionCheck`: verifica se um item existe ou verifica a condição de atributos específicos do item.

Quando uma transação é concluída no DynamoDB, suas alterações começam a se propagar para índices secundários globais (GSIs), fluxos e backups. Essa propagação ocorre gradualmente: registros de fluxo da mesma transação podem aparecer em momentos diferentes e podem ser intercalados com registros de outras transações. Os consumidores do fluxo não devem presumir a atomicidade da transação ou as garantias de ordenação.

Para garantir um snapshot atômico dos itens modificados em uma transação, use a operação TransactGetItems para ler todos os itens relevantes juntos. Essa operação fornece uma visão consistente dos dados, garantindo que você veja todas as alterações de uma transação concluída ou nenhuma.

Como a propagação não é imediata, se uma tabela for restaurada do backup (RestoreTableFromBackup) ou exportada para um ponto anterior no tempo (ExportTableToPointInTime) no meio da transação, ela poderá conter apenas algumas das alterações feitas durante uma transação recente.

Idempotência

Você pode, opcionalmente, incluir um token de cliente quando fizer uma chamada `TransactWriteItems` para garantir que a solicitação é idempotente. Realizar transações idempotentes ajuda a prevenir erros de aplicação se a mesma operação for enviada diversas vezes por conta de uma desconexão ou outro problema de conexão.

Se a chamada `TransactWriteItems` original for bem-sucedida, as chamadas `TransactWriteItems` subsequentes com o mesmo token de cliente retornarão com êxito sem nenhuma alteração. Se o parâmetro `ReturnConsumedCapacity` estiver definido, a chamada `TransactWriteItems` inicial retornará o número de unidades de capacidade de gravação consumidas para realizar as alterações. Chamadas `TransactWriteItems` subsequentes com o mesmo token de cliente devolvem o número de unidades de capacidade de leitura consumidas na leitura do item.

Pontos importantes sobre idempotência

- Um token de cliente é válido por 10 minutos após a solicitação utilizada acabar. Depois de 10 minutos, quaisquer solicitações que usarem o mesmo token de cliente são tratadas como uma nova solicitação. Você não deve reutilizar o mesmo token de cliente para a mesma solicitação após 10 minutos.
- Se você repetir uma solicitação com o mesmo token de cliente dentro de 10 minutos na janela de idempotência, mas alterar algum parâmetro de solicitação, o DynamoDB retornará uma exceção `IdempotentParameterMismatch`.

Tratamento de erros para gravação

Transações de gravação não são bem-sucedidas nas circunstâncias a seguir:

- Quando uma condição em uma das expressões de condição não é alcançada.
- Quando uma validação de transação ocorrer por conta de mais de uma ação na mesma operação `TransactWriteItems` ter como destino o mesmo item.
- Quando uma solicitação `TransactWriteItems` entra em conflito com uma operação `TransactWriteItems` em andamento em um ou mais itens na operação `TransactWriteItems`. Nesse caso, a solicitação falha com um `TransactionCanceledException`.
- Quando há capacidade provisionada insuficiente para a transação ser concluída.
- Quando o tamanho de um item é muito maior (maior que 400 KB), ou um índice secundário local (LSI) se torna muito grande, ou um erro de validação semelhante ocorre por conta das alterações feitas pela transação.
- Quando houver um erro de usuário, como formato de dados inválidos.

Para obter mais informações sobre como os conflitos com operações `TransactWriteItems` são gerenciados, consulte Tratamento de conflitos em transações com o DynamoDB.

API TransactGetItems

`TransactGetItems` é uma operação de leitura síncrona que agrupa até 100 ações `Get`. Essas ações podem ter como destino até 100 itens distintos em uma ou mais tabelas do DynamoDB na mesma conta e região da AWS. O tamanho agregado dos itens na transação não pode exceder 4 MB.

As ações `Get` são realizadas de forma atômica para que todas sejam bem-sucedidas ou nenhuma tenha êxito:

- `Get`: inicia uma operação `GetItem` para recuperar um conjunto de atributos do item com a chave primária fornecida. Se não houver item correspondente, `Get` não retornará quaisquer dados.

Tratamento de erros para leitura

Transações de leitura não são bem-sucedidas nas circunstâncias a seguir:

- Quando uma solicitação `TransactGetItems` entra em conflito com uma operação `TransactWriteItems` em andamento em um ou mais itens na operação `TransactGetItems`. Nesse caso, a solicitação falha com um `TransactionCanceledException`.
- Quando há capacidade provisionada insuficiente para a transação ser concluída.
- Quando houver um erro de usuário, como formato de dados inválidos.

Para obter mais informações sobre como os conflitos com operações `TransactGetItems` são gerenciados, consulte Tratamento de conflitos em transações com o DynamoDB.

Níveis de isolamento para transações do DynamoDB

Os níveis de isolamento de operações transacionais (`TransactWriteItems` ou `TransactGetItems`) e outras operações são os seguintes.

SERIALIZÁVEL

Isolamento Serializável garante que os resultados de várias operações concomitantes sejam os mesmos como se nenhuma operação começar com o anterior finalizado.

Há um isolamento serializável entre os tipos a seguir de operação:

- Entre qualquer operação transacional e qualquer padrão de operação de gravação (`PutItem`, `UpdateItem` ou `DeleteItem`).
- Entre qualquer operação transacional e qualquer padrão de operação de leitura (`GetItem`).
- Entre uma operação `TransactWriteItems` e `TransactGetItems`.

Apesar de haver isolamento serializável entre operações transacionais, e cada padrão individual de gravação em uma operação `BatchWriteItem`, não há isolamento serializável entre a transação e a operação `BatchWriteItem` como uma unidade.

De modo semelhante, o nível de isolamento entre uma operação transacional e `GetItems` individual em uma operação `BatchGetItem` é serializável. Porém, o nível de isolamento entre a transação e a operação `BatchGetItem` como uma unidade é confirmado para leitura.

Uma única solicitação `GetItem` é serializável em relação a uma solicitação `TransactWriteItems` de duas formas, antes ou depois da solicitação `TransactWriteItems`. Várias solicitações `GetItem`, contra chaves em solicitações `TransactWriteItems`, podem ser executadas em qualquer ordem e, portanto, os resultados são confirmado para leitura.

Por exemplo, se solicitações `GetItem` para o item A e o item B forem executadas simultaneamente com uma solicitação `TransactWriteItems` que modifica o item A e o item B, existem quatro possibilidades:

- Ambas as solicitações `GetItem` são executadas antes da solicitação `TransactWriteItems`.
- Ambas as solicitações `GetItem` são executadas após a solicitação `TransactWriteItems`.
- A solicitação `GetItem` para o item A é executada antes da solicitação `TransactWriteItems`. Para o item B, `GetItem` é executada após `TransactWriteItems`.
- A solicitação `GetItem` para o item B é executada antes da solicitação `TransactWriteItems`. Para o item A, `GetItem` é executada após `TransactWriteItems`.

Você deverá usar `TransactGetItems` se preferir o nível de isolamento serializável para várias solicitações `GetItem`.

Se uma leitura não transacional for feita em vários itens que faziam parte da mesma solicitação de gravação de transação em andamento, é possível que você consiga ler o novo estado de alguns dos itens e o estado antigo dos outros itens. Você poderá ler o novo estado de todos os itens que faziam parte da solicitação de gravação da transação somente quando uma resposta bem-sucedida for recebida para a gravação transacional, indicando que a transação foi concluída.

Depois que a transação for concluída com êxito e uma resposta for recebida, as operações de leitura finalmente consistente subsequentes ainda poderão exibir o estado antigo por um curto período devido ao modelo de consistência final do DynamoDB. Para garantir a leitura dos dados mais atualizados imediatamente após uma transação, você deve usar leituras altamente consistentes definindo `ConsistentRead` como verdadeiro.

CONFIRMADO PARA LEITURA

O isolamento confirmado para leitura garante que as operações de leitura sempre retornem valores confirmados para um item; a leitura nunca apresentará uma visualização do item representando um estado de uma gravação transacional que não teve êxito final. Isolamento confirmado para leitura não previne modificações do item imediatamente após operação de leitura.

O nível de isolamento é confirmado para leitura entre qualquer operação transacional e qualquer operação de leitura que envolva vários padrões de leitura (`BatchGetItem`, `Query` ou `Scan`). Se uma gravação transacional atualizar um item no meio de uma operação de `BatchGetItem`, `Query` ou `Scan`, a parte subsequente da operação de leitura retornará o novo valor confirmado [com `ConsistentRead`) ou possivelmente um valor confirmado anteriormente (leitura final consistente)].

Resumo da operação

Para resumir, a tabela a seguir mostra os níveis de isolamento entre uma operação transacional (`TransactWriteItems` ou `TransactGetItems`) e outras operações.

Operação	Nível de isolamento
`DeleteItem`	Serializável
`PutItem`	Serializável
`UpdateItem`	Serializável
`GetItem`	Serializável
`BatchGetItem`	Confirmado para leitura*
`BatchWriteItem`	NÃO serializável*
`Query`	Confirmado para leitura
`Scan`	Confirmado para leitura

Operação	Nível de isolamento
Outra operação transacional	Serializável

Níveis marcados com asterisco (*) aplicam-se à operação como uma unidade. No entanto, ações individuais dentro dessas operações tem um nível de isolamento serializável.

Tratamento de conflitos em transações com o DynamoDB

Um conflito de transação pode correr em solicitações simultâneas no nível do item, em um item dentro de uma transação. Os conflitos de transação podem ocorrer nos seguintes casos:

- Uma solicitação `PutItem`, `UpdateItem` ou `DeleteItem` de um item conflita com uma solicitação `TransactWriteItems` em andamento que inclui o mesmo item.
- Um item dentro de uma solicitação `TransactWriteItems` é parte de outra solicitação `TransactWriteItems` em andamento.
- Um item dentro de uma solicitação `TransactGetItems` é parte de outra solicitação `TransactWriteItems`, `BatchWriteItem`, `PutItem`, `UpdateItem` ou `DeleteItem` em andamento.

> **Note**
>
> - Quando uma solicitação `PutItem`, `UpdateItem` ou `DeleteItem` é rejeitada, a solicitação falha com um `TransactionConflictException`,
> - Se qualquer solicitação no nível do item dentro de `TransactWriteItems` ou `TransactGetItems` é rejeitada, a solicitação falha com um `TransactionCanceledException`. Se essa solicitação falhar, os AWS SDKs não tentarão repetir a solicitação.
>
>   Se você estiver usando AWS SDK para Java, a exceção conterá a lista de CancellationReasons, em ordem de acordo com a lista de itens no parâmetro de solicitação `TransactItems`. Para outros idiomas, uma representação em string da lista é incluída na mensagem de erro de exceção.
> - Se uma operação `TransactWriteItems` ou `TransactGetItems` em andamento entrar em conflito com uma solicitação `GetItem` simultânea, as duas podem ter êxito.

A métrica TransactionConflict do CloudWatch é incrementada em cada solicitação que falhou no nível do item

Usar APIs transacionais no DynamoDB Accelerator (DAX)

O DynamoDB Accelerator (DAX) oferece suporte a `TransactWriteItems` e `TransactGetItems` com os mesmos níveis de isolamento que no DynamoDB.

`TransactWriteItems` grava via DAX. O DAX passa uma chamada `TransactWriteItems` para o DynamoDB e retorna a resposta. Para preencher o cache após a gravação, o DAX chama `TransactGetItems` em segundo plano para cada item na operação `TransactWriteItems`, o que consome unidades de capacidade de leitura adicionais. (Para ter mais informações, consulte Gerenciar capacidade para transações.) Essa funcionalidade permite manter simples a lógica da aplicação e usar o DAX para operações transacionais e não transacionais.

Chamadas de `TransactGetItems` são transmitidas pelo DAX sem que os itens sejam armazenados em cache localmente. É o mesmo comportamento para APIs de leitura fortemente consistente no DAX.

Gerenciar capacidade para transações

Não há custo adicional para habilitar transações para suas tabelas do DynamoDB. Você paga apenas pelas leituras ou gravações que fazem parte de sua transação. O DynamoDB realiza duas leituras subjacentes ou gravações de cada item na transação: uma para preparar a transação e outra para enviar a transação. Essas duas operações de leitura/gravação subjacente são visíveis nas métricas do Amazon CloudWatch.

Planeje para leituras e gravações adicionais que são necessárias por APIs transacionais ao provisionar a capacidade de suas tabelas. Por exemplo, suponha que sua aplicação execute uma transação por segundo e cada transação grave itens de 500 bytes em sua tabela. Cada item requer duas capacidades de gravação (WCUs): uma para preparar a transação e uma para enviar a transação. Portanto, você precisa provisionar seis WCUs na tabela.

Se você usar o DynamoDB Accelerator (DAX) no exemplo anterior, use também as unidades de capacidade de leitura (RCUs) em cada item na chamada `TransactWriteItems`. Então, você precisa provisionar seis RCUs adicionais na tabela.

De forma semelhante, se sua aplicação executa uma transação de leitura por segundo, e cada transação lê itens de 500 bytes na sua tabela, é preciso prover seis unidades de capacidade de

leitura (RCUs) na tabela. A leitura de cada item requer duas RCUs: uma para preparar a transação e uma para enviar a transação.

Além disso, comportamento de SDK padrão é tentar transações novamente em caso de uma exceção `TransactionInProgressException`. Planeje para as unidades de capacidade de leitura adicional (RCUs) que essas tentativas repetitivas consumem. O mesmo é verdade se você estiver tentando novamente transações em seu código usando um `ClientRequestToken`.

Práticas recomendadas para transações

Considere as seguintes práticas recomendadas ao usar transações do DynamoDB.

- Habilitar Auto Scaling nas suas tabelas, ou garantir que você tem capacidade de throughput suficiente para realizar as duas operações de leitura ou gravação em cada item na transação.
- Se não estiver usando um SDK fornecido pela AWS, inclua um atributo `ClientRequestToken` quando realizar uma chamada `TransactWriteItems` para garantir que a solicitação é idempotente.
- Não agrupe operações em uma transação se não for necessário. Por exemplo, se uma transação única com 10 operações puder ser separada em várias transações sem comprometimento da exatidão do aplicativo, recomendamos separar a transação. Transações mais simples melhorar a taxa de transferência e têm maior chance de êxito.
- Transações múltiplas atualizando os mesmos itens simultaneamente podem causar conflitos que cancelam as transações. Recomendamos as práticas recomendadas do DynamoDB a seguir para modelagem de dados a fim de minimizar esses conflitos.
- Se um conjunto de atributos for frequentemente atualizado entre vários itens como parte de uma única transação, considere agrupar os atributos em um único item para reduzir o escopo da transação.
- Evite usar transações para adicionar dados no grupo. Para gravações em grupo, é melhor usar `BatchWriteItem`.

Usar APIs Transacionais com tabelas globais

As operações transacionais oferecem garantia de atomicidade, consistência, isolamento e durabilidade (ACID) somente na região da AWS em que a API de gravação foi invocada. As transações não são compatíveis entre regiões em tabelas globais. Por exemplo, suponha que você tenha uma tabela global com réplicas nas regiões Leste dos EUA (Ohio) e Oeste dos EUA (Oregon)

e execute uma operação `TransactWriteItems` na região Leste dos EUA (Norte da Virgínia). Você pode observar transações parcialmente concluídas na região Oeste dos EUA (Oregon) conforme as alterações são replicadas. As alterações só serão replicadas para outras regiões quando forem confirmadas na região de origem.

Transações do DynamoDB comparadas com biblioteca de clientes de transações do AWSLabs

As transações do DynamoDB fornecem uma substituição mais eficiente em termos de custos, efetiva e com maior performance para as transações da biblioteca de clientes do AWSLabs. Sugerimos atualizar os aplicativos para usar com as APIs de transação do lado do servidor nativas.

Usar o IAM com transações do DynamoDB

Você pode usar o AWS Identity and Access Management (IAM) para restringir as ações que as operações transacionais podem executar no Amazon DynamoDB. Para obter mais informações sobre como usar as políticas do IAM no DynamoDB, consulte Políticas baseadas em identidade para o DynamoDB.

Permissões para `Put`, `Update`, `Delete`, e ações `Get` são governadas pelas permissões usadas para operações `PutItem`, `UpdateItem`, `DeleteItem` e `GetItem` subjacentes. Para a ação `ConditionCheck`, é possível usar a permissão `dynamodb:ConditionCheckItem` nas políticas do IAM.

Veja a seguir exemplos de políticas do IAM que você pode usar para configurar transações do DynamoDB.

Exemplo 1: permitir operações transacionais

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
```

```
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

Exemplo 2: permitir apenas operações transacionais

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        }
    ]
}
```

Exemplo 3: permitir leituras e gravações não transacionais e bloquear leituras e gravações transacionais

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
             "Action": [
                 "dynamodb:PutItem",
                 "dynamodb:DeleteItem",
                 "dynamodb:GetItem",
                 "dynamodb:UpdateItem"
             ],
             "Resource": [
                 "arn:aws:dynamodb:*:*:table/table04"
             ]
         }
    ]
```

```
}
```

Exemplo 4: evitar que informações sejam devolvidas com a falha ConditionCheck

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/table01",
            "Condition": {
                "StringEqualsIfExists": {
                    "dynamodb:ReturnValues": "NONE"
                }
            }
        }
    ]
}
```

Exemplo de transações do DynamoDB

Como exemplo de uma situação em que as Amazon DynamoDB Transactions podem ser úteis, considere este exemplo de aplicação Java para um marketplace online.

A aplicação tem três tabelas do DynamoDB no backend:

- `Customers`: esta tabela armazena detalhes sobre os clientes do marketplace. Sua chave primária é um identificador exclusivo `CustomerId`.
- `ProductCatalog`: esta tabela armazena detalhes como preço e disponibilidade sobre os produtos para venda no mercado. Sua chave primária é um identificador exclusivo `ProductId`.

- `Orders`: esta tabela armazena detalhes sobre os pedidos do marketplace. Sua chave primária é um identificador exclusivo `OrderId`.

Fazer um pedido

Os trechos de código a seguir ilustram como usar transações do DynamoDB para coordenar as várias etapas necessárias para criar e processar um pedido. O uso de uma única operação de tudo ou nada garante que, se qualquer parte da transação falhar, nenhuma ação na transação será executada e nenhuma alteração será feita.

Neste exemplo, você configura um pedido de um cliente cujo `customerId` é `09e8e9c8-ec48`. Em seguida, execute-o como uma única transação usando o seguinte fluxo de trabalho simples de processamento de pedidos:

1. Determine se o ID do cliente é válido.
2. Verifique se o produto é `IN_STOCK` e atualize o status do produto para `SOLD`.
3. Certifique-se de que o pedido ainda não exista e crie-o.

Validar o cliente

Primeiro, defina uma ação para verificar se um cliente com `customerId` igual a `09e8e9c8-ec48` existe na tabela de clientes.

```
final String CUSTOMER_TABLE_NAME = "Customers";
final String CUSTOMER_PARTITION_KEY = "CustomerId";
final String customerId = "09e8e9c8-ec48";
final HashMap<String, AttributeValue> customerItemKey = new HashMap<>();
customerItemKey.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));

ConditionCheck checkCustomerValid = new ConditionCheck()
    .withTableName(CUSTOMER_TABLE_NAME)
    .withKey(customerItemKey)
    .withConditionExpression("attribute_exists(" + CUSTOMER_PARTITION_KEY + ")");
```

Atualizar o status do produto

Em seguida, defina uma ação para atualizar o status do produto para `SOLD` se a condição do status atual do produto `IN_STOCK` for `true`. Configurar a opção

`ReturnValuesOnConditionCheckFailure` retornará o item se o atributo de status do produto do item não for igual a `IN_STOCK`.

```
final String PRODUCT_TABLE_NAME = "ProductCatalog";
final String PRODUCT_PARTITION_KEY = "ProductId";
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
expressionAttributeValues.put(":new_status", new AttributeValue("SOLD"));
expressionAttributeValues.put(":expected_status", new AttributeValue("IN_STOCK"));

Update markItemSold = new Update()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey)
    .withUpdateExpression("SET ProductStatus = :new_status")
    .withExpressionAttributeValues(expressionAttributeValues)
    .withConditionExpression("ProductStatus = :expected_status")

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD);
```

Criar o pedido

Por último, crie o pedido, desde que um pedido com esse `OrderId` ainda não exista.

```
final String ORDER_PARTITION_KEY = "OrderId";
final String ORDER_TABLE_NAME = "Orders";

HashMap<String, AttributeValue> orderItem = new HashMap<>();
orderItem.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));
orderItem.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));
orderItem.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));
orderItem.put("OrderStatus", new AttributeValue("CONFIRMED"));
orderItem.put("OrderTotal", new AttributeValue("100"));

Put createOrder = new Put()
    .withTableName(ORDER_TABLE_NAME)
    .withItem(orderItem)

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD)
    .withConditionExpression("attribute_not_exists(" + ORDER_PARTITION_KEY + ")");
```

Executar a transação

O exemplo a seguir ilustra como executar as ações definidas anteriormente como uma única operação tudo ou nada.

```
    Collection<TransactWriteItem> actions = Arrays.asList(
        new TransactWriteItem().withConditionCheck(checkCustomerValid),
        new TransactWriteItem().withUpdate(markItemSold),
        new TransactWriteItem().withPut(createOrder));

    TransactWriteItemsRequest placeOrderTransaction = new TransactWriteItemsRequest()
        .withTransactItems(actions)
        .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

    // Run the transaction and process the result.
    try {
        client.transactWriteItems(placeOrderTransaction);
        System.out.println("Transaction Successful");

    } catch (ResourceNotFoundException rnf) {
        System.err.println("One of the table involved in the transaction is not found"
 + rnf.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println("Internal Server Error" + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.out.println("Transaction Canceled " + tce.getMessage());
    }
```

Ler os detalhes do pedido

O exemplo a seguir mostra como ler a ordem concluída transacionalmente entre as tabelas `Orders` e `ProductCatalog`.

```
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

HashMap<String, AttributeValue> orderKey = new HashMap<>();
orderKey.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));

Get readProductSold = new Get()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey);
```

```
Get readCreatedOrder = new Get()
    .withTableName(ORDER_TABLE_NAME)
    .withKey(orderKey);

Collection<TransactGetItem> getActions = Arrays.asList(
    new TransactGetItem().withGet(readProductSold),
    new TransactGetItem().withGet(readCreatedOrder));

TransactGetItemsRequest readCompletedOrder = new TransactGetItemsRequest()
    .withTransactItems(getActions)
    .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

// Run the transaction and process the result.
try {
    TransactGetItemsResult result = client.transactGetItems(readCompletedOrder);
    System.out.println(result.getResponses());
} catch (ResourceNotFoundException rnf) {
    System.err.println("One of the table involved in the transaction is not found" +
 rnf.getMessage());
} catch (InternalServerErrorException ise) {
    System.err.println("Internal Server Error" + ise.getMessage());
} catch (TransactionCanceledException tce) {
    System.err.println("Transaction Canceled" + tce.getMessage());
}
```

Captura de dados de alterações com o Amazon DynamoDB

Muitas aplicações podem se beneficiar da capacidade de capturar alterações nos itens armazenados em uma tabela do DynamoDB no momento em que elas ocorrem. Veja a seguir alguns exemplos de casos de uso:

- Uma aplicação móvel popular modifica os dados em uma tabela do DynamoDB a uma taxa de milhares de atualizações por segundo. Outra aplicação captura e armazena dados sobre essas atualizações, fornecendo métricas de uso quase em tempo real para a aplicação móvel.
- Uma aplicação financeira modifica os dados do mercado de ações em uma tabela do DynamoDB. Diferentes aplicações executadas em paralelo acompanham essas mudanças em tempo real, calculam o valor em risco e reequilibram automaticamente as carteiras com base nos movimentos dos preços das ações.
- Sensores em veículos de transporte e equipamentos industriais enviam dados para uma tabela do DynamoDB. Diferentes aplicações monitoram a performance e enviam alertas de mensagens

quando um problema é detectado, preveem possíveis defeitos aplicando algoritmos de machine learning e compactam e arquivam dados no Amazon Simple Storage Service (Amazon S3).

- Uma aplicação envia notificações automaticamente aos dispositivos móveis de todos os jogadores de um grupo assim que um deles carrega uma nova imagem.
- Um novo cliente adiciona dados a uma tabela do DynamoDB. Esse evento invoca outra aplicação que envia um e-mail de boas-vindas ao novo cliente.

O DynamoDB oferece suporte a streaming de registros de captura de dados de alteração em nível de item em tempo quase real. Você pode criar aplicações que consomem esses fluxos e executam ações com base no conteúdo.

> **Note**
>
> Não há suporte para a adição de tags ao DynamoDB Streams e o uso de controle de acesso por atributo (ABAC) com o DynamoDB Streams.

O vídeo a seguir apresenta uma introdução sobre o conceito de captura de dados de alteração.

Modos de capacidade de tabela

Tópicos

- Opções de streaming para captura de dados de alteração
- Use o Kinesis Data Streams para capturar alterações do DynamoDB
- Capturar dados de alterações para o DynamoDB Streams

Opções de streaming para captura de dados de alteração

O DynamoDB oferece dois modelos de streaming para captura de dados de alteração: Kinesis Data Streams para DynamoDB e DynamoDB Streams.

Para ajudar você a escolher a solução certa para sua aplicação, a tabela a seguir resume os recursos de cada modelo de streaming.

Propriedades	Kinesis Data Streams para DynamoDB	DynamoDB Streams
Retenção de dados	Até 1 ano.	24 horas
Compatibilidade com a Kinesis Client Library (KCL)	Compatível com a KCL versões 1.x, 2.x e 3.x.	Compatível com a KCL versões 1.X e 2.X.
Número de consumidores	Até 5 consumidores simultâneos por fragmento ou até 20 consumidores simultâneos por fragmento com fan-out aprimorado.	Até 2 consumidores simultâneos por fragmento.
Cotas de throughput	Ilimitada.	Sujeito a cotas de throughput por tabela do DynamoDB e região da AWS.
Registro de modelo de entrega	Modelo Pull via HTTP usando GetRecords e com fan-out aprimorado, o Kinesis Data Streams envia os registros por HTTP/2 usando SubscribeToShard.	Modelo Pull via HTTP usando GetRecords.
Ordenação dos registos	O atributo timestamp em cada registro de fluxo pode ser usado para identificar a ordem real na qual as alterações ocorreram na tabela do DynamoDB.	Para cada item modificado em uma tabela do DynamoDB, os registros de fluxo aparecem na mesma sequência que as modificações reais no item.
Registros duplicados	Registros duplicados podem aparecer ocasionalmente no fluxo.	Nenhum registro duplicado aparece no fluxo.
Opções de processamento de fluxos	Processe registros de fluxo usando o AWS Lambda, o	Processe registros de fluxo usando o AWS Lambda ou

Propriedades	Kinesis Data Streams para DynamoDB	DynamoDB Streams
	Amazon Managed Service for Apache Flink, Kinesis Data Firehose ou ETL de streaming do AWS Glue.	o adaptador do DynamoDB Streams Kinesis.
Nível de durabilidade	Zonas de disponibilidade para fornecer failover automático sem interrupção.	Zonas de disponibilidade para fornecer failover automático sem interrupção.

É possível habilitar ambos os modelos de streaming na mesma tabela do DynamoDB.

O vídeo a seguir aprofunda-se nas diferenças entre as duas opções.

DynamoDB Streams versus Kinesis Data Streams

Use o Kinesis Data Streams para capturar alterações do DynamoDB

É possível usar o Amazon Kinesis Data Streams para capturar alterações no Amazon DynamoDB.

O Kinesis Data Streams capta alterações no nível de item em qualquer DynamoDB e as replica em um fluxo de dados do Kinesis. Suas aplicações podem acessar esse fluxo e visualizar as alterações em nível de item praticamente em tempo real. Você pode capturar e armazenar continuamente terabytes de dados por hora. Aproveite o tempo de retenção de dados mais longo e com o recurso de distribuição avançada para atingir simultaneamente duas ou mais aplicações downstream. Outros benefícios incluem auditoria adicional e transparência de segurança.

O Kinesis Data Streams também oferece acesso ao Amazon Kinesis Data Firehose e ao Amazon Managed Service for Apache Flink. Isso permite construir aplicações para alimentar painéis em tempo real, gerar alertas, implementar definições de preço e de publicidade dinâmicas e implementar análises de dados e algoritmos de machine learning sofisticados.

> **Note**
>
> O uso do Kinesis Data Streams para DynamoDB está sujeito ao preço do Kinesis Data Streams para o fluxo de dados e ao preço do DynamoDB para a tabela de origem.

Como o Kinesis Data Streams funciona com o DynamoDB

Quando um fluxo de dados do Kinesis é habilitado para uma tabela do DynamoDB, a tabela envia um registro de dados que captura alterações nos dados dessa tabela. Esse registro de dados inclui:

- O horário específico em que um item foi criado, atualizado ou excluído recentemente
- A chave primária desse item
- Um snapshot do registro antes da modificação
- Um snapshot do registro após a modificação

Esses registros de dados são capturados e publicados praticamente em tempo real. Depois de gravados no fluxo de dados do Kinesis, poderão ser lidos como qualquer outro registro. É possível usar a Biblioteca de clientes Kinesis, usar o AWS Lambda, chamar a API do Kinesis Data Streams e usar outros serviços conectados. Para obter mais informações, consulte Ler dados do Amazon Kinesis Data Streams no Guia do desenvolvedor do Amazon Kinesis Data Streams.

Essas alterações nos dados também são capturadas de forma assíncrona. O Kinesis não afetará a performance da tabela da qual estiver fazendo streaming. Os registros de fluxo armazenados no fluxo de dados do Kinesis também são criptografados em repouso. Para obter mais informações, consulte Proteção de dados no Amazon Kinesis Data Streams.

Os registros de fluxo de dados do Kinesis poderão aparecer em uma sequência diferente da sequência na qual ocorreram as modificações dos itens. As mesmas notificações de item também poderão aparecer mais de uma vez no fluxo. Confira o atributo `ApproximateCreationDateTime` para identificar a sequência em que as modificações dos itens ocorreram e para identificar registros duplicados.

Ao habilitar um fluxo de dados do Kinesis como destino de streaming de uma tabela do DynamoDB, você pode configurar a precisão dos valores de `ApproximateCreationDateTime` em milissegundos ou microssegundos. Por padrão, `ApproximateCreationDateTime` indica a hora da alteração em milissegundos. Além disso, é possível alterar esse valor em um destino de streaming ativo. Depois dessa atualização, os registros de fluxo gravados no Kinesis terão valores de `ApproximateCreationDateTime` com a precisão desejada.

Valores binários gravados no DynamoDB devem ser codificados em formato codificado em base64. No entanto, quando os registros de dados são gravados em um fluxo de dados do Kinesis, esses valores binários são codificados em base64 pela segunda vez. Ao ler esses registros de um fluxo de

dados do Kinesis, para recuperar os valores binários brutos, as aplicações devem decodificar esses valores duas vezes.

O DynamoDB cobra pelo uso para Kinesis Data Streams em unidades de captura de dados de alterações. 1 KB de alteração por item único conta como uma unidade de captura de dados de alteração. Os KB de alteração em cada item são calculados pela maior das imagens de “antes” e “depois” do item gravado no fluxo, usando a mesma lógica que o consumo de unidades de capacidade para operações de gravação. Não é necessário provisionar a capacidade de throughput para unidades de captura de dados de alteração, semelhante a como o modo sob demanda do DynamoDB funciona.

Ativar um fluxo de dados do Kinesis para a tabela do DynamoDB

Você pode habilitar ou desabilitar o streaming para o Kinesis em uma tabela existente do DynamoDB usando o AWS Management Console, o AWS SDK ou a AWS Command Line Interface (AWS CLI).

- Só é possível fazer o streaming de dados do DynamoDB para o Kinesis Data Streams na mesma conta da AWS e região da AWS que sua tabela.
- Só é possível fazer o streaming de dados de uma tabela do DynamoDB para um fluxo de dados do Kinesis.

Alterar um destino do Kinesis Data Streams em uma tabela do DynamoDB

Por padrão, todos os registros de fluxo de dados do Kinesis incluem um atributo `ApproximateCreationDateTime`. Esse atributo representa um carimbo de data e hora, em milissegundos, da hora aproximada em que cada registro foi criado. Você pode alterar a precisão desses valores usando o https://console.aws.amazon.com/kinesis, o SDK ou a AWS CLI.

Conceitos básicos do Kinesis Data Streams para o Amazon DynamoDB

Esta seção descreve como usar o Kinesis Data Streams para tabelas do Amazon DynamoDB com o console do Amazon DynamoDB, a AWS Command Line Interface (AWS CLI) e a API.

Criar um fluxo de dados ativo do Amazon Kinesis

Todos esses exemplos usam a tabela `Music` do DynamoDB que foi criada como parte do tutorial Conceitos básicos do DynamoDB.

Para saber mais sobre como criar consumidores e conectar seu fluxo de dados do Kinesis a outros produtos da AWS, consulte Ler dados do Amazon Kinesis Data Streams no Guia do desenvolvedor do Amazon Kinesis Data Streams.

> **Note**
>
> Quando você estiver usando fragmentos do KDS pela primeira vez, recomendamos configurá-los para que aumentem e reduzam a escala verticalmente de acordo com os padrões de uso. Depois de acumular mais dados sobre os padrões de uso, você poderá ajustar os fragmentos em seu fluxo para fazer a correspondência.

Console

1. Faça login no AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com/kinesis/.
2. Escolha Create data stream (Criar fluxo de dados) e siga as instruções para criar um fluxo chamado `samplestream`.
3. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
4. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
5. Escolha a tabela Music.
6. Escolha a guia Exports and streams (Exportações e fluxos).
7. (Opcional) Em Detalhes do stream de dados do Amazon Kinesis, você pode alterar a precisão do carimbo de data e hora do registro de microssegundos (padrão) para milissegundos.
8. Escolha samplestream na lista suspensa.
9. Escolha o botão Ativar.

AWS CLI

1. Criar um fluxo de dados do Kinesis denominado `samplestream` usando o comando create-stream.

```
aws kinesis create-stream --stream-name samplestream --shard-count 3
```

Consulte Considerações sobre gerenciamento de fragmentos para o Kinesis Data Streams antes de definir o número de fragmentos para o fluxo de dados do Kinesis.

2. Verifique se o stream do Kinesis está ativo e pronto para uso usando o comando describe-stream.

```
aws kinesis describe-stream --stream-name samplestream
```

3. Habilite o streaming do Kinesis na tabela do DynamoDB usando o comando `enable-kinesis-streaming-destination` do DynamoDB. Substitua o valor de `stream-arn` pelo que foi retornado por `describe-stream` na etapa anterior. Opcionalmente, habilite o streaming com uma precisão mais granular (microssegundos) dos valores de carimbo de data e hora retornados em cada registro.

   Habilite o streaming com precisão de carimbo de data e hora em microssegundos:

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
  --enable-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

   Ou habilite o streaming com a precisão padrão do carimbo de data e hora (milissegundos):

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. Verifique se o streaming do Kinesis está ativa na tabela usando o comando `describe-kinesis-streaming-destination` do DynamoDB.

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. Grave os dados na tabela do DynamoDB usando o comando `put-item`, conforme descrito no Guia do desenvolvedor do DynamoDB.

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
```

```
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me
 Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"} }'
```

6. Use o comando get-records da CLI do Kinesis para recuperar o conteúdo de fluxo do Kinesis. Em seguida, use o trecho de código a seguir para desserializar o conteúdo do fluxo.

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we want to fetch the value
     * of this attribute from the new item image. The following code fetches
 this value.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
```

```
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Java

1. Siga as instruções no Guia do desenvolvedor do Kinesis Data Streams para criar um fluxo de dados do Kinesis chamada `samplestream` usando Java.

   Consulte Considerações sobre gerenciamento de fragmentos para o Kinesis Data Streams antes de definir o número de fragmentos para o fluxo de dados do Kinesis.

2. Use o trecho de código a seguir para habilitar a transmissão do Kinesis na tabela do DynamoDB. Opcionalmente, habilite o streaming com uma precisão mais granular (microssegundos) dos valores de carimbo de data e hora retornados em cada registro.

   Habilite o streaming com precisão de carimbo de data e hora em microssegundos:

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

   Ou habilite o streaming com a precisão padrão do carimbo de data e hora (milissegundos):

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
```

```
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. Siga as instruções do Kinesis Data Streams developer guide (Guia do desenvolvedor do Kinesis Data Streams) para ler o fluxo de dados criado.

4. Use o trecho de código a seguir para desserializar o conteúdo do fluxo

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we wanted to fetch the value
     * of this attribute from the new item image, the below code would fetch
 this.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
```

```
}
```

Alterar um fluxo de dados ativo do Amazon Kinesis

Esta seção descreve como alterar uma configuração ativa do Kinesis Data Streams para o DynamoDB usando o console, a AWS CLI e a API.

AWS Management Console

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. Acesse a tabela.
3. Escolha Exportações e streams.

AWS CLI

1. Chame `describe-kinesis-streaming-destination` para confirmar se o fluxo está `ACTIVE`.
2. Chame `UpdateKinesisStreamingDestination`, como neste exemplo:

```
aws dynamodb update-kinesis-streaming-destination --table-name
 enable_test_table --stream-arn arn:aws:kinesis:us-east-1:12345678901:stream/
enable_test_stream --update-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

3. Chame `describe-kinesis-streaming-destination` para confirmar se o fluxo está `UPDATING`.
4. Chame `describe-kinesis-streaming-destination` periodicamente até que o status de streaming volte a ser `ACTIVE`. Normalmente, demora até 5 minutos para que as atualizações de precisão do carimbo de data e hora entrem em vigor. A atualização desse status indicará que a operação terá sido concluída e que o novo valor de precisão será aplicado em registros futuros.
5. Grave na tabela usando `putItem`.
6. Use o comando `get-records` do Kinesis para obter o conteúdo do fluxo.
7. Confirme se `ApproximateCreationDateTime` das gravações tem a precisão desejada.

API Java

1. Forneça um trecho de código que construa uma solicitação `UpdateKinesisStreamingDestination` e uma resposta `UpdateKinesisStreamingDestination`.
2. Forneça um trecho de código que construa uma solicitação `DescribeKinesisStreamingDestination` e uma `DescribeKinesisStreamingDestination response`.
3. Chame `describe-kinesis-streaming-destination` periodicamente até que o status de streaming volte a ser `ACTIVE`, indicando que a atualização foi concluída e que o novo valor de precisão será aplicado em registros futuros.
4. Faça gravações na tabela.
5. Faça uma leitura do fluxo e desserialize o conteúdo do fluxo.
6. Confirme se `ApproximateCreationDateTime` das gravações tem a precisão desejada.

Usar fragmentos e métricas com o DynamoDB Streams e o Kinesis Data Streams

Considerações sobre gerenciamento de fragmentos para o Kinesis Data Streams

Um fluxo de dados do Kinesis contabiliza o throughput em fragmentos. No Amazon Kinesis Data Streams, você pode escolher entre os modos sob demanda e provisionado para os fluxos de dados.

Recomendamos usar o modo sob demanda para seu fluxo de dados do Kinesis se sua workload de gravação do DynamoDB for altamente variável e imprevisível. No modo sob demanda, nenhum planejamento de capacidade é necessário, já que o Kinesis Data Streams gerencia automaticamente os fragmentos para fornecer o throughput necessário.

Para workloads previsíveis, você pode usar o modo provisionado para seu fluxo de dados do Kinesis. No modo provisionado, você precisa especificar o número de fragmentos para o fluxo de dados a fim de acomodar os registros de captura de dados de alterações do DynamoDB. Para determinar o número de fragmentos de que o fluxo de dados do Kinesis precisará para oferecer suporte à sua tabela do DynamoDB, são necessários os seguintes valores de entrada:

- O tamanho médio do registro da tabela do DynamoDB em bytes (`average_record_size_in_bytes`).
- O número máximo de operações de gravação que a tabela do DynamoDB executará por segundo. Isso inclui operações de criação, exclusão e atualização executadas pelas aplicações, bem como operações geradas automaticamente, como operações de exclusão geradas por vida útil (`write_throughput`).

- O percentual de operações de atualização e substituição que você executou na tabela em comparação com operações de criação ou exclusão (`percentage_of_updates`). As operações de atualização e substituição replicam as imagens antigas e novas do item modificado para o fluxo. Isso gera o dobro do tamanho do item do DynamoDB.

Você pode calcular o número de fragmentos (`number_of_shards`) necessários para o fluxo de dados do Kinesis usando os valores de entrada na seguinte fórmula:

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

Por exemplo, você tem um throughput máximo de 1.040 operações de gravação por segundo (`write_throughput`) com um tamanho médio de registro de 800 bytes (`average_record_size_in_bytes`). Se 25% dessas operações de gravação forem operações de atualização (`percentage_of_updates`), serão necessários dois fragmentos (`number_of_shards`) para acomodar o throughput de streaming do DynamoDB:

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

Considere o seguinte antes de usar a fórmula para calcular o número de fragmentos necessários com o modo provisionado para fluxos de dados do Kinesis:

- Essa fórmula ajuda a estimar o número de fragmentos que serão necessários para acomodar seus registros de dados de alterações do DynamoDB. Ela não representa o número total de fragmentos necessários no fluxo de dados do Kinesis, como o número de fragmentos necessários para oferecer suporte aos consumidores adicionais de fluxos de dados do Kinesis.
- Ainda será possível ocorrer exceções de throughput de leitura e gravação no modo provisionado se o fluxo de dados não for configurado para lidar com a throughput máxima. Nesse caso, será preciso escalar manualmente o fluxo para acomodar o tráfego de dados.
- Essa fórmula leva em consideração o inchaço adicional gerado pelo DynamoDB antes de fazer streaming dos registros de dados dos logs de alterações para o fluxo de dados do Kinesis.

Para saber mais sobre os modos de capacidade no Kinesis Data Streams, consulte Choosing the Data Stream Capacity Mode. Para saber mais sobre a diferença de preços entre os diferentes modos de capacidade, consulte Preços do Amazon Kinesis Data Streams.

Monitorar a captura de dados de alterações com o Kinesis Data Streams

O DynamoDB fornece várias métricas do Amazon CloudWatch que ajudam a monitorar a replicação da captura de dados de alterações para o Kinesis. Para obter uma lista completa de métricas do CloudWatch, consulte Métricas e dimensões do DynamoDB.

Para determinar se o fluxo tem capacidade suficiente, recomendamos monitorar os seguintes itens durante a ativação do fluxo e na produção:

- `ThrottledPutRecordCount`: o número de registros aos quais o fluxo de dados do Kinesis aplicou controle de utilização devido à capacidade insuficiente do fluxo de dados do Kinesis. O `ThrottledPutRecordCount` deve permanecer o mais baixo possível, embora você possa sentir algum controle de utilização durante picos excepcionais de uso. O DynamoDB tenta novamente enviar os registros limitados ao fluxo de dados do Kinesis, mas isso pode resultar em maior latência de replicação.

  Se você perceber controle de utilização excessivo e regular, talvez seja necessário aumentar o número de fragmentos de fluxos do Kinesis proporcionalmente ao throughput de gravação observada da tabela. Para saber mais sobre a determinação do tamanho de um fluxo de dados do Kinesis, consulte Como determinar o tamanho inicial de um fluxo de dados do Kinesis.
- `AgeOfOldestUnreplicatedRecord`: o tempo decorrido desde a que alteração de nível de item mais antiga ainda a ser replicada para o fluxo de dados do Kinesis surgiu na tabela do DynamoDB. Em condições normais de operação, `AgeOfOldestUnreplicatedRecord` deve estar na ordem dos milissegundos. Esse número aumenta de acordo com as tentativas de replicação malsucedidas quando elas são causadas por opções de configuração controladas pelo cliente.

  Se a métrica `AgeOfOldestUnreplicatedRecord` exceder 168 horas, a replicação das alterações no nível do item da tabela do DynamoDB para o fluxo de dados do Kinesis será automaticamente desabilitada.

  São exemplos de configurações controladas pelo cliente que levam a tentativas de replicação malsucedidas: uma capacidade de fluxo de dados do Kinesis com provisionamento insuficiente, o que leva a controle de utilização excessivo, ou uma atualização manual das políticas de acesso do fluxo de dados do Kinesis que impede que o DynamoDB adicione dados ao fluxo de dados. Para manter essa métrica no mínimo possível, talvez seja necessário garantir o provisionamento correto da capacidade do fluxo de dados do Kinesis e que as permissões do DynamoDB permaneçam inalteradas.

- `FailedToReplicateRecordCount`: número de registros que o DynamoDB não conseguiu replicar para o fluxo de dados do Kinesis. Determinados itens com mais de 34 KB podem se expandir em tamanho para alterar registros de dados maiores que o limite de 1 MB para itens do Kinesis Data Streams. Essa expansão de tamanho ocorre quando os itens com mais de 34 KB contêm um grande número de valores de atributos booleanos ou vazios. Valores de atributos booleanos e vazios são armazenados como 1 byte no DynamoDB, mas expandem até 5 bytes quando são serializados usando JSON padrão para replicação do Kinesis Data Streams. O DynamoDB não consegue replicar esses registros de alteração para o fluxo de dados do Kinesis. O DynamoDB ignora esses registros de dados de alteração e continua automaticamente a replicar registros subsequentes.

Você pode criar alarmes do Amazon CloudWatch que enviam uma mensagem do Amazon Simple Notification Service (Amazon SNS) para notificação quando qualquer uma das métricas anteriores exceder um limite específico.

Usar políticas do IAM para o Amazon Kinesis Data Streams e Amazon DynamoDB

Na primeira vez que você habilitar o Amazon Kinesis Data Streams para Amazon DynamoDB, o DynamoDB criará automaticamente uma função vinculada ao serviço do AWS Identity and Access Management (IAM) para você. Esta função, `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`, permite que o DynamoDB gerencie a replicação de alterações em nível de item no Kinesis Data Streams em seu nome. Não exclua essa função vinculada ao serviço.

Para obter mais informações sobre funções vinculadas a serviços, consulte Usando funções vinculadas a serviços no Guia do Usuário do IAM.

> **Note**
>
> O DynamoDB não oferece suporte a condições baseadas em etiqueta para políticas do IAM.

Para habilitar o Amazon Kinesis Data Streams para o Amazon DynamoDB, é necessário ter as seguintes permissões na tabela:

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`

- `kinesis:PutRecords`
- `kinesis:DescribeStream`

A fim de descrever o Amazon Kinesis Data Streams para Amazon DynamoDB para determinada tabela do DynamoDB, é necessário ter as permissões a seguir na tabela.

- `dynamodb:DescribeKinesisStreamingDestination`
- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

A fim de desabilitar o Amazon Kinesis Data Streams para o Amazon DynamoDB, é necessário ter permissões a seguir na tabela.

- `dynamodb:DisableKinesisStreamingDestination`

Para atualizar o Amazon Kinesis Data Streams para o Amazon DynamoDB, é necessário ter as permissões a seguir na tabela.

- `dynamodb:UpdateKinesisStreamingDestination`

Os exemplos a seguir mostram como usar políticas do IAM para conceder permissões para o Amazon Kinesis Data Streams para Amazon DynamoDB.

Exemplo: habilitar o Amazon Kinesis Data Streams para Amazon DynamoDB

A política do IAM a seguir concede permissões para habilitar o Amazon Kinesis Data Streams para o Amazon DynamoDB na tabela `Music`: Ela não concede permissões para desabilitar, atualizar ou descrever o Kinesis Data Streams para o DynamoDB na tabela `Music`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::*:role/aws-
service-role/kinesisreplication.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBKinesisDataStreamsReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "kinesisreplication.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Exemplo: Atualizar o Amazon Kinesis Data Streams para o Amazon DynamoDB

A política do IAM a seguir concede permissões para atualizar o Amazon Kinesis Data Streams para o Amazon DynamoDB na tabela `Music`: Ela não concede permissões para habilitar, desabilitar ou descrever o Amazon Kinesis Data Streams para o Amazon DynamoDB na tabela `Music`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Exemplo: desabilitar o Amazon Kinesis Data Streams para Amazon DynamoDB

A política do IAM a seguir concede permissões para desabilitar o Amazon Kinesis Data Streams para o Amazon DynamoDB na tabela `Music`: Ela não concede permissões para habilitar, atualizar ou descrever o Amazon Kinesis Data Streams para o Amazon DynamoDB na tabela `Music`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Exemplo: aplicar seletivamente permissões para o Amazon Kinesis Data Streams para Amazon DynamoDB com base no recurso

A política do IAM a seguir concede permissões para habilitar e descrever o Amazon Kinesis Data Streams para o Amazon DynamoDB na tabela `Music` e nega permissões para desabilitar o Amazon Kinesis Data Streams para o Amazon DynamoDB na tabela `Orders`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
```

```
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Orders"
        }
    ]
}
```

Usar funções vinculadas ao serviço para o Kinesis Data Streams para DynamoDB

O Amazon Kinesis Data Streams para Amazon DynamoDB usa funções vinculadas ao serviço do AWS Identity and Access Management (IAM). A função vinculada ao serviço é um tipo exclusivo de perfil do IAM vinculada diretamente ao Kinesis Data Streams para DynamoDB. As funções vinculadas a serviços são predefinidas pelo Kinesis Data Streams para DynamoDB e incluem todas as permissões que o serviço requer para chamar outros serviços da AWS em seu nome.

Uma função vinculada ao serviço facilita a configuração do Kinesis Data Streams para DynamoDB porque você não precisa adicionar as permissões necessárias manualmente. O Kinesis Data Streams para DynamoDB define as permissões de suas funções vinculadas a serviços e, exceto se definido de outra forma, somente o Kinesis Data Streams para DynamoDB pode assumir suas funções. As permissões definidas incluem a política de confiança e a política de permissões, que não pode ser anexada a nenhuma outra entidade do IAM.

Para obter informações sobre outros serviços compatíveis com funções vinculadas a serviços, consulte Serviços da AWS compatíveis com o IAM e procure os serviços que contenham Sim na coluna Função vinculada ao serviço. Escolha um Sim com um link para visualizar a documentação da função vinculada a esse serviço.

Permissões de função vinculada ao serviço para o Kinesis Data Streams para DynamoDB

O Kinesis Data Streams para DynamoDB usa a função vinculada ao serviço chamada AWSServiceRoleForDynamoDBKinesisDataStreamsReplication. O objetivo dessa função vinculada ao serviço é permitir que o Amazon DynamoDB gerencie a replicação de alterações em nível de item no Kinesis Data Streams em seu nome.

O perfil vinculado ao serviço `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` confia nos seguintes serviços para aceitar o perfil:

- `kinesisreplication.dynamodb.amazonaws.com`

A política de permissões da função permite que o Kinesis Data Streams for DynamoDB conclua as seguintes ações nos recursos especificados:

- Ação: `Put records and describe` em `Kinesis stream`
- Ação: `Generate data keys` no `AWS KMS` para colocar dados em fluxos do Kinesis que são criptografados usando chaves do AWS KMS geradas pelo usuário.

Para obter o conteúdo exato do documento de política, consulte DynamoDBKinesisReplicationServiceRolePolicy.

Você deve configurar permissões para que uma entidade do IAM (por exemplo, um usuário, grupo ou função) crie, edite ou exclua um perfil vinculado a serviço. Para mais informações, consulte Permissões de perfil vinculado ao serviço no Guia do usuário do IAM.

Criar uma função vinculada ao serviço para o Kinesis Data Streams para DynamoDB

Não é necessário criar manualmente uma função vinculada ao serviço. Quando você habilita o Kinesis Data Streams para DynamoDB no AWS Management Console, na AWS CLI ou na API da AWS, o Kinesis Data Streams para DynamoDB cria a função vinculada ao serviço para você.

Se excluir essa função vinculada ao serviço e precisar criá-la novamente, você poderá usar esse mesmo processo para recriar a função em sua conta. Quando você habilita o Kinesis Data Streams para DynamoDB, o Kinesis Data Streams para DynamoDB cria a função vinculada ao serviço para você.

Editar uma função vinculada ao serviço para o Kinesis Data Streams para DynamoDB

O Kinesis Data Streams para DynamoDB não permite editar a função vinculada ao serviço `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Depois que criar um perfil vinculado ao serviço, você não poderá alterar o nome do perfil, pois várias entidades podem fazer referência a ele. No entanto, será possível editar a descrição do perfil usando o IAM. Para obter mais informações, consulte Editar uma função vinculada a serviço no Guia do usuário do IAM.

Excluir uma função vinculada ao serviço para o Kinesis Data Streams para DynamoDB

Também é possível usar o console do IAM, a AWS CLI ou a API da AWS para excluir manualmente a função vinculada ao serviço. Para isso, primeiro você deve limpar manualmente os recursos de sua função vinculada ao serviço e depois excluí-la manualmente.

> **Note**
>
> Se o serviço Kinesis Data Streams para DynamoDB estiver usando a função quando você tenta excluir os recursos, a exclusão poderá falhar. Se isso acontecer, espere alguns minutos e tente a operação novamente.

Como excluir manualmente o perfil vinculado ao serviço usando o IAM

Use o console do IAM, a AWS CLI ou a API da AWS para excluir a função vinculada ao serviço `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Para obter mais informações, consulte Excluir uma função vinculada ao serviço no Guia do usuário do IAM.

Capturar dados de alterações para o DynamoDB Streams

O DynamoDB Streams captura uma sequência em ordem temporal de modificações em nível de item em qualquer tabela do DynamoDB e armazena essas informações em um log por até 24 horas. As aplicações podem acessar esse log e visualizar os itens de dados à medida que eles aparecem antes e depois de serem modificados, quase em tempo real.

A criptografia em repouso criptografa os dados em fluxos do DynamoDB. Para obter mais informações, consulte Criptografia em repouso do DynamoDB.

Um fluxo do DynamoDB é um fluxo ordenado de informações sobre alterações em itens de uma tabela do DynamoDB. Quando você habilita um fluxo em uma tabela, o DynamoDB captura informações sobre todas as modificações em itens de dados na tabela.

Sempre que uma aplicação cria, atualiza ou exclui itens nessa tabela, o DynamoDB Streams grava um registro de fluxo com os atributos de chave primária dos itens que foram modificados. Um registro de fluxo contém informações sobre uma modificação de dados em um único item de uma tabela do DynamoDB. É possível configurar o stream de modo que os registros de stream capturem informações adicionais, como as imagens "antes" e "depois" de itens modificados.

O DynamoDB Streams ajuda a garantir:

- Cada registro do fluxo aparece exatamente uma vez no fluxo.
- Para cada item modificado em uma tabela do DynamoDB, os registros de fluxo aparecem na mesma sequência que as modificações reais no item.

O DynamoDB Streams grava registros de fluxo em tempo quase real para que você possa criar aplicações que consomem esses fluxos e executam ações com base no conteúdo.

Tópicos

- Endpoints para DynamoDB Streams
- Habilitar um fluxo
- Ler e processar um fluxo
- DynamoDB Streams e vida útil
- Usar o adaptador do DynamoDB Streams Kinesis Adapter para processar registros de fluxos
- API de baixo nível do DynamoDB Streams: exemplo em Java
- DynamoDB Streams e acionadores do AWS Lambda
- DynamoDB Streams e Apache Flink

Endpoints para DynamoDB Streams

O AWS mantém endpoints separados para fluxos do DynamoDB e DynamoDB Streams. Para trabalhar com índices e tabelas de banco de dados, sua aplicação precisa acessar um endpoint do DynamoDB. Para ler e processar registros do DynamoDB Streams, sua aplicação precisa acessar um endpoint do DynamoDB Streams na mesma região.

A convenção de nomenclatura dos endpoints do DynamoDB Streams é `streams.dynamodb.<region>.amazonaws.com`. Por exemplo, se você usasse o endpoint `dynamodb.us-west-2.amazonaws.com` para acessar o DynamoDB, usaria o endpoint `streams.dynamodb.us-west-2.amazonaws.com` para acessar o DynamoDB Streams.

> **Note**
>
> Para obter uma lista completa de regiões e endpoints do DynamoDB e do DynamoDB Streams, consulte Regiões e endpoints na Referência geral da AWS.

Os AWS SDKs fornecem clientes separados para o DynamoDB e o DynamoDB Streams. Dependendo das suas necessidades, sua aplicação pode acessar um endpoint do DynamoDB, um endpoint do DynamoDB Streams ou ambos ao mesmo tempo. Para conectar aos dois endpoints, a aplicação precisa instanciar dois clientes, um para o DynamoDB e outro para o DynamoDB Streams.

Habilitar um fluxo

Você pode habilitar um stream em uma nova tabela ao criá-la usando a AWS CLI ou um dos AWS SDKs. Também pode habilitar ou desabilitar um stream em uma tabela existente ou alterar as configurações de um stream. O DynamoDB Streams opera de forma assíncrona e, portanto, não haverá impacto sobre a performance de uma tabela se você habilitar um stream.

A maneira mais fácil de gerenciar o DynamoDB Streams é usar o AWS Management Console.

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel do console do DynamoDB, escolha Tables (Tabelas) e selecione uma tabela existente.
3. Escolha a guia Exports and streams (Exportações e fluxos).
4. Na seção Detalhes do stream do DynamoDB, escolha Ativar.
5. Na página Ativar fluxo do DynamoDB, escolha as informações que serão gravadas no fluxo sempre que os dados da tabela forem modificados:

   - Key attributes only (Somente atributos de chaves): somente os atributos de chaves do item modificado.
   - New image (Nova imagem): o item inteiro como é exibido depois de modificado.
   - Old image (Imagem antiga): o item inteiro como era exibido antes de modificado.
   - New and old images (Imagens nova e antiga): as imagens nova e antiga do item.

   Quando estiver de acordo com as configurações, escolha Ativar fluxo.
6. (Opcional) Para desabilitar um fluxo existente, escolha Desativar em Detalhes do stream do DynamoDB.

Você também pode usar as operações da API `CreateTable` ou `UpdateTable` para habilitar ou modificar um fluxo. O parâmetro `StreamSpecification` determina como o fluxo é configurado:

- `StreamEnabled`: especifica se um fluxo está habilitado (`true`) ou desabilitado (`false`) para a tabela.
- `StreamViewType`: especifica as informações que serão gravadas no fluxo sempre que os dados na tabela forem modificados:
  - `KEYS_ONLY`: somente os atributos de chaves do item modificado.
  - `NEW_IMAGE`: o item inteiro como é exibido depois de ser modificado.
  - `OLD_IMAGE`: o item inteiro como era exibido antes de ser modificado.
  - `NEW_AND_OLD_IMAGES`: as imagens nova e antiga do item.

É possível habilitar ou desabilitar um fluxo a qualquer momento. No entanto, você receberá uma `ValidationException` se tentar habilitar um fluxo em uma tabela que já tenha um fluxo. Você também receberá uma `ValidationException` se tentar desabilitar um fluxo em uma tabela que não tenha um fluxo.

Quando você define `StreamEnabled` como `true`, o DynamoDB cria um novo fluxo com um descritor de streaming exclusivo atribuído a ele. Se você desabilitar e reabilitar um fluxo na tabela, será criado um fluxo com um descritor diferente.

Cada fluxo é identificado exclusivamente por um nome do recurso da Amazon (ARN). Veja a seguir um ARN de exemplo para um fluxo em uma tabela do DynamoDB chamada `TestTable`.

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

Para determinar o descritor de streaming mais recente de uma tabela, emita uma solicitação `DescribeTable` do DynamoDB e procure o elemento `LatestStreamArn` na resposta.

> **Note**
>
> Não é possível editar um `StreamViewType` após a configuração de um fluxo. Se você precisar fazer alterações em um fluxo depois que ele tiver sido configurado, será necessário desativar o fluxo atual e criar outro.

Ler e processar um fluxo

Para ler e processar um fluxo, sua aplicação deve se conectar a um endpoint do DynamoDB Streams e emitir solicitações de API.

Um fluxo consiste em registros de fluxo. Cada registro de stream representa uma única modificação de dados na tabela do DynamoDB à qual o fluxo pertence. Cada registro de fluxo recebe um número de sequência, refletindo a ordem em que ele foi publicado no fluxo.

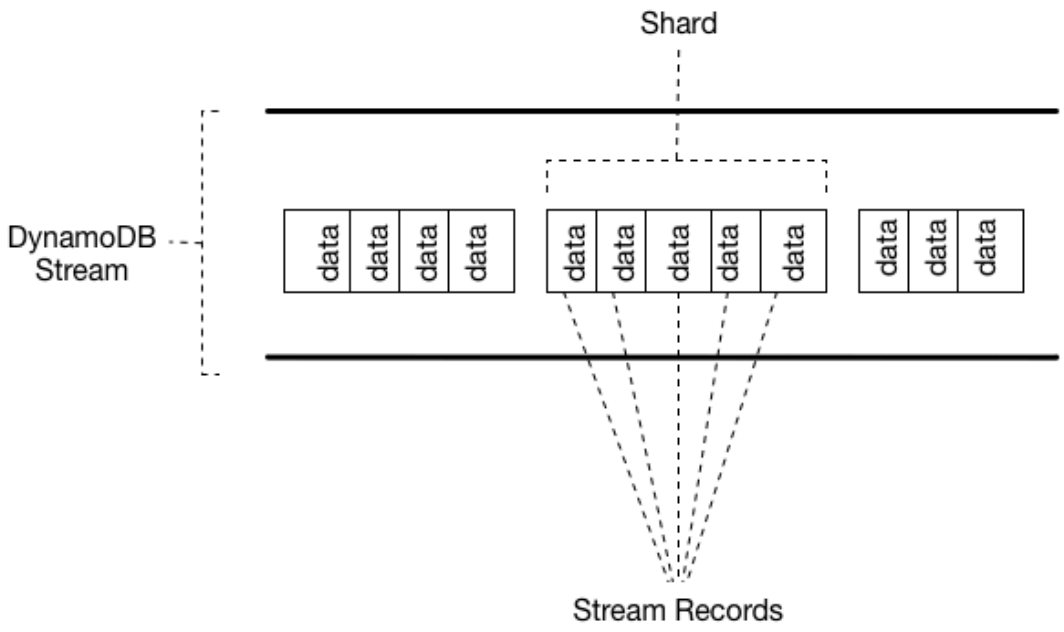
Os registros de fluxo estão organizados em grupos ou fragmentos. Cada fragmento atua como um contêiner para vários registros de fluxo e contém informações necessárias para acessar esses registros e fazer iterações neles. Os registros de fluxo em um fragmento são removidos automaticamente depois de 24 horas.

Fragmentos são efêmeros: eles são criados e excluídos automaticamente, conforme necessário. Qualquer fragmento também pode ser dividido em vários novos fragmentos, o que também ocorre automaticamente. (Também é possível que um fragmento pai tenha apenas um fragmento filho.) Um fragmento pode ser dividido em resposta a altos níveis de atividades de gravação em sua tabela principal e, portanto, as aplicações podem processar registros de vários fragmentos em paralelo.

Se você desabilitar um fluxo, todos os fragmentos que estiverem abertos serão fechados. Os dados no stream continuarão legíveis por 24 horas.

Como fragmentos têm uma linhagem (pai e filhos), uma aplicação sempre deverá processar um fragmento pai antes de um fragmento filho. Isso ajuda a garantir que os registros de fluxo também sejam processados na ordem correta. (Se você usa o DynamoDB Streams Kinesis Adapter, isso será feito para você. Sua aplicação processará os fragmentos e os registros de fluxo na ordem correta. Ela manipulará automaticamente fragmentos novos ou expirados, bem como fragmentos que são divididos enquanto a aplicação está em execução. Para obter mais informações, consulte Usar o adaptador do DynamoDB Streams Kinesis Adapter para processar registros de fluxos.)

O diagrama a seguir mostra a relação entre um fluxo, os fragmentos nesse fluxo e os registros de fluxo nesses fragmentos.



> **Note**
>
> Se você executar uma operação `PutItem` ou `UpdateItem` que não altere nenhum dado em um item, o DynamoDB Streams não gravará um registro de fluxo para essa operação.

Para acessar um fluxo e processar os registros de fluxo dentro dele, você deve fazer o seguinte:

- Determinar o ARN exclusivo do fluxo que deseja acessar.
- Determinar quais fragmentos no fluxo contêm os registros de fluxo desejados.
- Acessar os fragmentos e recuperar os registros de fluxo desejados.

> **Note**
>
> No máximo dois processos devem estar lendo simultaneamente do mesmo fragmento de fluxo. Ter mais de dois leitores por fragmento pode resultar em controle de utilização.

A API do DynamoDB Streams fornece as seguintes ações para uso por programas de aplicações:

- `ListStreams`: retorna uma lista de descritores de fluxos para a conta e o endpoint atuais. Opcionalmente, você pode solicitar apenas os descritores de fluxo de um nome de tabela específico.
- `DescribeStream`: retorna informações sobre um streaming, incluindo o status atual do streaming, o Nome de recurso da Amazon (ARN), a composição de seus estilhaços e sua tabela correspondente do DynamoDB Você pode opcionalmente usar o campo `ShardFilter` para recuperar o fragmento filho existente associado ao fragmento pai.
- `GetShardIterator`: retorna um iterador de fragmentos que descreve um local dentro de um fragmento. Você pode solicitar que o iterador forneça acesso ao ponto mais antigo, o ponto mais novo ou um ponto específico no fluxo.
- `GetRecords`: retorna os registros do fluxo de um determinado fragmento. Você deve fornecer o iterador de fragmentos retornado de uma solicitação `GetShardIterator`.

Para obter descrições completas dessas operações da API, incluindo solicitações e respostas de exemplo, consulte a Referência da API do Amazon DynamoDB Streams.

Descoberta de fragmentos

Descubra novos fragmentos em seu stream do DynamoDB com dois métodos poderosos. Como usuário do Amazon DynamoDB Streams, você tem duas maneiras eficazes de rastrear e identificar novos fragmentos:

Pesquisando toda a topologia do stream

Use a API `DescribeStream` para pesquisar regularmente o stream. Isso retorna todos os fragmentos no stream, incluindo novos fragmentos que tenham sido criados. Ao comparar os resultados ao longo do tempo, você pode detectar fragmentos recém-adicionados.

Descobrindo fragmentos filhos

Use a API `DescribeStream` com o parâmetro `ShardFilter` para encontrar um subconjunto de fragmentos. Ao especificar um fragmento pai na solicitação, o DynamoDB Streams retornará seus fragmentos filhos imediatos. Essa abordagem é útil quando você só precisa rastrear a linhagem do fragmento sem escanear todo o stream.

Os aplicativos que consomem dados do DynamoDB Streams podem fazer a transição eficiente da leitura de um fragmento fechado para seu fragmento filho usando esse parâmetro `ShardFilter`, evitando chamadas repetidas à API `DescribeStream` para recuperar e percorrer o mapa de

fragmentos de todos os fragmentos fechados e abertos. Isso ajuda a descobrir rapidamente os fragmentos filhos após o fechamento de um fragmento pai, tornando seus aplicativos de processamento de stream mais responsivos e econômicos.

Ambos os métodos permitem que você fique por dentro da estrutura em evolução do seu DynamoDB Streams, garantindo que você nunca perca atualizações críticas de dados ou modificações de fragmentos.

Limite de retenção de dados para o DynamoDB Streams

Todos os dados no DynamoDB Streams estão sujeitos a um tempo de vida de 24 horas. É possível recuperar e analisar as atividades das últimas 24 horas de qualquer tabela. No entanto, os dados mais antigos que 24 horas estão suscetíveis a remoção a qualquer momento.

Se você desabilitar um fluxo em uma tabela, os dados nesse fluxo continuarão legíveis por 24 horas. Depois desse tempo, os dados expirarão, e os registros de fluxo serão excluídos automaticamente. Não há nenhum mecanismo para excluir manualmente um fluxo existente. Aguarde até que o limite de retenção expire (24 horas), e todos os registros do fluxo sejam excluídos.

DynamoDB Streams e vida útil

Você pode fazer backup ou processar os itens excluídos por vida útil (TTL) habilitando o Amazon DynamoDB Streams na tabela e processando os registros de fluxos dos itens expirados. Para obter mais informações, consulte Ler e processar um fluxo.

O registro de fluxos contém um campo de identidade do usuário `Records[<index>].userIdentity`.

Os itens excluídos pelo processo de vida útil após a expiração têm os seguintes campos:

- `Records[<index>].userIdentity.type`

  `"Service"`

- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> **Note**
>
> Quando você usa o TTL em uma tabela global, a região em que o TTL foi executado terá o campo `userIdentity` definido. Esse campo não será definido em outras regiões quando a exclusão for replicada.

O JSON a seguir mostra a parte relevante de um único registro de fluxos.

```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

Usar o DynamoDB Streams e o Lambda para arquivar itens excluídos do TTL

Combinar a vida útil (TTL) do DynamoDB, o DynamoDB Streams e o AWS Lambda pode ajudar a simplificar os dados de arquivo, reduzir os custos de armazenamento do DynamoDB e diminuir a complexidade do código. O uso do Lambda como consumidor de fluxo oferece muitas vantagens, principalmente a redução de custos em comparação com outros consumidores, como a Kinesis Client Library (KCL). Você não é cobrado por chamadas de API `GetRecords` no fluxo do DynamoDB ao usar o Lambda para consumir eventos, e o Lambda pode fornecer filtragem de eventos por meio da identificação de padrões JSON em um evento de fluxo. Com a filtragem de conteúdo de padrão de evento, é possível definir até cinco filtros diferentes para controlar quais eventos são enviados ao Lambda para processamento. Isso ajuda a reduzir as invocações de suas funções do Lambda, simplifica o código e diminui o custo geral.

Embora o DynamoDB Streams contenha todas as modificações de dados, como as ações `Create`, `Modify` e `Remove`, isso pode resultar em invocações indesejadas da função do Lambda de arquivo. Por exemplo, digamos que você tenha uma tabela com 2 milhões de modificações de dados por hora

fluindo para o fluxo, mas menos de 5% delas são exclusões de itens que expirarão no processo de TTL e precisam ser arquivadas. Com filtros de origem de eventos do Lambda, a função do Lambda invocará apenas 100 mil vezes por hora. O resultado da filtragem de eventos é que você é cobrado apenas pelas invocações necessárias, e não pelos 2 milhões de invocações que você teria sem a filtragem de eventos.

A filtragem de eventos é aplicada ao mapeamento da origem de eventos do Lambda, que é um recurso que lê com base em um evento escolhido (o fluxo do DynamoDB) e invoca uma função do Lambda. No diagrama a seguir, você pode ver como um item excluído por vida útil é consumido por uma função do Lambda usando fluxos e filtros de eventos.

Padrão de filtro de eventos da vida útil do DynamoDB

A adição do JSON a seguir aos critérios de filtro do mapeamento de origem de eventos permite que você invoque sua função do Lambda somente para itens excluídos por TTL:

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

Crie um mapeamento da origem de eventos no AWS Lambda.

Use os trechos de código a seguir para criar um mapeamento de origem de eventos filtrado que você possa conectar ao fluxo do DynamoDB de uma tabela. Cada bloco de código inclui o padrão de filtro de eventos.

AWS CLI

```
aws lambda create-event-source-mapping \
--event-source-arn 'arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000' \
```

```
--batch-size 10 \
--enabled \
--function-name test_func \
--starting-position LATEST \
--filter-criteria '{"Filters": [{"Pattern": "{\"userIdentity\":{\"type\":[\"Service
\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}"}]}'
```

Java

```
LambdaClient client = LambdaClient.builder()
        .region(Region.EU_WEST_1)
        .build();

Filter userIdentity = Filter.builder()
        .pattern("{\"userIdentity\":{\"type\":[\"Service\"],\"principalId\":
[\"dynamodb.amazonaws.com\"]}}")
        .build();

FilterCriteria filterCriteria = FilterCriteria.builder()
        .filters(userIdentity)
        .build();

CreateEventSourceMappingRequest mappingRequest =
 CreateEventSourceMappingRequest.builder()
        .eventSourceArn("arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000")
        .batchSize(10)
        .enabled(Boolean.TRUE)
        .functionName("test_func")
        .startingPosition("LATEST")
        .filterCriteria(filterCriteria)
        .build();

try{
    CreateEventSourceMappingResponse eventSourceMappingResponse =
 client.createEventSourceMapping(mappingRequest);
    System.out.println("The mapping ARN is
 "+eventSourceMappingResponse.eventSourceArn());

}catch (ServiceException e){
    System.out.println(e.getMessage());
}
```

Node

```
const client = new LambdaClient({ region: "eu-west-1" });

const input = {
    EventSourceArn: "arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000",
    BatchSize: 10,
    Enabled: true,
    FunctionName: "test_func",
    StartingPosition: "LATEST",
    FilterCriteria: { "Filters": [{ "Pattern": "{\"userIdentity\":{\"type\":
[\"Service\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}" }] }
}

const command = new CreateEventSourceMappingCommand(input);

try {
    const results = await client.send(command);
    console.log(results);
} catch (err) {
    console.error(err);
}
```

Python

```
session = boto3.session.Session(region_name = 'eu-west-1')
client = session.client('lambda')

try:
    response = client.create_event_source_mapping(
        EventSourceArn='arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000',
        BatchSize=10,
        Enabled=True,
        FunctionName='test_func',
        StartingPosition='LATEST',
        FilterCriteria={
            'Filters': [
                {
                    'Pattern': "{\"userIdentity\":{\"type\":[\"Service\"],
\"principalId\":[\"dynamodb.amazonaws.com\"]}}"
                },
```

```
            ]
        }
    )
    print(response)
except Exception as e:
    print(e)
```

JSON

```
{
  "userIdentity": {
     "type": ["Service"],
     "principalId": ["dynamodb.amazonaws.com"]
   }
}
```

Usar o adaptador do DynamoDB Streams Kinesis Adapter para processar registros de fluxos

Usar o Amazon Kinesis Adapter é a forma recomendada para consumir fluxos do Amazon DynamoDB. A API do DynamoDB Streams é intencionalmente semelhante à do Kinesis Data Streams. Em ambos os serviços, os fluxos de dados são compostos de fragmentos, os quais são contêineres de registros de stream. A API de ambos os serviços que contém a API do DynamoDB Streams é intencionalmente semelhante à do Kinesis Data Streams. Em ambos os serviços, os fluxos de dados são compostos de fragmentos, os quais são contêineres de registros de stream. Ambas as APIs de serviços contêm as operações `ListStreams`, `DescribeStream`, `GetShards` e `GetShardIterator`. (Embora essas ações do DynamoDB Streams sejam semelhantes às suas equivalentes no Kinesis Data Streams, elas não são 100% idênticas.)

Como um cliente do DynamoDB Streams, você pode utilizar os padrões de design encontrados no Kinesis Client Library (KCL) para processar fragmentos e registros de fluxos do DynamoDB Streams. Para isso, você pode usar o DynamoDB Streams Kinesis Adapter. O Kinesis Adapter implementa a interface do Kinesis Data Streams para que a KCL possa ser usada para consumir e processar registros do DynamoDB Streams. Para obter instruções sobre como configurar e instalar o DynamoDB Streams Kinesis Adapter, consulte o repositório do GitHub.

Como um usuário do DynamoDB Streams, você pode utilizar os padrões de design encontrados no KCL para processar fragmentos e registros de fluxos do DynamoDB Streams. Para isso, você pode usar o DynamoDB Streams Kinesis Adapter. O Kinesis Adapter implementa a interface do Kinesis

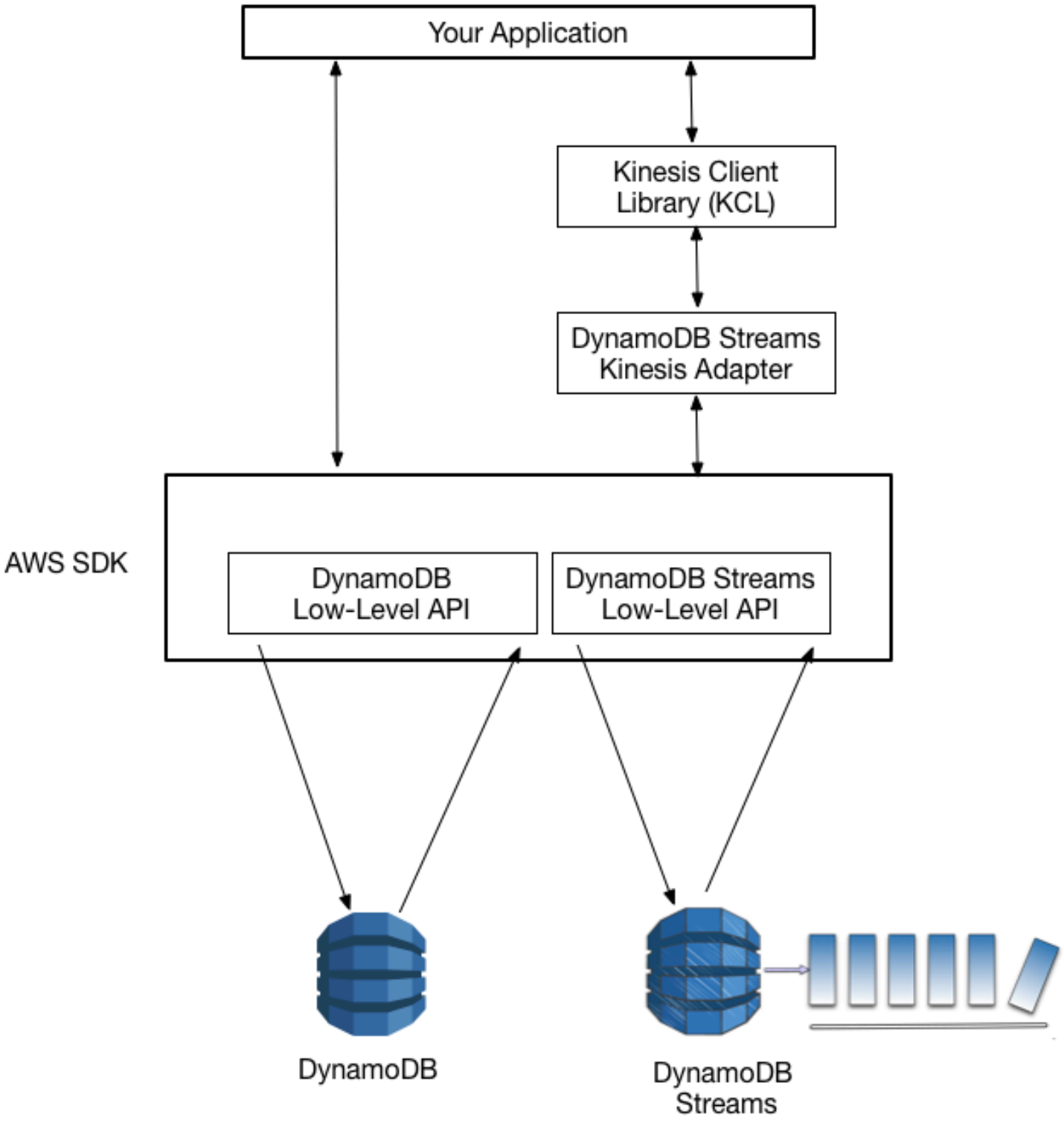
Data Streams para que a KCL possa ser usada para consumir e processar registros do DynamoDB Streams. Para obter instruções sobre como configurar e instalar o DynamoDB Streams Kinesis Adapter, consulte o repositório do GitHub.

Você pode escrever aplicações para Kinesis Data Streams usando a Kinesis Client Library (KCL). A KCL simplifica a codificação fornecendo abstrações úteis acima da API de baixo nível do Kinesis Data Streams. Para obter mais informações sobre a KCL, consulte Desenvolver consumidores usando a biblioteca de clientes do Kinesis no Guia do desenvolvedor do Amazon Kinesis Data Streams Developer Guide.

O DynamoDB recomenda usar a KCL versão 3.x com o AWS SDK para Java v2.x. Como previsto, durante o período de transição a versão atual da KCL 1.x com o AWS SDK para Java v1.x continuará a receber suporte total ao longo do ciclo de vida, de acordo com o estabelecido em AWS SDKs and Tools maintenance policy.

> **Note**
>
> As versões 1.x e 2.x da Amazon Kinesis Client Library (KCL) estão desatualizadas. A KCL 1.x chegará ao fim do suporte em 30 de janeiro de 2026. É altamente recomendável que você migre suas aplicações de KCL que usam a versão 1.x para a versão mais recente da KCL antes de 30 de janeiro de 2026. Para encontrar a versão mais recente da KCL, consulte a página da Amazon Kinesis Client Library no GitHub. Para ter mais informações sobre as versões mais recentes da KCL, consulte Use Kinesis Client Library.. Para obter mais informações sobre como migrar da KCL 1.x para a KCL 3.x, consulte Migração da KCL 1.x para a KCL 3.x .

O diagrama a seguir mostra como essas bibliotecas interagem entre si.



Com o DynamoDB Streams Kinesis Adapter implementado, você pode começar a desenvolver a interface da KCL com as chamadas de API perfeitamente direcionadas no endpoint do DynamoDB Streams.

Quando a aplicação é iniciada, ela chama a KCL para instanciar um operador. Você deve fornecer ao operador informações de configuração da aplicação, como o descritor do fluxo e as credenciais da AWS, além do nome de uma classe de processador de registro que você fornecer. Como ele executa o código no processador de registro, o operador executa as seguintes tarefas:

- Conecta-se ao stream
- Enumera os fragmentos no fluxo
- Verifica e enumera fragmentos filhos de um fragmento pai fechado dentro do fluxo
- Verifica e enumera fragmentos filhos de um fragmento pai fechado dentro do fluxo
- Coordena as associações do estilhaço com outros operadores (se houver)
- Cria uma instância de um processador de registro para cada fragmento que gerencia
- Extrai registros do fluxo
- Envia os registros ao processador de registros correspondente
- Registros processados pelos pontos de verificação
- Equilibra as associações de estilhaço-operador quando a contagem de instância de operadores muda
- Equilibra as associações de fragmento-operador quando os fragmentos se dividem

> **Note**
>
> Para obter uma descrição dos conceitos da KCL aqui listados, consulte Desenvolver consumidores usando a biblioteca clientes do Kinesis no Guia do desenvolvedor do Amazon Kinesis Data Streams.
> Para obter mais informações sobre como usar fluxos com o AWS Lambda, consulte DynamoDB Streams e acionadores do AWS Lambda

Migrar da KCL 1.x para a KCL 3.x

Visão geral

Este guia oferece instruções para migrar uma aplicação de consumidor da KCL 1.x para a KCL 3.x. Devido a diferenças de arquitetura entre a KCL 1.x e a KCL 3.x, a migração requer a atualização de vários componentes para garantir compatibilidade.

A KCL 1.x usa classes e interfaces diferentes em comparação com o KCL 3.x. Você deve primeiro migrar o processador de registros, a fábrica do processador de registros e as classes de operador para o formato compatível com a KCL 3.x e seguir as etapas de migração da KCL 1.x para a KCL 3.x.

Etapas da migração

Tópicos

- Etapa 1: migrar o processador de registros
- Etapa 2: migrar a fábrica do processador de registros
- Etapa 3: migrar o operador
- Etapa 4: visão geral e recomendações de configuração da KCL 3.x
- Etapa 5: migrar da KCL 2.x para a KCL 3.x

Etapa 1: migrar o processador de registros

Este exemplo mostra um processador de registros implementado para o DynamoDB Streams Kinesis Adapter da KCL 1.x:

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class StreamsRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
```

```
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (Record record : processRecordsInput.getRecords()) {
            String data = new String(record.getData().array(),
 Charset.forName("UTF-8"));
            System.out.println(data);
            if (record instanceof RecordAdapter) {
                // record processing and checkpointing logic
            }
        }
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
}
```

Como migrar a classe RecordProcessor

1. Altere as interfaces de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` e

`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` para `software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordP` da seguinte forma:

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;

import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
```

2. Atualize as instruções de importação para os métodos `initialize` e `processRecords`.

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

// import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
```

3. Substitua o método `shutdownRequested` pelos novos métodos a seguir: `leaseLost`, `shardEnded`, e `shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...) and
 shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
```

```
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

Esta é a versão atualizada da classe de processador de registros:

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor;
```

```
import
 software.amazon.dynamodb.streamsadapter.adapter.DynamoDBStreamsKinesisClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import software.amazon.awssdk.services.dynamodb.model.Record;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput processRecordsInput)
 {
        for (DynamoDBStreamsKinesisClientRecord record: processRecordsInput.records())
            Record ddbRecord = record.getRecord();
            // processing and checkpointing logic for the ddbRecord
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
```

```
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

> **Note**
>
> O DynamoDB Streams Kinesis Adapter agora usa o modelo de registro SDKv2. No SDKv2, objetos `AttributeValue` complexos (`BS`, `NS`, `M`, `L` e `SS`) nunca retornam null. Use os métodos `hasBs()`, `hasNs()`, `hasM()`, `hasL()` e `hasSs()` para verificar se esses valores existem.

Etapa 2: migrar a fábrica do processador de registros

A fábrica do processador de registros é responsável por criar processadores de registro quando uma concessão é realizada. Veja o seguinte exemplo de fábrica da KCL 1.x:

```
package com.amazonaws.codesamples;

import software.amazon.dynamodb.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

Migrar para **`RecordProcessorFactory`**

- Altere a interface implementada de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` para `software.amazon.kinesis.processor.ShardRecordProcessorFactory` da seguinte forma:

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

Change the return signature for createProcessor.

// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Veja o seguinte exemplo de fábrica de processador de registros em 3.0:

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor();
    }
}
```

Etapa 3: migrar o operador

Na versão 3.0 da KCL, uma nova classe, chamada Scheduler, substitui a classe Worker. Veja o seguinte exemplo de operador da KCL 1.x:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
        amazonDynamoDB,
        amazonCloudWatchClient);
```

Para migrar o operador

1. Altere a instrução `import` para a classe `Worker` para as instruções de importação para as classes `Scheduler` e `ConfigsBuilder`.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Importe `StreamTracker` e altere a importação de `StreamsWorkerFactory` para `StreamsSchedulerFactory`.

   ```
   import software.amazon.kinesis.processor.StreamTracker;
   // import software.amazon.dynamodb.streamsadapter.StreamsWorkerFactory;
   import software.amazon.dynamodb.streamsadapter.StreamsSchedulerFactory;
   ```

3. Escolha a posição por meio da qual iniciar a aplicação. Ela pode ser `TRIM_HORIZON` ou `LATEST`.

   ```
   import software.amazon.kinesis.common.InitialPositionInStream;
   import software.amazon.kinesis.common.InitialPositionInStreamExtended;
   ```

4. Crie uma instância de `StreamTracker`.

   ```
   StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
           streamArn,
   ```

```
 InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
);
```

5. Crie o objeto `AmazonDynamoDBStreamsAdapterClient`.

```
import software.amazon.dynamodb.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

...

AwsCredentialsProvider credentialsProvider = DefaultCredentialsProvider.create();

AmazonDynamoDBStreamsAdapterClient adapterClient = new
 AmazonDynamoDBStreamsAdapterClient(
        credentialsProvider, awsRegion);
```

6. Crie o objeto `ConfigsBuilder`.

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. Crie o `Scheduler` usando `ConfigsBuilder` como mostrado no seguinte exemplo:

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;

import software.amazon.kinesis.common.KinesisClientUtil;
```

```
import software.amazon.kinesis.coordinator.Scheduler;

...


DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();


DynamoDBStreamsPollingConfig pollingConfig = new
 DynamoDBStreamsPollingConfig(adapterClient);
pollingConfig.idleTimeBetweenReadsInMillis(idleTimeBetweenReadsInMillis);

// Use ConfigsBuilder to configure settings
RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
retrievalConfig.retrievalSpecificConfig(pollingConfig);

CoordinatorConfig coordinatorConfig = configsBuilder.coordinatorConfig();
coordinatorConfig.clientVersionConfig(CoordinatorConfig.ClientVersionConfig.CLIENT_VERSION_

Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                coordinatorConfig,
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                adapterClient
        );
```

Important

A configuração `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` mantém a compatibilidade entre o DynamoDB Streams Kinesis Adapter para a KCL v3 e a KCL v1, não entre a KCL v2 e a v3.

Etapa 4: visão geral e recomendações de configuração da KCL 3.x

Para obter uma descrição detalhada das configurações introduzidas após a KCL 1.x que são relevantes na KCL 3.x, consulte KCL configurations e KCL migration client configuration.

> **Important**
>
> Em vez de criar objetos diretamente de `checkpointConfig`, `coordinatorConfig`, `leaseManagementConfig`, `metricsConfig`, `processorConfig` e `retrievalConfig`, recomendamos usar `ConfigsBuilder` para definir configurações na KCL 3.x e versões posteriores para evitar problemas de inicialização do Agendador. O `ConfigsBuilder` oferece uma maneira mais flexível e sustentável de configurar sua aplicação da KCL.

Configurações com valor padrão de atualização na KCL 3.x

`billingMode`

Na KCL versão 1.x, o valor padrão para `billingMode` é definido como `PROVISIONED`. No entanto, na KCL versão 3.x, o padrão `billingMode` é `PAY_PER_REQUEST` (modo sob demanda). Recomendamos que você use o modo de capacidade sob demanda em sua tabela de concessões para ajustar automaticamente a capacidade com base no uso. Para obter orientações sobre como usar a capacidade provisionada para suas tabelas de concessões, consulte Best practices for the lease table with provisioned capacity mode.

`idleTimeBetweenReadsInMillis`

Na KCL versão 1.x, o valor padrão para `idleTimeBetweenReadsInMillis` é definido como 1.000 (ou 1 segundo). A KCL versão 3.x define o valor padrão para `idleTimeBetweenReadsInMillis` como 1.500 (ou 1,5 segundo), mas o Amazon DynamoDB Streams Kinesis Adapter substitui esse valor padrão, definindo-o como 1.000 (ou 1 segundo).

Novas configurações na KCL 3.x

`leaseAssignmentIntervalMillis`

Essa configuração define o intervalo de tempo antes que os fragmentos recém-descobertos comecem a ser processados, e é calculada como 1,5 × `leaseAssignmentIntervalMillis`. Se essa configuração não for definida explicitamente, o intervalo de tempo será padronizado como 1,5 × `failoverTimeMillis`. O processamento de novos fragmentos exige a verificação

da tabela de concessões e a consulta a um índice secundário global (GSI) na tabela de concessões. A redução de `leaseAssignmentIntervalMillis` aumenta a frequência dessas operações de verificação e consulta, aumentando os custos do DynamoDB. Recomendamos definir esse valor como 2 mil (ou 2 segundos) para minimizar o atraso no processamento de novos fragmentos.

`shardConsumerDispatchPollIntervalMillis`

Essa configuração define o intervalo entre pesquisas sucessivas feitas pelo consumidor do fragmento para acionar transições de estado. Na KCL versão 1.x, esse comportamento era controlado pelo parâmetro `idleTimeInMillis`, que não era exposto como uma definição configurável. Na KCL versão 3.x, recomendamos definir essa configuração para corresponder ao valor usado em `idleTimeInMillis` na configuração da KCL versão 1.x.

Etapa 5: migrar da KCL 2.x para a KCL 3.x

Para garantir uma transição tranquila e a compatibilidade com a versão mais recente da Kinesis Client Library (KCL), siga as etapas de 5 a 8 nas instruções do guia de migração para atualizar da KCL 2.x para a KCL 3.x.

Para solucionar problemas comuns da KCL 3.x, consulte Troubleshooting KCL consumer applications.

Reverter para a versão anterior da KCL

Este tópico explica como reverter sua aplicação de consumidor para a versão anterior da KCL. O processo de reversão consiste em duas etapas:

1. Execute a Ferramenta de Migração da KCL.
2. Reimplante o código da versão anterior da KCL.

Etapa 1: executar a Ferramenta de Migração da KCL

Quando precisar reverter para a versão anterior da KCL, você deve executar a Ferramenta de Migração da KCL. Essa ferramenta executa duas tarefas importantes:

- Ela remove uma tabela de metadados chamada tabela de métricas do operador e o índice secundário global na tabela de concessões no DynamoDB. Esses artefatos são criados pela KCL 3.x, mas não são necessários quando você reverte para a versão anterior.

- Ela faz com que todos os operadores funcionem em um modo compatível com a KCL 1.x e comecem a usar o algoritmo de balanceamento de carga usado nas versões anteriores da KCL. Se você tiver problemas com o novo algoritmo de balanceamento de carga na KCL 3.x, isso mitigará o problema imediatamente.

> **Important**
>
> A tabela de estados do coordenador no DynamoDB deve existir e não deve ser excluída durante o processo de migração, reversão e avanço.

> **Note**
>
> É importante que todos os operadores em sua aplicação de consumidor usem o mesmo algoritmo de balanceamento de carga em um determinado momento. A Ferramenta de Migração da KCL garante que todos os operadores em sua aplicação de consumidor da KCL 3.x mudem para o modo compatível com a KCL 1.x para que todos os operadores executem o mesmo algoritmo de balanceamento de carga durante a reversão da aplicação para a versão anterior da KCL.

Você pode baixar a Ferramenta de Migração da KCL no diretório de scripts do repositório da KCL no GitHub. Execute o script usando um operador ou host com as permissões apropriadas para gravar na tabela de estados do coordenador, na tabela de métricas do operador e na tabela de concessões. As permissões do IAM apropriadas devem estar configuradas para aplicações de consumidor da KCL. Execute o script uma única vez para cada aplicação da KCL usando o comando especificado:

```
python3 ./KclMigrationTool.py --region region --mode rollback [--
application_name applicationName] [--lease_table_name leaseTableName]
 [--coordinator_state_table_name coordinatorStateTableName] [--
worker_metrics_table_name workerMetricsTableName]
```

Parâmetros

`--region`

Substitua *region* pela Região da AWS.

`--application_name`

Esse parâmetro é obrigatório se você estiver usando nomes padrão para suas tabelas de metadados do DynamoDB (tabela de concessões, tabela de estados do coordenador e tabela de métricas do operador). Se você tiver especificado nomes personalizados para essas tabelas, poderá omitir esse parâmetro. Substitua *`applicationName`* pelo nome da aplicação da KCL. A ferramenta usa esse nome para obter os nomes de tabela padrão se os nomes personalizados não forem fornecidos.

`--lease_table_name`

Esse parâmetro é necessário quando você define um nome personalizado para a tabela de concessões na configuração da KCL. Se você estiver usando o nome padrão da tabela, poderá omitir esse parâmetro. Substitua *`leaseTableName`* pelo nome da tabela personalizada que você especificou para a tabela de concessões.

`--coordinator_state_table_name`

Esse parâmetro é necessário quando você define um nome personalizado para a tabela de estados do coordenador na configuração da KCL. Se você estiver usando o nome padrão da tabela, poderá omitir esse parâmetro. Substitua *`coordinatorStateTableName`* pelo nome da tabela personalizada que você especificou para a tabela de estados do coordenador.

`--worker_metrics_table_name`

Esse parâmetro é necessário quando você define um nome personalizado para a tabela de métricas do operador na configuração da KCL. Se você estiver usando o nome padrão da tabela, poderá omitir esse parâmetro. Substitua *`workerMetricsTableName`* pelo nome da tabela personalizada que você especificou para a tabela de métricas do operador.

Etapa 2: reimplantar o código com a versão anterior da KCL

> **Important**
>
> Qualquer menção à versão 2.x na saída gerada pela Ferramenta de Migração da KCL deve ser interpretada como sendo a versão 1.x da KCL. A execução do script não realiza uma reversão completa, apenas alterna o algoritmo de balanceamento de carga para o usado na versão 1.x da KCL.

Depois de executar a Ferramenta de Migração da KCL para uma reversão, você verá uma destas mensagens:

Mensagem 1

> “Rollback completed. Your application was running 2x compatible functionality. Please rollback to your previous application binaries by deploying the code with your previous KCL version”.
>
> Ação necessária: isso significa que os operadores estavam executando no modo compatível com a KCL 1.x. Reimplante o código para os operadores com a versão anterior da KCL.

Mensagem 2

> “Rollback completed. Your KCL Application was running 3x functionality and will rollback to 2x compatible functionality. If you don't see mitigation after a short period of time, please rollback to your previous application binaries by deploying the code with your previous KCL version”.
>
> Ação necessária: isso significa que os operadores estavam executando no modo da KCL 3.x e a Ferramenta de Migração da KCL mudou todos os operadores para o modo compatível com a KCL 1.x. Reimplante o código para os operadores com a versão anterior da KCL.

Mensagem 3

> “Application was already rolled back. Any KCLv3 resources that could be deleted were cleaned up to avoid charges until the application can be rolled forward with migration”.
>
> Ação necessária: isso significa que os operadores já foram revertidos para execução no modo compatível com a KCL 1.x. Reimplante o código para os operadores com a versão anterior da KCL.

Avançar para a KCL 3.x após uma reversão

Este tópico explica como avançar sua aplicação de consumidor para a KCL 3.x após uma reversão. Quando precisar avançar, você deve concluir um processo de duas etapas:

1. Execute a Ferramenta de Migração da KCL.
2. Implantar o código com a KCL 3.x.

Etapa 1: executar a Ferramenta de Migração da KCL

Execute a Ferramenta de Migração da KCL com o seguinte comando para avançar para a KCL 3.x:

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

Parâmetros

`--region`

Substitua *region* pela Região da AWS.

`--application_name`

Esse parâmetro será obrigatório se você estiver usando nomes padrão para a tabela de estados do coordenador. Se você tiver especificado nomes personalizados para a tabela de estados do coordenador, poderá omitir esse parâmetro. Substitua *applicationName* pelo nome da aplicação da KCL. A ferramenta usa esse nome para obter os nomes de tabela padrão se os nomes personalizados não forem fornecidos.

`--coordinator_state_table_name`

Esse parâmetro é necessário quando você define um nome personalizado para a tabela de estados do coordenador na configuração da KCL. Se você estiver usando o nome padrão da tabela, poderá omitir esse parâmetro. Substitua *coordinatorStateTableName* pelo nome da tabela personalizada que você especificou para a tabela de estados do coordenador.

Após a execução da Ferramenta de Migração no modo de avanço, o KCL cria os seguintes recursos do DynamoDB necessários para a KCL 3.x:

- Um índice secundário global na tabela de concessões
- Uma tabela de métricas do operador

Etapa 2: implantar o código com a KCL 3.x

Depois de executar a Ferramenta de Migração da KCL para um avanço, implante seu código com a KCL 3.x nos operadores. Para concluir sua migração, consulte Step 8: Complete the migration.

Demonstração: DynamoDB Streams Kinesis Adapter

Esta seção é uma demonstração de uma aplicação em Java que usa a Amazon Kinesis Client Library e o Amazon DynamoDB Streams Kinesis Adapter. A aplicação mostra um exemplo de

replicação de dados, no qual as atividades de gravação de uma tabela são aplicadas a uma segunda tabela, com o conteúdo de ambas mantido em sincronia. Para o código-fonte, consulte Programa completo: DynamoDB Streams Kinesis Adapter.

O programa faz o seguinte:

1. Cria duas tabelas do DynamoDB chamadas `KCL-Demo-src` e `KCL-Demo-dst`. Cada uma dessas tabelas tem um fluxo habilitado.
2. Gera atividades de atualização na tabela de origem, adicionando, atualizando e excluindo itens. Isso faz com que os dados sejam gravados no fluxo da tabela.
3. Lê os registros do fluxo, faz a reconstrução desses registros como solicitações do DynamoDB e aplica essas solicitações à tabela de destino.
4. Verifica as tabelas de origem e destino para garantir que o conteúdo seja idêntico.
5. Realiza uma limpeza excluindo as tabelas.

Essas etapas estão descritas nas seções a seguir, e a aplicação completa é mostrada no final do passo-a-passo.

Tópicos

- Etapa 1: criar tabelas do DynamoDB
- Etapa 2: gerar atividades de atualização na tabela de origem
- Etapa 3: processar o fluxo
- Etapa 4: garantir que as duas tabelas tenham conteúdo idêntico
- Etapa 5: limpar
- Programa completo: DynamoDB Streams Kinesis Adapter

Etapa 1: criar tabelas do DynamoDB

A primeira etapa é criar duas tabelas do DynamoDB: uma de origem e outra de destino. O `StreamViewType` no fluxo da tabela de origem é `NEW_IMAGE`. Isso significa que sempre que um item é modificado nessa tabela, o item “depois” da imagem é gravado no fluxo. Dessa forma, o fluxo mantém o controle de todas as atividades de gravação na tabela.

O exemplo a seguir mostra o código usado para a criação das duas tabelas.

```
java.util.List<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

java.util.List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

  // key

ProvisionedThroughput provisionedThroughput = new
 ProvisionedThroughput().withReadCapacityUnits(2L)
    .withWriteCapacityUnits(2L);

StreamSpecification streamSpecification = new StreamSpecification();
streamSpecification.setStreamEnabled(true);
streamSpecification.setStreamViewType(StreamViewType.NEW_IMAGE);
CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions).withKeySchema(keySchema)

 .withProvisionedThroughput(provisionedThroughput).withStreamSpecification(streamSpecification)
```

Etapa 2: gerar atividades de atualização na tabela de origem

O próximo passo é gerar algumas atividades de gravação na tabela de origem. Enquanto essas atividades estão ocorrendo, o fluxo da tabela de origem também é atualizado quase em tempo real.

A aplicação define uma classe auxiliar com métodos que chamam as operações da API `PutItem`, `UpdateItem` e `DeleteItem` para gravar os dados. O exemplo de código a seguir mostra como esses métodos são usados.

```
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "101", "test1");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "101", "test2");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "101");
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "102", "demo3");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "102", "demo4");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "102");
```

Etapa 3: processar o fluxo

Agora, o programa inicia o processamento do fluxo. O DynamoDB Streams Kinesis Adapter atua como uma camada transparente entre a KCL e o endpoint do DynamoDB Streams para que o código possa usar totalmente a KCL em vez de precisar fazer chamadas de baixo nível ao DynamoDB Streams. O programa realiza as seguintes tarefas:

- Ele define uma classe de processador de registro, `StreamsRecordProcessor`, com métodos que estão em conformidade com a definição de interface da KCL: `initialize`, `processRecords` e `shutdown`. O método `processRecords` contém a lógica necessária para leituras do fluxo da tabela de origem e para gravações na tabela de destino.
- Ele define uma fábrica de classes para a classe de processador de registro (`StreamsRecordProcessorFactory`). Isso é necessário para programas Java que usam a KCL.
- Ele instancia um novo `Worker` da KCL, que está associado à fábrica de classes.
- Ele desliga `Worker` quando o processamento do registro é concluído.

Para saber mais sobre a definição da interface da KCL, consulte Desenvolvimento de consumidores usando a Amazon Kinesis Client Library no Guia do desenvolvedor do Amazon Kinesis Data Streams.

O exemplo de código a seguir mostra o loop principal em `StreamsRecordProcessor`. A instrução `case` determina a ação a ser executada, com base no `OperationType` que aparece no registro de fluxo.

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
                software.amazon.dynamodb.model.Record streamRecord = ((RecordAdapter)
 record)
                    .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getNewImage());
```

```
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
    }
    checkpointCounter += 1;
    if (checkpointCounter % 10 == 0) {
        try {
            checkpointer.checkpoint();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Etapa 4: garantir que as duas tabelas tenham conteúdo idêntico

Neste ponto, o conteúdo das tabelas de origem e destino está sincronizado. A aplicação emite solicitações `Scan` em ambas as tabelas para verificar se o conteúdo delas é realmente idêntico.

A classe `DemoHelper` contém um método `ScanTable` que chama a API de `Scan` de baixo nível. O exemplo a seguir mostra como fazer isso.

```
if (StreamsAdapterDemoHelper.scanTable(dynamoDBClient, srcTable).getItems()
    .equals(StreamsAdapterDemoHelper.scanTable(dynamoDBClient, destTable).getItems()))
 {
    System.out.println("Scan result is equal.");
}
else {
    System.out.println("Tables are different!");
}
```

Etapa 5: limpar

A demonstração está concluída e, portanto, a aplicação exclui as tabelas de origem e destino. Consulte o seguinte exemplo de código. Mesmo depois que as tabelas são excluídas, seus fluxos permanecem disponíveis por até 24 horas. Após esse período, eles serão automaticamente excluídos.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

Programa completo: DynamoDB Streams Kinesis Adapter

Veja a seguir o programa Java completo que realiza as tarefas descritas em Demonstração: DynamoDB Streams Kinesis Adapter. Quando executá-lo, você verá uma saída semelhante à seguinte:

```
Creating table KCL-Demo-src
Creating table KCL-Demo-dest
Table is active.
Creating worker for stream: arn:aws:dynamodb:us-west-2:111122223333:table/KCL-Demo-src/
stream/2015-05-19T22:48:56.601
Starting worker...
Scan result is equal.
Done.
```

Important

Para executar esse programa, verifique se a aplicação cliente tem acesso ao DynamoDB e ao Amazon CloudWatch usando políticas. Para obter mais informações, consulte Políticas baseadas em identidade para o DynamoDB.

O código-fonte consiste em quatro arquivos `.java`:

- `StreamsAdapterDemo.java`
- `StreamsRecordProcessor.java`
- `StreamsRecordProcessorFactory.java`
- `StreamsAdapterDemoHelper.java`

StreamsAdapterDemo.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
```

```
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsSchedulerFactory;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.polling.DynamoDBStreamsPollingConfig;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.processor.StreamTracker;
import software.amazon.kinesis.retrieval.RetrievalConfig;

public class StreamsAdapterDemo {

    private static DynamoDbAsyncClient dynamoDbAsyncClient;
    private static CloudWatchAsyncClient cloudWatchAsyncClient;
    private static AmazonDynamoDBStreamsAdapterClient
 amazonDynamoDbStreamsAdapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Region region = Region.US_EAST_1;
    private static AwsCredentialsProvider credentialsProvider =
 DefaultCredentialsProvider.create();

    public static void main( String[] args ) throws Exception {
        System.out.println("Starting demo...");
        dynamoDbAsyncClient = DynamoDbAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        amazonDynamoDbStreamsAdapterClient = new
 AmazonDynamoDBStreamsAdapterClient(credentialsProvider, region);
```

```
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

        setUpTables();

        StreamTracker streamTracker =
StreamsSchedulerFactory.createSingleStreamTracker(streamArn,

InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON));

        ShardRecordProcessorFactory shardRecordProcessorFactory =
                new StreamsAdapterDemoProcessorFactory(dynamoDbAsyncClient, destTable);

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                "streams-adapter-demo",
                amazonDynamoDbStreamsAdapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                "streams-demo-worker",
                shardRecordProcessorFactory
        );

        DynamoDBStreamsPollingConfig pollingConfig = new
DynamoDBStreamsPollingConfig(amazonDynamoDbStreamsAdapterClient);
        RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
        retrievalConfig.retrievalSpecificConfig(pollingConfig);

        System.out.println("Creating scheduler for stream " + streamArn);
        Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                amazonDynamoDbStreamsAdapterClient
        );

        System.out.println("Starting scheduler...");
        Thread t = new Thread(scheduler);
        t.start();
```

```
        Thread.sleep(250000);

        System.out.println("Stopping scheduler...");
        scheduler.shutdown();
        t.join();

        if (StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient, srcTable).items()
                .equals(StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient,
destTable).items())) {
            System.out.println("Scan result is equal.");
        } else {
            System.out.println("Tables are different!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient,
srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
    }

    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResponse result =
StreamsAdapterDemoHelper.describeTable(dynamoDbAsyncClient, tableName);
            created = result.table().tableStatusAsString().equals("ACTIVE");
            if (created) {
                System.out.println("Table is active.");
                return;
            } else {
                retries++;
                try {
```

```
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // do nothing
                }
            }
        }
        System.out.println("Timeout after table creation. Exiting...");
        cleanupAndExit(1);
    }

    private static void performOps(String tableName) {
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "101",
 "test1");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "101",
 "test2");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "101");
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "102",
 "demo3");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "102",
 "demo4");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "102");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(srcTable).build());

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(destTable).build());
        System.exit(returnValue);
    }
}
```

StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
```

```
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    private Integer checkpointCounter;

    private final DynamoDbAsyncClient dynamoDbAsyncClient;
    private final String tableName;

    public StreamsRecordProcessor(DynamoDbAsyncClient dynamoDbAsyncClient, String
 tableName) {
        this.dynamoDbAsyncClient = dynamoDbAsyncClient;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        this.checkpointCounter = 0;
    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput
 dynamoDBStreamsProcessRecordsInput) {
        for (DynamoDBStreamsClientRecord record:
 dynamoDBStreamsProcessRecordsInput.records()) {
            String data = new String(record.data().array(), StandardCharsets.UTF_8);
            System.out.println(data);
            Record streamRecord = record.getRecord();

            switch (streamRecord.eventName()) {
                case INSERT:
                case MODIFY:
```

```
                    StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().newImage());
                case REMOVE:
                    StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().keys().get("Id").n());
            }
            checkpointCounter += 1;
            if (checkpointCounter % 10 == 0) {
                try {
                    dynamoDBStreamsProcessRecordsInput.checkpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        System.out.println("Lease Lost");
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsAdapterDemoProcessorFactory implements ShardRecordProcessorFactory
 {
    private final String tableName;
    private final DynamoDbAsyncClient dynamoDbAsyncClient;

    public StreamsAdapterDemoProcessorFactory(DynamoDbAsyncClient asyncClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDbAsyncClient = asyncClient;
    }

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor(dynamoDbAsyncClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.ResourceInUseException;
```

```
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.StreamViewType;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(DynamoDbAsyncClient client, String tableName) {
        List<AttributeDefinition> attributeDefinitions = new ArrayList<>();
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("Id")
                .attributeType("N")
                .build());

        List<KeySchemaElement> keySchema = new ArrayList<>();
        keySchema.add(KeySchemaElement.builder()
                .attributeName("Id")
                .keyType(KeyType.HASH) // Partition key
                .build());

        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType(StreamViewType.NEW_IMAGE)
                .build();

        CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(attributeDefinitions)
                .keySchema(keySchema)
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .streamSpecification(streamSpecification)
                .build();

        try {
            System.out.println("Creating table " + tableName);
```

```
            CreateTableResponse result = client.createTable(createTableRequest).join();
            return result.tableDescription().latestStreamArn();
        } catch (Exception e) {
            if (e.getCause() instanceof ResourceInUseException) {
                System.out.println("Table already exists.");
                return describeTable(client, tableName).table().latestStreamArn();
            }
            throw e;
        }
    }

    public static DescribeTableResponse describeTable(DynamoDbAsyncClient client,
String tableName) {
        return client.describeTable(DescribeTableRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static ScanResponse scanTable(DynamoDbAsyncClient dynamoDbClient, String
tableName) {
        return dynamoDbClient.scan(ScanRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put("Id", AttributeValue.builder().n(id).build());
        item.put("attribute-1", AttributeValue.builder().s(val).build());

        putItem(dynamoDbClient, tableName, item);
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
                               Map<String, AttributeValue> items) {
        PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(items)
                .build();
        dynamoDbClient.putItem(putItemRequest).join();
    }
```

```
    public static void updateItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id, String val) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attr2", "attribute-2");

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":val", AttributeValue.builder().s(val).build());

        UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #attr2 = :val")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

        dynamoDbClient.updateItem(updateItemRequest).join();
    }

    public static void deleteItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        DeleteItemRequest deleteItemRequest = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .build();
        dynamoDbClient.deleteItem(deleteItemRequest).join();
    }
}
```

API de baixo nível do DynamoDB Streams: exemplo em Java

> **Note**
>
> O código nesta página não é completo e não trata todos os cenários de consumo do Amazon DynamoDB Streams. A maneira recomendada de consumir registros de stream do DynamoDB é por meio do Amazon Kinesis Adapter usando a Kinesis Client Library

(KCL), conforme descrito em Usar o adaptador do DynamoDB Streams Kinesis Adapter para processar registros de fluxos.

Esta seção contém um programa em Java que mostra o DynamoDB Streams em ação. O programa faz o seguinte:

1. Cria uma tabela do DynamoDB com um fluxo habilitado.
2. Descreve as configurações de fluxo dessa tabela.
3. Modifica os dados na tabela.
4. Descreve os fragmentos no fluxo.
5. Lê os registros de fluxo dos fragmentos.
6. Busca fragmentos filhos e continua lendo os registros.
7. Limpa.

Quando executar o programa, você verá um resultado semelhante ao seguinte:

```
Testing Streams Demo
Creating an Amazon DynamoDB table TestTableForStreams with a simple primary key: Id
Waiting for TestTableForStreams to be created...
Current stream ARN for TestTableForStreams: arn:aws:dynamodb:us-
east-2:123456789012:table/TestTableForStreams/stream/2018-03-20T16:49:55.208
Stream enabled: true
Update view type: NEW_AND_OLD_IMAGES

Performing write activities on TestTableForStreams
Processing item 1 of 100
Processing item 2 of 100
Processing item 3 of 100
...
Processing item 100 of 100
Shard: {ShardId: shardId-1234567890-...,SequenceNumberRange: {StartingSequenceNumber:
 100002572486797508907,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=200000158404754583390 9, SizeBytes=22,
 StreamViewType=NEW_AND_OLD_IMAGES)
```

```
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=210000360486976789270l, SizeBytes=55,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, SequenceNumber=2200001099771112898434, SizeBytes=36,
 StreamViewType=NEW_AND_OLD_IMAGES)
...
Deleting the table...
Table StreamsDemoTable deleted.
Demo complete
```

Example Exemplo

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsRequest;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsResponse;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
```

```
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.Shard;
import software.amazon.awssdk.services.dynamodb.model.ShardFilter;
import software.amazon.awssdk.services.dynamodb.model.ShardFilterType;
import software.amazon.awssdk.services.dynamodb.model.ShardIteratorType;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.streams.DynamoDbStreamsClient;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

public class StreamsLowLevelDemo {


    public static void main(String[] args) {
        final String usage = "Testing Streams Demo";
        try {
            System.out.println(usage);

            String tableName = "StreamsDemoTable";
            String key = "Id";
            System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
            Region region = Region.US_WEST_2;
            DynamoDbClient ddb = DynamoDbClient.builder()
                    .region(region)
                    .build();

            DynamoDbStreamsClient ddbStreams = DynamoDbStreamsClient.builder()
                    .region(region)
                    .build();
            DescribeTableRequest describeTableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();
            TableDescription tableDescription = null;
            try{
                tableDescription = ddb.describeTable(describeTableRequest).table();
            }catch (Exception e){
                System.out.println("Table " + tableName + " does not exist.");
                tableDescription = createTable(ddb, tableName, key);
            }
```

```
            // Print the stream settings for the table
            String streamArn = tableDescription.latestStreamArn();

            StreamSpecification streamSpec = tableDescription.streamSpecification();
            System.out.println("Current stream ARN for " + tableDescription.tableName()
+ ": " +
                   streamArn);
            System.out.println("Stream enabled: " + streamSpec.streamEnabled());
            System.out.println("Update view type: " + streamSpec.streamViewType());
            System.out.println();
            // Generate write activity in the table
            System.out.println("Performing write activities on " + tableName);
            int maxItemCount = 100;
            for (Integer i = 1; i <= maxItemCount; i++) {
                System.out.println("Processing item " + i + " of " + maxItemCount);
                // Write a new item
                putItemInTable(key, i, tableName, ddb);
                // Update the item
                updateItemInTable(key, i, tableName, ddb);
                // Delete the item
                deleteDynamoDBItem(key, i, tableName, ddb);
            }

            // Process Stream
            processStream(streamArn, maxItemCount, ddb, ddbStreams, tableName);

            // Delete the table
            System.out.println("Deleting the table...");
            DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build();
            ddb.deleteTable(deleteTableRequest);
            System.out.println("Table " + tableName + " deleted.");
            System.out.println("Demo complete");
            ddb.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void processStream(String streamArn, int maxItemCount,
DynamoDbClient ddb, DynamoDbStreamsClient ddbStreams, String tableName) {
        // Get all the shard IDs from the stream. Note that DescribeStream returns
        // the shard IDs one page at a time.
```

```
        String lastEvaluatedShardId = null;
        do {
            DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                    .streamArn(streamArn)
                    .exclusiveStartShardId(lastEvaluatedShardId).build();
            DescribeStreamResponse describeStreamResponse =
ddbStreams.describeStream(describeStreamRequest);

            List<Shard> shards = describeStreamResponse.streamDescription().shards();

            // Process each shard on this page

            fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams, shards);

            // If LastEvaluatedShardId is set, then there is
            // at least one more page of shard IDs to retrieve
            lastEvaluatedShardId =
describeStreamResponse.streamDescription().lastEvaluatedShardId();

        } while (lastEvaluatedShardId != null);

    }

    private static void fetchShardsAndReadRecords(String streamArn, int maxItemCount,
DynamoDbStreamsClient ddbStreams, List<Shard> shards) {
        for (Shard shard : shards) {
            String shardId = shard.shardId();
            System.out.println("Shard: " + shard);

            // Get an iterator for the current shard
            GetShardIteratorRequest shardIteratorRequest =
GetShardIteratorRequest.builder()
                    .streamArn(streamArn).shardId(shardId)
                    .shardIteratorType(ShardIteratorType.TRIM_HORIZON).build();

            GetShardIteratorResponse getShardIteratorResult =
ddbStreams.getShardIterator(shardIteratorRequest);

            String currentShardIter = getShardIteratorResult.shardIterator();

            // Shard iterator is not null until the Shard is sealed (marked as
READ_ONLY).
```

```
            // To prevent running the loop until the Shard is sealed, we process only
the
            // items that were written into DynamoDB and then exit.
            int processedRecordCount = 0;
            while (currentShardIter != null && processedRecordCount < maxItemCount) {
                // Use the shard iterator to read the stream records
                GetRecordsRequest getRecordsRequest = GetRecordsRequest.builder()
                        .shardIterator(currentShardIter).build();
                GetRecordsResponse getRecordsResult =
ddbStreams.getRecords(getRecordsRequest);
                List<Record> records = getRecordsResult.records();
                for (Record record : records) {
                    System.out.println("        " + record.dynamodb());
                }
                processedRecordCount += records.size();
                currentShardIter = getRecordsResult.nextShardIterator();
            }
            if (currentShardIter == null){
                System.out.println("Shard has been fully processed. Shard iterator is
null.");
                System.out.println("Fetch the child shard to continue processing
instead of bulk fetching all shards");
                DescribeStreamRequest describeStreamRequestForChildShards =
DescribeStreamRequest.builder()
                        .streamArn(streamArn)
                        .shardFilter(ShardFilter.builder()
                                .type(ShardFilterType.CHILD_SHARDS)
                                .shardId(shardId).build())
                        .build();
                DescribeStreamResponse describeStreamResponseChildShards =
ddbStreams.describeStream(describeStreamRequestForChildShards);
                fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams,
describeStreamResponseChildShards.streamDescription().shards());
            }
        }
    }

    private static void putItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());
        item.put("Message", AttributeValue.builder()
```

```
                .s("New Item!")
                .build());
        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(item)
                .build();
        ddb.putItem(request);
    }

    private static void updateItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());


        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("Message", AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s("This is an updated item").build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();
        ddb.updateItem(request);
    }

    public static void deleteDynamoDBItem(String key, Integer i, String tableName,
DynamoDbClient ddb) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();
        ddb.deleteItem(deleteReq);
```

```
    }

    public static TableDescription createTable(DynamoDbClient ddb, String tableName,
String key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType("NEW_AND_OLD_IMAGES")
                .build();
        CreateTableRequest request = CreateTableRequest.builder()
                .attributeDefinitions(AttributeDefinition.builder()
                        .attributeName(key)
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(KeySchemaElement.builder()
                        .attributeName(key)
                        .keyType(KeyType.HASH)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
                .tableName(tableName)
                .streamSpecification(streamSpecification)
                .build();

        TableDescription newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            System.out.println("Waiting for " + tableName + " to be created...");

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
```

```
        return null;
    }



}
```

DynamoDB Streams e acionadores do AWS Lambda

O Amazon DynamoDB é integrado ao AWS Lambda para que você possa criar acionadores (trechos de código que respondem automaticamente a eventos no DynamoDB Streams). Com os acionadores, você pode criar aplicações que reagem às modificações de dados em tabelas do DynamoDB.

Tópicos

- Tutorial 1: Usar filtros para processar todos os eventos com o Amazon DynamoDB e o AWS Lambda usando a AWS CLI
- Tutorial 2: Usar filtros para processar alguns eventos com o DynamoDB e o Lambda
- Práticas recomendadas de uso do DynamoDB Streams com o Lambda

Se você habilitar o DynamoDB Streams em uma tabela, poderá associar o nome do recurso da Amazon (ARN) do fluxo a uma função do AWS Lambda escrita por você. Todas as ações de mutação nessa tabela do DynamoDB poderão então ser capturadas como um item no fluxo. Por exemplo, é possível definir um gatilho para que, quando um item em uma tabela for modificado, um novo registro apareça imediatamente no fluxo dessa tabela.

> **Note**
>
> Se você inscrever mais de duas funções do Lambda em um fluxo do DynamoDB, poderá ocorrer controle de utilização de leitura.

O serviço AWS Lambda pesquisa o fluxo em busca de novos registros quatro vezes por segundo. Quando novos registros de fluxo estão disponíveis, a função do Lambda é invocada de maneira síncrona. É possível inscrever até duas funções do Lambda no mesmo fluxo do DynamoDB. Se você inscrever mais de duas funções do Lambda no mesmo fluxo do DynamoDB, poderá ocorrer controle de utilização de leitura.

A função do Lambda pode enviar uma notificação, iniciar uma workflow ou realizar qualquer outra ação especificada. É possível escrever uma função do Lambda para simplificar a cópia de cada registro de fluxo no armazenamento persistente, como o Gateway de Arquivos do Amazon S3 (Amazon S3), e criar uma trilha de auditoria permanente de atividades de gravação na tabela. Ou suponhamos que você tenha um aplicativo de jogos móveis que grava em uma tabela `GameScores`. Sempre que o atributo `TopScore` da tabela `GameScores` é atualizado, um registro de fluxo correspondente é gravado no fluxo da tabela. Este evento poderia, em seguida, acionar uma função do Lambda que posta uma mensagem de felicitações em uma rede de mídia social. Essa função também seria escrita para ignorar quaisquer registros de fluxo que não são atualizações para `GameScores` ou que não modificam o atributo `TopScore`.

Se a sua função retornar um erro, o Lambda tentará executar novamente o lote até que o processamento seja bem-sucedido ou os dados expirem. Você também pode configurar o Lambda para tentar novamente com um lote menor, limitar o número de tentativas, descartar registros quando eles se tornarem muito antigos e outras opções.

Como práticas recomendadas de performance, a função do Lambda precisa ser de curta duração. Para evitar a introdução de atrasos de processamento desnecessários, ela também não deve executar uma lógica complexa. Para um fluxo de alta velocidade em particular, é melhor acionar fluxos de trabalho assíncronos de função de etapa de pós-processamento do que Lambdas síncronos de longa execução.

Não é possível usar o mesmo acionador do Lambda em diferentes contas da AWS. A tabela do DynamoDB e as funções do Lambda devem pertencer à mesma conta da AWS.

Para obter mais informações sobre o AWS Lambda, consulte o Guia do desenvolvedor do AWS Lambda.

Tutorial 1: Usar filtros para processar todos os eventos com o Amazon DynamoDB e o AWS Lambda usando a AWS CLI

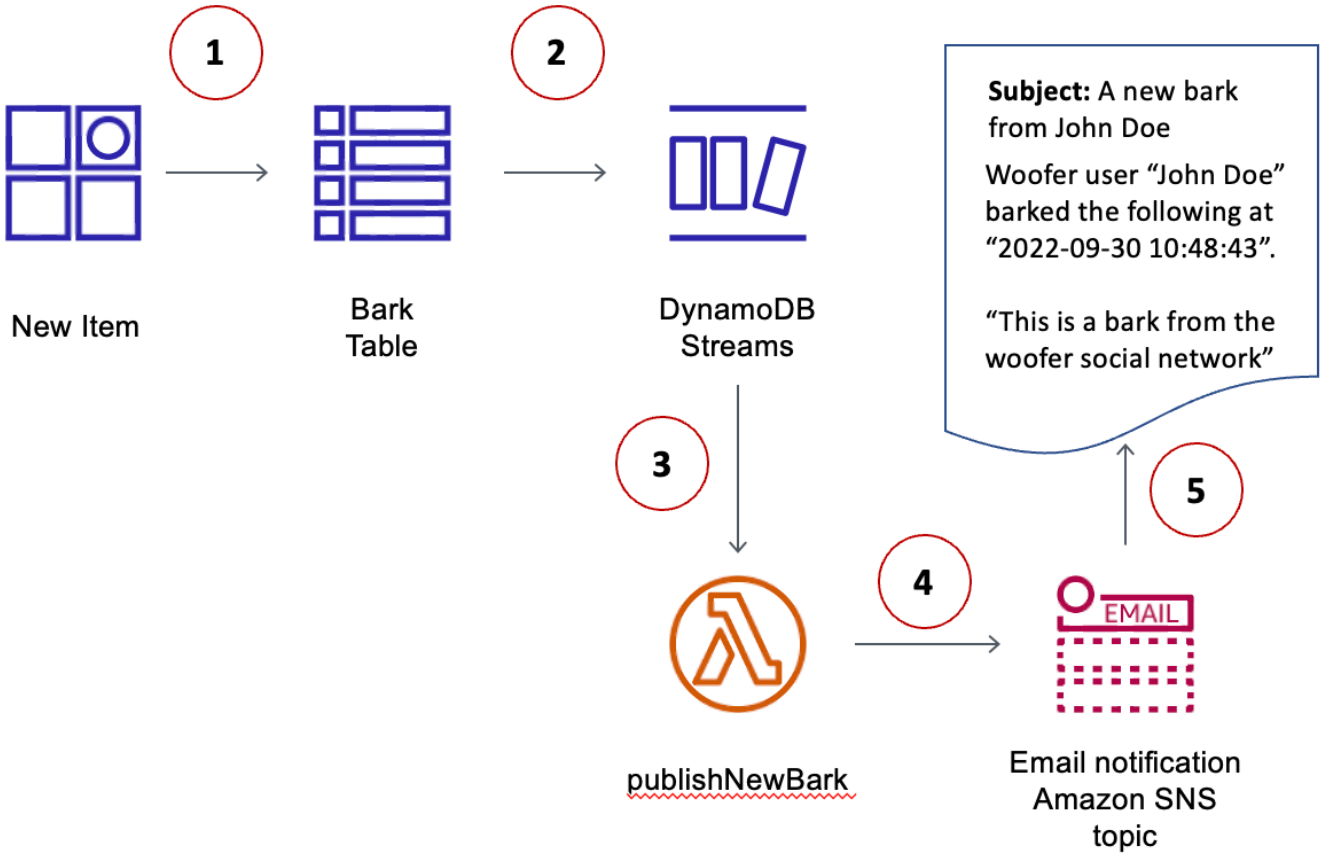
Neste tutorial, você cria um acionador do AWS Lambda para processar um fluxo de uma tabela do DynamoDB.

Tópicos

- Etapa 1: criar uma tabela do DynamoDB com um fluxo habilitado
- Etapa 2: criar uma função de execução do Lambda
- Etapa 3: criar um tópico do Amazon SNS

- Etapa 4: criar e testar uma função do Lambda
- Etapa 5: criar e testar um acionador

O cenário deste tutorial é o Woofer, uma rede social simples. Os usuários do Woofer se comunicam usando barks (mensagens de texto curtas) que são enviados a outros usuários do Woofer. O diagrama a seguir mostra os componentes e o fluxo de trabalho desse aplicativo.



1. Um usuário grava um item em uma tabela do DynamoDB (`BarkTable`). Cada item na tabela representa um bark.
2. Um novo registro de fluxo é gravado para refletir que um novo item foi adicionado à `BarkTable`.
3. O novo registro de fluxo aciona uma função do AWS Lambda (`publishNewBark`).
4. Se o registro de fluxo indicar que um novo item foi adicionado à `BarkTable`, a função do Lambda lerá os dados do registro de fluxo e publicará uma mensagem em um tópico no Amazon Simple Notification Service (Amazon SNS).
5. A mensagem é recebida pelos assinantes do tópico do Amazon SNS. (Neste tutorial, o único assinante é um endereço de e-mail.)

Antes de começar

Este tutorial usa a AWS Command Line Interface AWS CLI. Se você ainda não tiver feito isso, siga as instruções de instalação no Guia do usuário do AWS Command Line Interface para instalar e configurar a AWS CLI.

Etapa 1: criar uma tabela do DynamoDB com um fluxo habilitado

Nesta etapa, você cria uma tabela do DynamoDB (`BarkTable`) para armazenar todos os barks dos usuários do Woofer. A chave primária é composta de `Username` (chave de partição) e de `Timestamp` (chave de classificação). Ambos os atributos são do tipo string.

`BarkTable` tem um fluxo habilitado. Mais adiante neste tutorial, você criará um acionador associando uma função do AWS Lambda ao fluxo.

1. Use o seguinte comando para criar a tabela.

```
aws dynamodb create-table \
    --table-name BarkTable \
    --attribute-definitions AttributeName=Username,AttributeType=S
 AttributeName=Timestamp,AttributeType=S \
    --key-schema AttributeName=Username,KeyType=HASH
 AttributeName=Timestamp,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

2. Na saída, procure o `LatestStreamArn`.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

   Anote a *`region`* e o *`accountID`*, pois eles serão necessários para as outras etapas deste tutorial.

Etapa 2: criar uma função de execução do Lambda

Nesta etapa, você cria uma função do AWS Identity and Access Management (IAM) (`WooferLambdaRole`) e atribui permissões a ela. Essa função será usada pela função do Lambda que você cria em Etapa 4: criar e testar uma função do Lambda.

Você também cria uma política para a função. A política contém todas as permissões de que a função do Lambda precisa em tempo de execução.

1. Crie um arquivo denominado `trust-relationship.json` com o seguinte conteúdo:

   JSON

   ```
   {
      "Version": "2012-10-17"       ,
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
   ```

2. Insira o seguinte comando para criar a `WooferLambdaRole`.

   ```
   aws iam create-role --role-name WooferLambdaRole \
       --path "/service-role/" \
       --assume-role-policy-document file://trust-relationship.json
   ```

3. Crie um arquivo denominado `role-policy.json` com o seguinte conteúdo: (Substitua *`region`* e *`accountID`* por sua região e seu ID de conta da AWS.)

   JSON

   ```
   {
       "Version": "2012-10-17"       ,
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "logs:CreateLogGroup",
                   "logs:CreateLogStream",
                   "logs:PutLogEvents"
               ],
   ```

```
            "Resource": "arn:aws:logs:us-east-1:111122223333:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/
BarkTable/stream/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

A política tem quatro declarações que fornecem permissões à `WooferLambdaRole` para fazer o seguinte:

- Execute uma função do Lambda (`publishNewBark`). Você cria a função mais adiante neste tutorial.
- Acesse o Amazon CloudWatch Logs. A função do Lambda grava o diagnóstico no CloudWatch Logs em tempo de execução.
- Leia os dados do fluxo do DynamoDB para `BarkTable`.
- Publique mensagens no Amazon SNS.

4. Execute o seguinte comando para anexar a política à função `WooferLambdaRole`.

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

Etapa 3: criar um tópico do Amazon SNS

Nesta etapa, você cria um tópico do Amazon SNS (`wooferTopic`) e inscreve um endereço de e-mail nele. A função do Lambda usa esse tópico para publicar novos barks de usuários do Woofer.

1. Digite o seguinte comando para criar um novo tópico do Amazon SNS.

```
aws sns create-topic --name wooferTopic
```

2. Digite o seguinte comando para inscrever um endereço de e-mail no `wooferTopic`. (Substitua *`region`* e *`accountID`* por sua região e ID da conta da AWS e substitua *`example@example.com`* por um endereço de e-mail válido.)

```
aws sns subscribe \
    --topic-arn arn:aws:sns:region:accountID:wooferTopic \
    --protocol email \
    --notification-endpoint example@example.com
```

3. O Amazon SNS envia uma mensagem de confirmação ao seu endereço de e-mail. Selecione o link Confirm subscription (Confirmar assinatura) na mensagem para concluir o processo de assinatura.

Etapa 4: criar e testar uma função do Lambda

Nesta etapa, você cria uma função do AWS Lambda (`publishNewBark`) para processar registros de fluxo da `BarkTable`.

A função `publishNewBark` processa apenas os eventos de fluxo que correspondem a novos itens na `BarkTable`. A função lê dados de um evento como esse e, em seguida, invoca o Amazon SNS; para publicá-lo.

1. Crie um arquivo denominado `publishNewBark.js` com o seguinte conteúdo: (Substitua *`region`* e *`accountID`* por sua região e seu ID de conta da AWS.)

```
'use strict';
var AWS = require("aws-sdk");
var sns = new AWS.SNS();

exports.handler = (event, context, callback) => {

    event.Records.forEach((record) => {
```

```
        console.log('Stream record: ', JSON.stringify(record, null, 2));

        if (record.eventName == 'INSERT') {
            var who = JSON.stringify(record.dynamodb.NewImage.Username.S);
            var when = JSON.stringify(record.dynamodb.NewImage.Timestamp.S);
            var what = JSON.stringify(record.dynamodb.NewImage.Message.S);
            var params = {
                Subject: 'A new bark from ' + who,
                Message: 'Woofer user ' + who + ' barked the following at ' + when
 + ':\n\n ' + what,
                TopicArn: 'arn:aws:sns:region:accountID:wooferTopic'
            };
            sns.publish(params, function(err, data) {
                if (err) {
                    console.error("Unable to send message. Error JSON:",
 JSON.stringify(err, null, 2));
                } else {
                    console.log("Results from sending message: ",
 JSON.stringify(data, null, 2));
                }
            });
        }
    });
    callback(null, `Successfully processed ${event.Records.length} records.`);
};
```

2. Crie um arquivo zip para conter `publishNewBark.js`. Se você tiver o utilitário de linha de comando zip, poderá digitar o seguinte comando para fazer isso.

```
zip publishNewBark.zip publishNewBark.js
```

3. Ao criar a função do Lambda, você especifica o nome do recurso da Amazon (ARN) da `WooferLambdaRole` que você criou em Etapa 2: criar uma função de execução do Lambda. Digite o seguinte comando para recuperar o ARN.

```
aws iam get-role --role-name WooferLambdaRole
```

   Na saída, procure o ARN da `WooferLambdaRole`.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

Use o seguinte comando para criar a função do Lambda. Substitua *roleARN* pelo ARN da `WooferLambdaRole`.

```
aws lambda create-function \
    --region region \
    --function-name publishNewBark \
    --zip-file fileb://publishNewBark.zip \
    --role roleARN \
    --handler publishNewBark.handler \
    --timeout 5 \
    --runtime nodejs16.x
```

4. Agora teste o `publishNewBark` para verificar se ele funciona. Para fazer isso, você deve fornecer informações semelhantes a um registro real do DynamoDB Streams.

   Crie um arquivo denominado `payload.json` com o seguinte conteúdo: Substitua *region* e *accountID* por sua Região da AWS e seu ID de conta.

```
{
    "Records": [
        {
            "eventID": "7de3041dd709b024af6f29e4fa13d34c",
            "eventName": "INSERT",
            "eventVersion": "1.1",
            "eventSource": "aws:dynamodb",
            "awsRegion": "region",
            "dynamodb": {
                "ApproximateCreationDateTime": 1479499740,
                "Keys": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "NewImage": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Message": {
```

```
                        "S": "This is a bark from the Woofer social network"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "SequenceNumber": "130216000000000001596893679",
                "SizeBytes": 112,
                "StreamViewType": "NEW_IMAGE"
            },
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

Use o seguinte comando para testar a função `publishNewBark`.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

Se o teste for bem-sucedido, você verá a seguinte saída.

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Além disso, o arquivo `output.txt` conterá o seguinte texto.

```
"Successfully processed 1 records."
```

Você também receberá uma nova mensagem de e-mail dentro de alguns minutos.

> **Note**
>
> OAWS Lambda grava informações de diagnóstico no Amazon CloudWatch Logs. Se você encontrar erros em sua função do Lambda, poderá usar essas informações de diagnóstico para fins de solução de problemas:

1. Abra o console do CloudWatch em https://console.aws.amazon.com/cloudwatch/.
2. No painel de navegação, selecione Logs.
3. Escolha o grupo de logs a seguir: `/aws/lambda/publishNewBark`
4. Escolha o fluxo de logs mais recente para visualizar a saída (e os erros) da função.

Etapa 5: criar e testar um acionador

Em Etapa 4: criar e testar uma função do Lambda, você testou a função do Lambda para garantir que ela fosse executada corretamente. Nesta etapa, você cria um acionador associando a função do Lambda (`publishNewBark`) à origem de um evento (o fluxo `BarkTable`).

1. Ao criar o acionador, você deve especificar o ARN do fluxo de `BarkTable`. Digite o seguinte comando para recuperar o ARN.

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   Na saída, procure o `LatestStreamArn`.

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. Insira o seguinte comando para criar o acionador. Substitua *`streamARN`* pelo ARN do fluxo atual.

   ```
   aws lambda create-event-source-mapping \
       --region region \
       --function-name publishNewBark \
       --event-source streamARN  \
       --batch-size 1 \
       --starting-position TRIM_HORIZON
   ```

3. Teste o acionador. Insira o seguinte comando para adicionar um item a `BarkTable`.

   ```
   aws dynamodb put-item \
       --table-name BarkTable \
   ```

```
    --item Username={S="Jane
Doe"},Timestamp={S="2016-11-18:14:32:17"},Message={S="Testing...1...2...3"}
```

Você deve receber uma nova mensagem de e-mail dentro de alguns minutos.

4. Abra o console do DynamoDB e adicione mais alguns itens a `BarkTable`. Você deve especificar valores para os atributos `Username` e `Timestamp`. (Você também deve especificar um valor para `Message`, embora isso não seja obrigatório.) Você deve receber uma nova mensagem de e-mail para cada item que adicionar a `BarkTable`.

   A função do Lambda processa apenas novos itens que você adiciona a `BarkTable`. Se você atualizar ou excluir um item na tabela, a função não fará nada.

> **Note**
>
> O AWS Lambda grava informações de diagnóstico no Amazon CloudWatch Logs. Se você encontrar erros em sua função do Lambda, poderá usar essas informações de diagnóstico para fins de solução de problemas.
>
> 1. Abra o console do CloudWatch em https://console.aws.amazon.com/cloudwatch/.
> 2. No painel de navegação, selecione Logs.
> 3. Escolha o grupo de logs a seguir: `/aws/lambda/publishNewBark`
> 4. Escolha o fluxo de logs mais recente para visualizar a saída (e os erros) da função.

Tutorial 2: Usar filtros para processar alguns eventos com o DynamoDB e o Lambda

Neste tutorial, você criará um acionador do AWS Lambda para processar somente alguns eventos em um fluxo de uma tabela do DynamoDB.

Tópicos

- Reunir todos os componentes: AWS CloudFormation
- Reunir todos os componentes: CDK

Com a filtragem de eventos do Lambda, é possível utilizar expressões de filtro para controlar quais eventos o Lambda enviará para a função processar. É possível configurar até cinco filtros diferentes

por fluxo do DynamoDB. Se você estiver usando janelas em lotes, o Lambda aplicará os critérios de filtro a cada novo evento para determinar se deseja adicioná-lo ao lote atual.

Os filtros são aplicados por meio de estruturas chamadas `FilterCriteria`. Os três principais atributos de `FilterCriteria` são `metadata properties`, `data properties` e `filter patterns`.

Aqui está um exemplo de estrutura de um evento do DynamoDB Streams:

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Essas `metadata properties` são os campos do objeto do evento. No caso dos DynamoDB Streams, as `metadata properties` são campos como o `dynamodb` ou o `eventName`.

Essas `data properties` são os campos do corpo do evento. Para filtrar as `data properties`, certifique-se de contê-las em `FilterCriteria` dentro da chave adequada. Para fontes de eventos do DynamoDB, a chave de dados é `NewImage` ou `OldImage`.

Por fim, as regras de filtro definirão a expressão de filtro que você deseja aplicar a uma propriedade específica. Veja alguns exemplos:

Operador de comparação	Exemplo	Sintaxe da regra (parcial)
Nulo	O tipo de produto é nulo	`{ "product_type": { "S": null } }`
Vazio	O nome do produto está vazio	`{ "product_name": { "S": [ ""] } }`
Igual	O estado é igual a Flórida	`{ "state": { "S": ["FL"] } }`
E	O estado do produto é igual à Flórida e a categoria do produto é Chocolate	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
Ou	O estado do produto é Flórida ou Califórnia	`{ "state": { "S": ["FL","CA"] } }`
Não	O estado do produto não é Flórida	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
Existe	O produto caseiro existe	`{"homemade": {"S": [{"exists": true}]}}`
Não existe	O produto Homemade não existe	`{"homemade": {"S": [{"exists": false}]}}`
Começa com	PK começa com COMPANY	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

É possível especificar até cinco padrões de filtragem de eventos em uma função do Lambda. Observe que cada um desses cinco eventos será avaliado como um OR lógico. Então, se você configurar dois filtros chamados `Filter_One` e `Filter_Two`, a função do Lambda executará `Filter_One` OU `Filter_Two`.

> **Note**
>
> Na página de filtragem de eventos do Lambda, há algumas opções para filtrar e comparar valores numéricos. No entanto, no caso de eventos de filtro do DynamoDB, isso não se aplica porque os números no DynamoDB são armazenados como strings. Por exemplo `"quantity": { "N": "50" }`, sabemos que é um número por causa da propriedade `"N"`.

Reunir todos os componentes: AWS CloudFormation

Para mostrar a funcionalidade de filtragem de eventos na prática, aqui está um exemplo de modelo do CloudFormation. Esse modelo gerará uma tabela simples do DynamoDB com uma chave de partição PK e uma chave de classificação SK com o Amazon DynamoDB Streams habilitado. Ele criará uma função do Lambda e uma função simples de execução do Lambda que permitirá gravar logs no Amazon Cloudwatch e ler os eventos do Amazon DynamoDB Stream. Ele também adicionará o mapeamento da origem do evento entre os DynamoDB Streams e a função do Lambda, para que a função possa ser executada sempre que houver um evento no Amazon DynamoDB Streams.

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Sample application that presents AWS Lambda event source filtering
with Amazon DynamoDB Streams.

Resources:
  StreamsSampleDDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: "PK"
          AttributeType: "S"
        - AttributeName: "SK"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "PK"
```

```
        KeyType: "HASH"
      - AttributeName: "SK"
        KeyType: "RANGE"
    StreamSpecification:
      StreamViewType: "NEW_AND_OLD_IMAGES"
    ProvisionedThroughput:
      ReadCapacityUnits: 5
      WriteCapacityUnits: 5

LambdaExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - lambda.amazonaws.com
          Action:
            - sts:AssumeRole
    Path: "/"
    Policies:
      - PolicyName: root
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - logs:CreateLogGroup
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource: arn:aws:logs:*:*:*
            - Effect: Allow
              Action:
                - dynamodb:DescribeStream
                - dynamodb:GetRecords
                - dynamodb:GetShardIterator
                - dynamodb:ListStreams
              Resource: !GetAtt StreamsSampleDDBTable.StreamArn

EventSourceDDBTableStream:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
```

```
      BatchSize: 1
      Enabled: True
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST

  ProcessEventLambda:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      Timeout: 300
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import logging

          LOGGER = logging.getLogger()
          LOGGER.setLevel(logging.INFO)

          def handler(event, context):
            LOGGER.info('Received Event: %s', event)
            for rec in event['Records']:
              LOGGER.info('Record: %s', rec)

Outputs:
  StreamsSampleDDBTable:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.Arn
  StreamARN:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.StreamArn
```

Depois de implantar esse modelo de formação de nuvem, é possível inserir o seguinte item do Amazon DynamoDB:

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
```

```
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Graças à função simples do Lambda incluída em linha nesse modelo de formação de nuvem, você verá os eventos nos grupos de logs do Amazon CloudWatch para a função do Lambda da seguinte forma:

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Exemplos de filtragem

- Somente produtos que correspondam a um determinado estado

Este exemplo modifica o modelo do CloudFormation para incluir um filtro que corresponda a todos os produtos provenientes da Flórida, com a abreviatura “FL”.

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Depois de reimplantar a pilha, é possível adicionar o seguinte item do DynamoDB à tabela. Observe que ele não aparecerá nos logs de funções do Lambda, porque o produto neste exemplo é da Califórnia.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK#1000",
 "company_id": "1000",
 "fabric": "Florida Chocolates",
 "price": 15,
 "product_id": "1000",
 "quantity": 50,
 "state": "CA",
 "stores": 5,
 "type": ""
}
```

- Somente os itens que começam com alguns valores em PK e SK

Este exemplo modifica o modelo do CloudFormation para incluir a seguinte condição:

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
```

```
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
"COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Observe que a condição AND exige que a condição esteja dentro do padrão, onde as chaves PK e SK estão na mesma expressão separadas por vírgula.

Comece com alguns valores em PK e SK ou de determinado estado.

Este exemplo modifica o modelo do CloudFormation para incluir as seguintes condições:

```
 EventSourceDDBTableStream:
   Type: AWS::Lambda::EventSourceMapping
   Properties:
     BatchSize: 1
     Enabled: True
     FilterCriteria:
       Filters:
         - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
"COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
         - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
     EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
     FunctionName: !GetAtt ProcessEventLambda.Arn
     StartingPosition: LATEST
```

Observe que a condição OR é adicionada introduzindo novos padrões na seção de filtro.

Reunir todos os componentes: CDK

O exemplo de modelo de formação de projeto CDK a seguir mostra a funcionalidade de filtragem de eventos. Antes de trabalhar com esse projeto de CDK, será preciso instalar os pré-requisitos, incluindo a execução de scripts de preparação.

Criar um projeto de CDK

Primeiro, crie um novo projeto do AWS CDK, invocando `cdk init` em um diretório vazio.

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

O comando `cdk init` usa o nome da pasta do projeto para nomear vários elementos do projeto, incluindo classes, subpastas e arquivos. Todos os hifens no nome da pasta são convertidos em sublinhados. Caso contrário, o nome deve seguir a forma de um identificador Python. Por exemplo, ele não deve começar com um número nem conter espaços.

Para trabalhar com o novo projeto, ative o respectivo ambiente virtual. Isso permite que as dependências do projeto sejam instaladas localmente na pasta do projeto, em vez de globalmente.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> É possível reconhecer isso como o comando Mac/Linux para ativar um ambiente virtual. Os modelos do Python incluem um arquivo em lote, `source.bat`, que permite que o mesmo comando seja utilizado no Windows. O comando tradicional do Windows `.venv\Scripts\activate.bat` também funciona. Se você inicializou seu projeto do AWS CDK usando o AWS CDK Toolkit v1.70.0 ou anterior, seu ambiente virtual está no diretório `.env` em vez de `.venv`.

Infraestrutura base

Abra o arquivo `./ddb_filters/ddb_filters_stack.py` com o editor de texto de sua preferência. Esse arquivo foi gerado automaticamente quando você criou o projeto do AWS CDK.

Em seguida, adicione as funções `_create_ddb_table` e `_set_ddb_trigger_function`. Essas funções criarão uma tabela do DynamoDB com a chave de partição PK e a chave de classificação SK no modo de provisionamento sob demanda, com o Amazon DynamoDB Streams habilitado por padrão para mostrar imagens novas e antigas.

A função do Lambda será armazenada na pasta `lambda` abaixo do arquivo `app.py`. Esse arquivo será criado posteriormente. Ele incluirá uma variável de ambiente `APP_TABLE_NAME`, que será o nome da tabela do Amazon DynamoDB criada por essa pilha. Na mesma função, concederemos

permissões de leitura de fluxo para a função do Lambda. Por fim, ele se inscreverá no DynamoDB Streams como fonte de eventos para a função do Lambda.

No final do arquivo no método `__init__`, você chamará as respectivas estruturas para inicializá-las na pilha. Para projetos maiores que exigem componentes e serviços adicionais, talvez seja melhor definir essas estruturas fora da pilha base.

```
import os
import json

import aws_cdk as cdk
from aws_cdk import (
    Stack,
    aws_lambda as _lambda,
    aws_dynamodb as dynamodb,
)
from constructs import Construct


class DdbFiltersStack(Stack):

    def _create_ddb_table(self):
        dynamodb_table = dynamodb.Table(
            self,
            "AppTable",
            partition_key=dynamodb.Attribute(
                name="PK", type=dynamodb.AttributeType.STRING
            ),
            sort_key=dynamodb.Attribute(
                name="SK", type=dynamodb.AttributeType.STRING),
            billing_mode=dynamodb.BillingMode.PAY_PER_REQUEST,
            stream=dynamodb.StreamViewType.NEW_AND_OLD_IMAGES,
            removal_policy=cdk.RemovalPolicy.DESTROY,
        )

        cdk.CfnOutput(self, "AppTableName", value=dynamodb_table.table_name)
        return dynamodb_table

    def _set_ddb_trigger_function(self, ddb_table):
        events_lambda = _lambda.Function(
            self,
            "LambdaHandler",
            runtime=_lambda.Runtime.PYTHON_3_9,
```

```
            code=_lambda.Code.from_asset("lambda"),
            handler="app.handler",
            environment={
                "APP_TABLE_NAME": ddb_table.table_name,
            },
        )

        ddb_table.grant_stream_read(events_lambda)

        event_subscription = _lambda.CfnEventSourceMapping(
            scope=self,
            id="companyInsertsOnlyEventSourceMapping",
            function_name=events_lambda.function_name,
            event_source_arn=ddb_table.table_stream_arn,
            maximum_batching_window_in_seconds=1,
            starting_position="LATEST",
            batch_size=1,
        )

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ddb_table = self._create_ddb_table()
        self._set_ddb_trigger_function(ddb_table)
```

Agora, criaremos uma função do Lambda muito simples que imprimirá os logs no Amazon CloudWatch. Para fazer isso, crie uma pasta chamada `lambda`.

```
mkdir lambda
touch app.py
```

Usando o editor de texto de sua preferência, adicione o seguinte conteúdo ao arquivo `app.py`:

```
import logging

LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)


def handler(event, context):
    LOGGER.info('Received Event: %s', event)
    for rec in event['Records']:
```

```
        LOGGER.info('Record: %s', rec)
```

Garantindo que você esteja na pasta `/ddb_filters/`, digite o seguinte comando para criar a aplicação de exemplo:

```
cdk deploy
```

Em algum momento, você deverá confirmar se deseja implantar a solução. Aceite as alterações digitando Y.

```
##########################################################################################
# + # ${LambdaHandler/ServiceRole} # arn:${AWS::Partition}:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole #
##########################################################################################

Do you wish to deploy these changes (y/n)? y

...

#  Deployment time: 67.73s

Outputs:
DdbFiltersStack.AppTableName = DdbFiltersStack-AppTable815C50BC-1M1W7209V5YPP
Stack ARN:
arn:aws:cloudformation:us-east-2:111122223333:stack/
DdbFiltersStack/66873140-40f3-11ed-8e93-0a74f296a8f6
```

Depois que as alterações forem implantadas, abra o console da AWS e adicione um item à tabela.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Os logs do CloudWatch agora devem conter todas as informações dessa entrada.

Exemplos de filtragem

- Somente produtos que correspondam a um determinado estado

Abra o arquivo `ddb_filters/ddb_filters/ddb_filters_stack.py` e modifique-o para incluir o filtro que corresponde a todos os produtos que são iguais a “FL”. Isso pode ser revisado logo abaixo de `event_subscription` na linha 45.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

- Somente os itens que começam com alguns valores em PK e SK

Modifique o script Python para incluir a seguinte condição:

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
```

```
                )
            },
        ]
    },
```

- Comece com alguns valores em PK e SK ou de determinado estado.

Modifique o script Python para incluir as seguintes condições:

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

Observe que a condição OR é adicionada incluindo mais elementos à matriz Filters (Filtros).

Limpeza

Localize a pilha de filtros na base do diretório de trabalho e execute `cdk destroy`. Confirme a exclusão do recurso:

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Práticas recomendadas de uso do DynamoDB Streams com o Lambda

Uma função do AWS Lambda é executada em um contêiner, um ambiente de execução isolado de outras funções. Quando você executa uma função pela primeira vez, o AWS Lambda cria um novo contêiner e começa a executar o código da função.

Uma função do Lambda tem um manipulador que é executado uma vez por invocação. O manipulador contém a lógica de negócios principal da função. Por exemplo, a função do Lambda mostrada em Etapa 4: criar e testar uma função do Lambda tem um identificador que pode processar registros em um fluxo do DynamoDB.

Você também pode fornecer o código de inicialização que é executado apenas uma vez: depois que o contêiner é criado, mas antes que o AWS Lambda execute o manipulador pela primeira vez. A função do Lambda mostrada em Etapa 4: criar e testar uma função do Lambda possui um código de inicialização que importa o SDK for JavaScript in Node.js e cria um cliente para o Amazon SNS. Esses objetos devem ser definidos somente uma vez, fora do manipulador.

Depois da execução da função, o AWS Lambda pode optar por reutilizar o contêiner para invocações subsequentes da função. Neste caso, o manipulador da função pode reutilizar os recursos que você definiu no seu código de inicialização. (Você não pode controlar por quanto tempo o AWS Lambda reterá o contêiner, ou se o contêiner será reutilizado.)

Para acionadores do DynamoDB que usam o AWS Lambda, recomendamos o seguinte:

- Os clientes de serviço da AWS devem ser instanciados no código de inicialização, e não no manipulador. Isso permite que o AWS Lambda reutilize conexões existentes, durante o ciclo de vida do contêiner.
- Em geral, você não precisa gerenciar explicitamente as conexões ou implementar o pool de conexões porque o AWS Lambda gerencia isso para você.

Um consumidor do Lambda para um fluxo do DynamoDB não garante entrega exatamente uma vez, podendo resultar em duplicações ocasionais. Verifique se o código da função do Lambda é idempotente para evitar que problemas inesperados ocorram devido ao processamento de duplicações.

Para obter mais informações, consulte Práticas recomendadas para trabalhar com funções do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

DynamoDB Streams e Apache Flink

É possível consumir registros do Amazon DynamoDB Streams com o Apache Flink. Com o Amazon Managed Service for Apache Flink, é possível transformar e analisar dados de streaming em tempo real usando o Apache Flink. O Apache Flink é um framework de processamento de fluxos de código aberto para processar dados em tempo real. O conector do Amazon DynamoDB Streams para Apache Flink simplifica a criação e o gerenciamento de workloads do Apache Flink e permite que você integre aplicações com outros Serviços da AWS.

O Amazon Managed Service for Apache Flink ajuda você a criar rapidamente aplicações de processamento de fluxos de ponta a ponta para analytics de logs, analytics de clickstream, Internet das Coisas (IoT), tecnologia de anúncios, jogos e muito mais. Os quatro casos de uso mais comuns são extração, transformação e carregamento (ETL) de streaming, aplicações orientadas a eventos, analytics responsivas em tempo real e consultas interativas de fluxos de dados. Para ter mais informações sobre como gravar do Amazon DynamoDB Streams no Apache Flink, consulte Amazon DynamoDB Streams Connector.

Usar o parâmetro ShardFilter com o conector DynamoDB Streams para Apache Flink

O Amazon DynamoDB Streams dá suporte ao parâmetro `ShardFilter` na API `DescribeStream` para encontrar um subconjunto de fragmentos. Ao especificar um fragmento pai na solicitação, o DynamoDB Streams retornará seus fragmentos filhos imediatos. Você pode usar o parâmetro `ShardFilter` para descobrir rapidamente fragmentos filhos após o fechamento de um fragmento pai, tornando seus aplicativos de processamento de stream mais responsivos e econômicos.

O Amazon Managed Service for Apache Flink dá suporte ao parâmetro `ShardFilter` ao ler do DynamoDB Streams. Para usar esse recurso, você deve solicitar acesso a uma versão especial do conector Flink que inclui suporte a `ShardFilter`. Para começar, abra um caso de suporte no Centro de Suporte da AWS e solicite acesso a esses binários. Para obter mais informações sobre como usar o DynamoDB Streams com Apache Flink, consulte Ler e processar um fluxo no Guia do desenvolvedor do DynamoDB.

Aceleração em memória com o DynamoDB Accelerator (DAX)

O Amazon DynamoDB foi concebido para escala e performance. Na maioria dos casos, os tempos de resposta do DynamoDB podem ser medidos em milissegundos de um dígito. No entanto, existem certos casos de uso que exigem tempos de resposta em microssegundos. Para esses casos de uso, o DynamoDB Accelerator (DAX) oferece tempos de resposta rápidos para acessar dados finais consistentes.

O DAX é um serviço de armazenamento em cache compatível com o DynamoDB no qual você pode se beneficiar da rápida performance em memória para aplicações exigentes. O DAX lida com três cenários principais:

1. Como um cache na memória, o DAX reduz os tempos de resposta de workloads de leitura final consistente por uma ordem de magnitude que varia de milissegundos de um único dígito até microssegundos.
2. O DAX reduz a complexidade operacional e da aplicação fornecendo um serviço gerenciado que é compatível com a API do DynamoDB. Portanto, ele exige apenas alterações funcionais mínimas para uso com um aplicativo existente.
3. Para workloads de leitura intermitentes ou pesadas, o DAX fornece throughput mais alto e economia potencial de custos operacionais reduzindo a necessidade de provisionar unidades de capacidade de leitura em excesso. Isso é especialmente benéfico para aplicativos que exigem leituras repetidas para chaves individuais.

O DAX é compatível com a criptografia do lado do servidor. Com a criptografia em repouso, os dados persistentes pelo DAX no disco serão criptografados. O DAX grava dados ao disco como parte das alterações de propagação do nó primário para as réplicas de leitura. Para obter mais informações, consulte Criptografia em repouso do DAX .

O DAX também oferece suporte à criptografia em trânsito, garantindo que todas as solicitações e respostas entre a aplicação e o cluster sejam criptografadas por TLS (Transport Level Security) e que as conexões com o cluster possam ser autenticadas pela verificação de um certificado x509 de cluster. Para obter mais informações, consulte Criptografia em trânsito do DAX.

Tópicos

- Casos de uso para o DAX

- Observações sobre o uso do DAX
- DAX: como ele funciona
- Componentes de cluster do DAX
- Criar um cluster do DAX
- Modelos de consistência do DAX e do DynamoDB
- Desenvolver com o cliente do DynamoDB Accelerator (DAX)
- Gerenciar clusters do DAX
- Monitorar o DynamoDB Accelerator
- Instâncias expansíveis T3/T2 do DAX
- Controle de acesso do DAX
- Criptografia em repouso do DAX
- Criptografia em trânsito do DAX
- Usar funções vinculadas ao serviço do IAM para o DAX
- Acessando o DAX em contas da AWS
- Guia de dimensionamento de clusters do DAX

Casos de uso para o DAX

O DAX dá acesso a dados finais consistentes de tabelas do DynamoDB, com latência de microssegundos. Um cluster do DAX multi-AZ pode servir milhões de solicitações por segundo.

O DAX é ideal para os seguintes tipos de aplicações:

- Aplicativos que exigem o melhor tempo de resposta possível para leituras. Alguns exemplos incluem lances em tempo real, jogos sociais e aplicações de negócios. O DAX oferece uma performance de leitura rápida na memória para esses casos de uso.
- Aplicativos que fazem a leitura de um pequeno número de itens com mais frequência do que outros. Por exemplo, considere um sistema de comércio eletrônico que tem uma promoção de um produto popular válida por apenas um dia. Durante a promoção, a demanda por esse produto (e seus dados no DynamoDB) aumentaria drasticamente em comparação a todos os outros produtos. Para mitigar os impactos de uma chave de "aceleração" e uma distribuição de tráfego não uniforme, você pode descarregar as atividades de leitura em um cache do DAX até que essa promoção de um dia acabe.

- Aplicativos que exigem leitura intensa, mas que também são sensíveis aos custos. Com o DynamoDB, você fornece o número de leituras por segundo que a sua aplicação exige. Se as atividades de leitura aumentarem, você poderá aumentar o throughput de leitura provisionado das suas tabelas (a um custo adicional). Como alternativa, é possível descarregar as atividades da sua aplicação em um cluster do DAX e reduzir a quantidade de unidades de capacidade de leitura que você precisa comprar.
- Aplicativos que exigem leituras repetidas em um grande conjunto de dados. Esses aplicativos poderiam desviar os recursos de banco de dados de outros aplicativos. Por exemplo, uma análise de longa execução de dados meteorológicos regionais pode consumir toda a capacidade de leitura em uma tabela do DynamoDB. Essa situação pode afetar negativamente outros aplicativos que precisam acessar os mesmos dados. Com o DAX, a análise meteorológica pode ser realizada com base nos dados em cache.

O DAX não é ideal para os seguintes tipos de aplicação:

- Aplicativos que exigem leituras fortemente consistentes (ou que não toleram leituras eventualmente consistentes).
- Aplicativos que não precisam de tempos de resposta em microssegundos para leituras ou descarregar atividades de leitura repetidas de tabelas subjacentes.
- Aplicações que exigem alto volume de gravações. O alto volume de gravações leva ao aumento da replicação entre nós do DAX em um cluster. Isso causa um aumento no consumo de recursos e no risco de problemas de disponibilidade.
- Aplicações sem muitas leituras repetidas. O DAX tem melhor desempenho quando as taxas de acertos de cache excedem 90%. Taxas de acertos de cache mais baixas aumentam as perdas no cache, o que consome mais recursos em todo o cluster do DAX.

Observações sobre o uso do DAX

- Para obter uma lista das regiões da AWS onde o DAX está disponível, consulte Preços do Amazon DynamoDB.
- O DAX oferece suporte a aplicações escritas em Go, Java, Node.js, Python e .NET usando os clientes fornecidos pela AWS para essas linguagens de programação.
- O DAX só está disponível para a plataforma EC2-VPC.
- A política da função de serviço do cluster do DAX deve permitir a ação `dynamodb:DescribeTable` para manter os metadados sobre a tabela do DynamoDB.

- Os clusters do DAX mantêm metadados sobre os nomes de atributos de itens que armazenam. Esses metadados são mantidos indefinidamente (mesmo depois que o item expira ou é removido do cache). As aplicações que usam um número não vinculado de nomes de atributos podem, com o tempo, provocar exaustão de memória no cluster do DAX. Essa limitação aplica-se somente aos nomes de atributo nível superior, não a nomes de atributo aninhados. Exemplos de nomes de atributo de nível superior problemáticos incluem time stamps, UUIDs e IDs de sessão.

  Essa limitação se aplica a nomes de atributos, não a seus valores. Itens como esses não constituem um problema.

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

  Mas itens como esses serão um problema, se houver um número suficiente deles e cada um tiver um timestamp diferente.

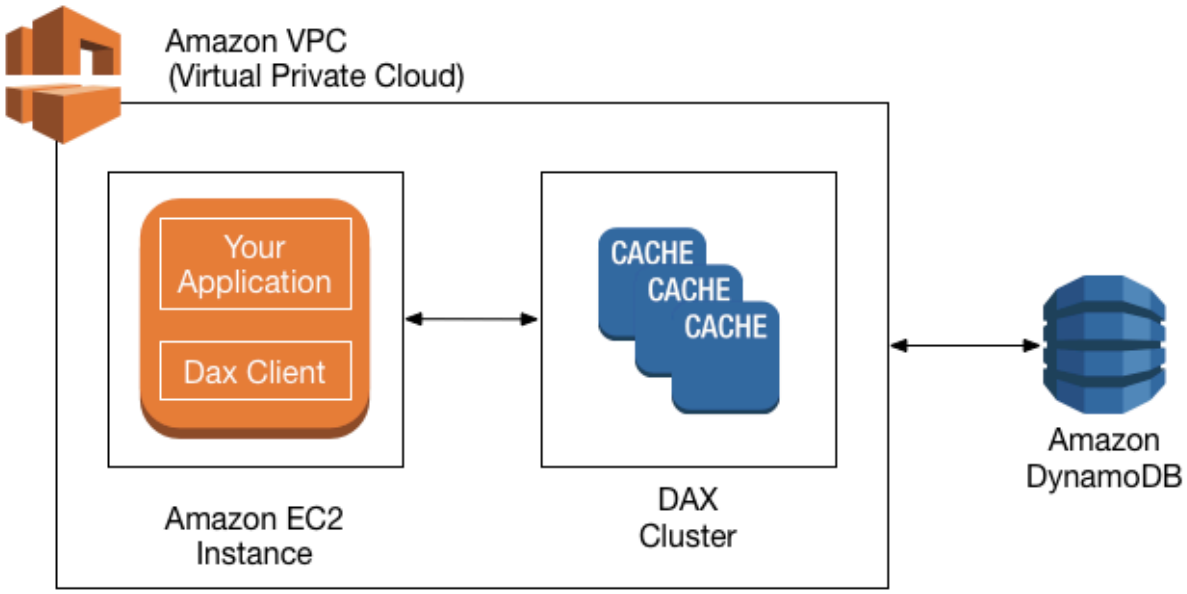
```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "2017-10-24T01:02:03+00:00": "created"
}
```

DAX: como ele funciona

O Amazon DynamoDB Accelerator (DAX) foi projetado para ser executado dentro de um ambiente da Amazon Virtual Private Cloud (Amazon VPC). O serviço Amazon VPC define uma rede virtual que lembra muito um datacenter tradicional. Com uma VPC, você tem controle sobre o intervalo de endereços IP, as sub-redes, as tabelas de roteamento, os gateways de rede e as configurações de segurança. Você pode iniciar um cluster do DAX em sua rede virtual e controlar o acesso ao cluster usando grupos de segurança da Amazon VPC.

> **Note**
>
> Se sua conta da AWS foi criada após 4 de dezembro de 2013, você já tem uma VPC padrão em cada região da AWS. A VPC está pronta para ser usada imediatamente, sem precisar executar quaisquer etapas de configuração adicionais.
> Para obter mais informações, consulte VPC padrão e sub-redes padrão no Guia do usuário da Amazon VPC.

O diagrama a seguir mostra uma visão geral de alto nível do DAX:



Para criar um cluster do DAX, você deve usar o AWS Management Console. A não ser que você especifique o contrário, o cluster do DAX será executado em sua VPC padrão. Para executar sua aplicação, você deve executar uma instância do Amazon EC2 na Amazon VPC. Em seguida, você implanta a aplicação (com o cliente do DAX) na instância do EC2.

Em tempo de execução, o cliente do DAX direciona todas as solicitações da API do DynamoDB da sua aplicação para o cluster do DAX. Se o DAX puder, ele processará uma dessas solicitações da API diretamente. Caso contrário, ele passará a solicitação para o DynamoDB.

Por fim, o cluster do DAX retorna os resultados para sua aplicação.

Tópicos

- Como o DAX processa solicitações
- Cache de itens
- Cache de consultas

Como o DAX processa solicitações

Um cluster do DAX consiste em um ou mais nós. Cada nó executa sua própria instância do software de armazenamento em cache do DAX. Um dos nós serve como o nó primário do cluster. Os nós adicionais (se houver) servem como réplicas de leitura. Para ter mais informações, consulte Nós.

Sua aplicação pode acessar o DAX especificando o endpoint do cluster do DAX. O software cliente do DAX funciona com o endpoint do cluster para executar balanceamento de carga e roteamento inteligentes.

Operações de leitura

O DAX pode responder às seguintes chamadas de API:

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

Se a solicitação especificar leituras finais consistentes (o comportamento padrão), ela tentará ler o item do DAX:

- Se o DAX tiver o item disponível (um acerto de cache), o DAX retornará o item para a aplicação sem acessar o DynamoDB.
- Se o DAX não tiver o item disponível (um erro de cache), o DAX passará a solicitação para o DynamoDB. Ao receber a resposta do DynamoDB, o DAX retorna os resultados para a aplicação. Mas também grava os resultados no cache do nó primário.

> **Note**
>
> Se houver réplicas de leitura no cluster, o DAX manterá automaticamente as réplicas sincronizadas com o nó principal. Para ter mais informações, consulte Clusters.

Se a solicitação especificar leituras fortemente consistentes, o DAX passa a solicitação para o DynamoDB. Os resultados do DynamoDB não são armazenados em cache no DAX. Em vez disso, eles são simplesmente retornados ao aplicativo.

Operações de gravação

As seguintes operações da API do DAX são consideradas "write-through":

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Com essas operações, os dados são gravados primeiro na tabela do DynamoDB e no cluster do DAX. A operação é bem-sucedida somente se os dados são gravados com êxito na tabela e no DAX.

Outras operações

O DAX não reconhece as operações do DynamoDB para gerenciar tabelas (tais como `CreateTable`, `UpdateTable` e assim por diante). Se a sua aplicação precisar executar essas operações, ela precisará acessar o DynamoDB diretamente em vez de usar o DAX.

Para obter informações detalhadas sobre a consistência do DAX e do DynamoDB, consulte Modelos de consistência do DAX e do DynamoDB.

Para obter informações sobre como as transações funcionam no DAX, consulte Usar APIs transacionais no DynamoDB Accelerator (DAX).

Limitação de taxa de solicitações

Se o número de solicitações enviadas ao DAX exceder a capacidade de um nó, o DAX limitará a taxa na qual aceita solicitações adicionais retornando uma ThrottlingException. O DAX avalia

continuamente a utilização da CPU para determinar o volume de solicitações que ela pode processar enquanto mantém um estado de cluster íntegro.

Você pode monitorar a métrica ThrottledRequestCount publicada pelo DAX no Amazon CloudWatch. Se você vir essas exceções regularmente, considere aumentar o cluster.

Cache de itens

O DAX mantém um cache de itens para armazenar os resultados das operações `GetItem` e `BatchGetItem`. Os itens no cache representam dados finais consistentes do DynamoDB e são armazenados por seus valores de chave primária.

Quando uma aplicação envia uma solicitação de `GetItem` ou `BatchGetItem`, o DAX tenta ler os itens diretamente do cache de itens usando os valores de chave especificados. Se os itens forem encontrados (acerto de cache), o DAX os retornará imediatamente para a aplicação. Se os itens não forem encontrados (erro de cache), o DAX envia a solicitação para o DynamoDB. O DynamoDB processa as solicitações usando leituras eventualmente consistentes e retorna os resultados para o DAX. O DAX armazena-os no cache de itens e, em seguida, retorna-os para a aplicação.

O cache de itens tem uma configuração de vida útil (TTL) que é 5 minutos por padrão. O DAX atribui uma marca de data e hora para cada item que ele grava no cache de itens. Um item expira se ele permaneceu no cache por mais tempo que a configuração de TTL. Se você emitir uma solicitação de `GetItem` em um item expirado, isso será considerado um erro de cache, e o DAX enviará a solicitação de `GetItem` ao DynamoDB.

> **Note**
>
> Você pode especificar a configuração de TTL para o cache de itens ao criar um novo cluster do DAX. Para ter mais informações, consulte Gerenciar clusters do DAX .

O DAX também mantém uma lista Last Recently Used (LRU – Menos usados recentemente) para o cache de itens. A lista de LRU rastreia quando um item foi gravado pela primeira vez no cache, e quando o item foi lido pela última vez no cache. Se o cache de itens encher, o DAX removerá os itens mais antigos (mesmo que ainda não tenham expirado) para abrir espaço para novos itens. O algoritmo de LRU está sempre habilitado para o cache de itens, e não pode ser configurado pelo usuário.

Se você especificar zero como a configuração de TTL do cache de itens, os itens no cache de itens só serão atualizados devido a uma remoção por LRU ou a uma operação "write-through".

Para obter informações detalhadas sobre a consistência do cache de itens no DAX, consulte Comportamento do cache de itens do DAX.

Cache de consultas

O DAX também mantém um cache de consultas para armazenar os resultados das operações `Query` e `Scan`. Os itens nesse cache representam conjuntos de resultados de consultas e verificações nas tabelas do DynamoDB. Esses conjuntos de resultados são armazenados por seus valores de parâmetro.

Quando uma aplicação envia uma solicitação de `Query` ou `Scan`, o DAX tenta ler um conjunto de resultados correspondente no cache de consultas usando os valores dos parâmetros especificados. Se o conjunto de resultados for encontrado (acerto de cache), o DAX o retornará para a aplicação imediatamente. Se o conjunto de resultados não for encontrado (erro de cache), o DAX enviará a solicitação para o DynamoDB. O DynamoDB processa as solicitações usando leituras finais consistentes e retorna o conjunto de resultados para o DAX. O DAX o armazena no cache de itens e, em seguida, retorna-o para a aplicação.

> **Note**
>
> Você pode especificar a configuração de TTL para o cache de consultas ao criar um novo cluster do DAX. Para ter mais informações, consulte Gerenciar clusters do DAX .

O DAX também mantém uma lista de LRU para o cache de consultas. A lista rastreia quando um conjunto de resultados foi gravado pela primeira vez no cache, e quando o resultado foi lido pela última vez no cache. Se o cache de consultas encher, o DAX removerá os conjuntos de resultados mais antigos (mesmo que eles ainda não tenham expirado) para abrit espaço para novos conjuntos de resultados. O algoritmo LRU está sempre ativado para o cache de consultas, e não pode ser configurado pelo usuário.

Se você especificar zero como configuração de TTL do cache de consultas, a resposta da consulta não será armazenada em cache.

Para obter informações detalhadas sobre a consistência do cache de consultas no DAX, consulte Comportamento do cache de consultas DAX.

Componentes de cluster do DAX

Um cluster do Amazon DynamoDB Accelerator (DAX) consiste em componentes da infraestrutura da AWS. Esta seção descreve esses componentes e como eles funcionam em conjunto.

Tópicos

- Nós
- Clusters
- Regiões e zonas de disponibilidade
- Grupos de parâmetros
- Grupos de segurança
- ARN do cluster
- Endpoint do cluster
- Endpoints de nó
- Grupos de sub-redes
- Eventos
- Janela de manutenção

Nós

Um nó é o menor bloco de criação de um cluster do DAX. Cada nó executa uma instância do software DAX e mantém uma única réplica dos dados em cache.

Você pode escalar seu cluster do DAX de uma das seguintes formas:

- Adicionando mais nós ao cluster. Isso aumenta o throughput de leitura geral do cluster.
- Ao usar um tipo de nó maior. Tipos de nó maiores fornecem mais capacidade e podem aumentar o throughput. (Você deve criar um novo cluster com o novo tipo de nó.)

Cada nó em um cluster é do mesmo tipo de nó, e executa o mesmo software de cache do DAX. Para obter uma lista de tipos de nós disponíveis, consulte Preços do Amazon DynamoDB.

Clusters

Um cluster é um agrupamento lógico de um ou mais nós que o DAX gerencia como uma unidade. Um dos nós do cluster é designado como o nó primário e os outros nós (se houver) são réplicas de leitura.

O nó primário é responsável pelo seguinte:

- Cumprir solicitações de aplicativo para dados em cache.
- Tratar de operações de gravação para DynamoDB.
- Remover dados do cache, de acordo com a política de remoção do cluster.

Quando alterações são feitas nos dados armazenados em cache no nó primário, o DAX propaga as alterações para todos os nós de réplica de leitura usando logs de replicação. Depois que a confirmação é recebida de todas as réplicas de leitura, o DynamoDB exclui os logs de replicação do nó primário.

Um cluster do DAX pode oferecer suporte a até 11 nós por cluster (o nó primário, mais um máximo de dez réplicas de leitura).

As réplicas de leitura são responsáveis pelo seguinte:

- Cumprir solicitações de aplicativo para dados em cache.
- Remover dados do cache, de acordo com a política de remoção do cluster.

No entanto, ao contrário do nó primário, as réplicas de leitura não gravam no DynamoDB.

As réplicas de leitura têm duas finalidades adicionais:

- Escalabilidade. Se houver um grande número de clientes de aplicações que precisam acessar o DAX simultaneamente, você poderá adicionar mais réplicas para escalabilidade de leitura. O DAX distribuirá a carga uniformemente entre todos os nós no cluster. (Outra forma de aumentar o throughput é usar maiores tipos de nó de cache.)
- Alta disponibilidade. Em caso de falha de um nó principal, o DAX recupera automaticamente uma réplica de leitura e a designa como o novo nó principal. Se um nó de réplica falhar, outros nós no cluster do DAX ainda poderão atender às solicitações até que o nó com falha seja recuperado. Para obter a máxima tolerância a falhas, você deve implantar as réplicas de leitura em Zonas de

disponibilidade separadas. Essa configuração garante que o cluster do DAX continue a funcionar, mesmo que toda uma zona de disponibilidade se torne indisponível.

> **Important**
>
> Para uso em produção, é altamente recomendável usar o DAX com pelo menos três nós e posicionar cada nó em diferentes zonas de disponibilidade. São necessários três nós para que um cluster DAX seja tolerante a falhas.
> Um cluster do DAX pode ser implantado com um ou dois nós para workloads de desenvolvimento ou de teste. Os clusters de um e de dois nós não são tolerantes a falhas; portanto, não convém usar menos de três nós na produção. Se o cluster de um ou de dois nós encontrar erros de software ou de hardware, ele poderá se tornar indisponível ou perder os dados armazenados em cache.

> **Important**
>
> Um cluster do DAX comporta no máximo quinhentas tabelas do DynamoDB. Se você ultrapassar quinhentas tabelas, o cluster poderá sofrer uma degradação na disponibilidade e na performance.

Regiões e zonas de disponibilidade

Um cluster do DAX em uma região da AWS só pode interagir com tabelas do DynamoDB que estão na mesma região. Por esse motivo, é necessário iniciar o cluster do DAX na região correta. Se você tiver tabelas do DynamoDB em outras regiões, inicie os clusters do DAX também nessas regiões.

Cada região é projetada para ser completamente isolada das outras. Dentro de cada região há várias zonas de disponibilidade. Ao iniciar seus nós em diferentes zonas de disponibilidade, você é capaz de alcançar o máximo possível de tolerância a falhas.

> **Important**
>
> Não coloque todos os nós do cluster em uma única zona de disponibilidade. Nessa configuração, o cluster do DAX se tornará indisponível no caso de uma falha na zona de disponibilidade.

Para uso em produção, é altamente recomendável usar o DAX com pelo menos três nós e posicionar cada nó em diferentes zonas de disponibilidade. São necessários três nós para que um cluster DAX seja tolerante a falhas.
Um cluster do DAX pode ser implantado com um ou dois nós para workloads de desenvolvimento ou de teste. Os clusters de um e dois nós não são tolerantes a falhas, portanto, não convém usar menos de três nós na produção. Se o cluster de um ou dois nós encontrar erros de software ou de hardware, ele poderá se tornar indisponível ou perder os dados armazenados em cache.

Grupos de parâmetros

Os grupos de parâmetros são usados para gerenciar as configurações de tempo de execução dos clusters do DAX. O DAX tem vários parâmetros que você pode usar para otimizar a performance (como a definição de uma política de TTL para dados armazenados em cache). Um grupo de parâmetros é um conjunto de parâmetros que você pode aplicar a um cluster. Dessa maneira, você garante que todos os nós desse cluster sejam configurados exatamente da mesma forma.

Grupos de segurança

Um cluster do DAX é executado em um ambiente da Amazon Virtual Private Cloud (Amazon VPC). Esse ambiente é uma rede virtual dedicada à sua conta da AWS e é isolada de outras VPCs. Um grupo de segurança atua como um firewall virtual para a VPC, permitindo que você controle o tráfego de entrada e saída de rede.

Ao iniciar um cluster na VPC, você adiciona uma regra de entrada ao grupo de segurança para permitir tráfego de rede de entrada. A regra de entrada especifica o protocolo (TCP) e o número da porta (8111) para seu cluster. Depois que você adiciona essa regra de entrada ao grupo de segurança, as aplicações em execução na VPC podem acessar o cluster do DAX.

ARN do cluster

A cada cluster do DAX é atribuído um nome do recurso da Amazon (ARN). O formato do ARN é o seguinte.

```
arn:aws:dax:region:accountID:cache/clusterName
```

Você usa o ARN do cluster em uma política do IAM para definir permissões para operações da API do DAX. Para ter mais informações, consulte Controle de acesso do DAX.

Endpoint do cluster

Cada cluster do DAX fornece um endpoint de cluster para ser usado pela sua aplicação. Ao acessar o cluster usando o endpoint, o aplicativo não precisa saber os nomes de hosts e os números de portas de nós individuais no cluster. O aplicativo "conhece" automaticamente todos os nós no cluster, mesmo que você adicione ou remova réplicas de leitura.

Veja a seguir um exemplo de endpoint de cluster na região us-east-1 que não está configurado para usar criptografia em trânsito.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Veja a seguir um exemplo de endpoint de cluster na mesma região que está configurada para usar criptografia em trânsito.

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Endpoints de nó

Cada um dos nós individuais em um cluster do DAX tem seu próprio nome de host e número de porta. Veja a seguir um exemplo de endpoint de nó.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

O aplicativo pode acessar um nó diretamente usando o endpoint desse nó. No entanto, recomendamos tratar o cluster do DAX como uma unidade única e acessá-lo usando o endpoint do cluster. O endpoint do cluster faz com que o aplicativo não precise manter e atualizar uma lista de nós quando você adiciona ou remove nós do cluster.

Grupos de sub-redes

O acesso aos nós do cluster do DAX é restrito a aplicações em execução em instâncias do Amazon EC2 em um ambiente de Amazon VPC. Você pode usar grupos de sub-redes para conceder acesso ao cluster a partir de instâncias do Amazon EC2 em execução em sub-redes específicas. Um grupo de sub-redes é uma coleção de sub-redes (normalmente privadas) que você pode designar para clusters em execução em um ambiente de Amazon VPC.

Ao criar um cluster do DAX, você deve especificar um grupo de sub-redes. O DAX utiliza esse grupo de sub-redes para selecionar uma sub-rede e endereços IP nessa sub-rede para associar aos seus nós.

Eventos

O DAX registra eventos significativos em seus clusters, como uma falha ao adicionar um nó, êxito ao adicionar um nó ou alterações nos security groups. Ao monitorar eventos importantes, você pode saber o estado atual dos seus clusters e, dependendo do evento, poderá executar a ação corretiva. Você pode acessar esses eventos usando o AWS Management Console ou a ação `DescribeEvents` na API de gerenciamento do DAX.

Também é possível solicitar que essas notificações sejam enviadas a um tópico específico do Amazon Simple Notification Service (Amazon SNS). Com isso, você ficará sabendo imediatamente quando um evento ocorrer em seu cluster do DAX.

Janela de manutenção

Cada cluster tem uma janela de manutenção semanal para aplicação das alterações do sistema. À medida que as alterações são aplicadas sequencialmente, um nó existente é substituído e um novo nó com as alterações aplicadas é adicionado ao cluster. Durante esse período, sua aplicação pode observar controles de utilização ou erros transitórios. Portanto, recomendamos que você agende a janela de manutenção durante o período de menor uso e ajuste essa programação periodicamente conforme necessário. Você pode especificar um intervalo de tempo de até 24 horas de duração durante o qual todas as atividades de manutenção solicitadas devem ocorrer.

Se você não especificar uma janela de manutenção de sua preferência ao criar ou modificar um cluster de cache, o DAX atribuirá uma janela de manutenção de 60 minutos em um dia aleatório da semana. Essa janela de 60 minutos é selecionada aleatoriamente dentro de um bloco de 8 horas para cada Região da AWS. A seguinte tabela lista os blocos de tempo de cada região dos quais as janelas de manutenção padrão são atribuídas.

Código da região	Nome da região	Janela de manutenção
ap-northeast-1	Região Ásia-Pacífico (Tóquio)	13h às 21h (UTC)
ap-southeast-1	Região Ásia-Pacífico (Singapura)	Das 14:00 às 22:00 (UTC)
ap-southeast-2	Ásia-Pacífico (Sydney)	12h às 20h (UTC)
ap-south-1	Região Ásia-Pacífico (Mumbai)	17h30 à 1h30 (UTC)

Código da região	Nome da região	Janela de manutenção
cn-northwest-1	Região China (Ningxia)	Das 23h às 7h (UTC)
cn-north-1	Região China (Pequim)	Das 14:00 às 22:00 (UTC)
eu-central-1	Região Europa (Frankfurt)	Das 23h às 7h (UTC)
eu-north-1	Região Europa (Estocolmo)	1h às 9h UTC
eu-south-2	Região Europa (Espanha)	Das 21:00 às 05:00 (UTC)
eu-west-1	Região Europa (Irlanda)	22h às 6h (UTC)
eu-west-2	Região Europa (Londres)	Das 23h às 7h (UTC)
eu-west-3	Região Europa (Paris)	Das 23h às 7h (UTC)
sa-east-1	Região América do Sul (São Paulo)	1h às 9h UTC
us-east-1	Região Leste dos EUA (Norte da Virgínia)	3h às 7h (UTC)
us-east-2	Região Leste dos EUA (Ohio)	Das 23h às 7h (UTC)
us-west-1	Região Oeste dos EUA (Norte da Califórnia)	Das 6h às 14h (UTC)
us-west-2	Região Oeste dos EUA (Oregon)	Das 6h às 14h (UTC)

Criar um cluster do DAX

Esta seção orienta você durante a primeira configuração e o uso do Amazon DynamoDB Accelerator (DAX) no seu ambiente Amazon Virtual Private Cloud (Amazon VPC) padrão. Você pode criar seu primeiro cluster do DAX usando a AWS Command Line Interface (AWS CLI) ou o AWS Management Console.

Depois de criar o cluster do DAX, você poderá acessá-lo de uma instância do Amazon EC2 em execução na mesma VPC. Você pode usar o cluster do DAX com um programa de aplicação. Para obter mais informações, consulte Desenvolver com o cliente do DynamoDB Accelerator (DAX).

Tópicos

- Criar um perfil de serviço do IAM para o DAX acessar o DynamoDB
- Criar um cluster do DAX usando a AWS CLI
- Criar um cluster do DAX usando o AWS Management Console

Criar um perfil de serviço do IAM para o DAX acessar o DynamoDB

Para que o cluster do DAX acesse as tabelas do DynamoDB em seu nome, será necessário criar uma função de serviço. Uma função de serviço é uma função do AWS Identity and Access Management (IAM) que autoriza um serviço da AWS a atuar em seu nome. A função de serviço permite que o DAX acesse as tabelas do DynamoDB como se você mesmo estivesse acessando essas tabelas. Você deve criar a função de serviço antes de criar o cluster do DAX.

Se você estiver usando o console, o fluxo de trabalho para a criação de um cluster verifica a presença de uma função de serviço do DAX pré-existente. Se nenhuma for encontrada, o console criará uma nova função de serviço para você. Para obter mais informações, consulte the section called “Etapa 2: criar um cluster do DAX”.

Se você estiver usando a AWS CLI, especifique uma função de serviço do DAX criada anteriormente. Caso contrário, é necessário criar uma nova função de serviço com antecedência. Para obter mais informações, consulte Etapa 1: criar um perfil de serviço do IAM para DAX para acessar o DynamoDB usando a AWS CLI.

Permissões necessárias para criar um perfil de serviço

A política `AdministratorAccess` gerenciada pela AWS fornece todas as permissões necessárias para criar um cluster do DAX e uma função de serviço. Se o usuário tiver `AdministratorAccess` anexada, nenhuma ação adicional será necessária.

Caso contrário, você deverá adicionar as seguintes permissões à política do IAM para que o usuário possa criar o perfil de serviço:

- `iam:CreateRole`

- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

Anexe essas permissões ao usuário que está tentando executar a ação.

> **Note**
>
> As permissões `iam:CreateRole`, `iam:CreatePolicy`, `iam:AttachRolePolicy` e `iam:PassRole` não são incluídas nas políticas gerenciadas pela AWS para DynamoDB. Isso é por design, porque essas permissões fornecem a possibilidade de escalonamento de privilégios: isto é, um usuário poderia usar essas permissões para criar uma nova política de administrador e anexá-la a uma função existente. Por esse motivo, você (o administrador do seu cluster do DAX) deve adicionar explicitamente essas permissões à sua política.

Solução de problemas

Se sua política de usuário não tiver as permissões `iam:CreateRole`, `iam:CreatePolicy` e `iam:AttachPolicy`, você receberá mensagens de erro. A seguinte tabela lista essas mensagens e descreve como corrigir os problemas.

Se você vir essa mensagem de erro...	Faça o seguinte:
`User: arn:aws:iam::` *`accountID`* `:user/`*`userName`* `is not authorized to perform: iam:CreateRole on resource: arn:aws:iam::` *`accountID`* `:role/service-role/` *`roleName`*	Adicione `iam:CreateRole` à sua política de usuário.
`User: arn:aws:iam::` *`accountID`* `:user/`*`userName`* `is not authorized to perform: iam:CreatePolicy on resource: policy` *`policyName`*	Adicione `iam:CreatePolicy` à sua política de usuário.
`User: arn:aws:iam::` *`accountID`* `:user/`*`userName`* `is not authorized`	Adicione `iam:AttachRolePolicy` à sua política de usuário.

Se você vir essa mensagem de erro...	Faça o seguinte:
`to perform: iam:AttachRolePolicy on resource: role` *daxServiceRole*	

Para obter mais informações sobre políticas do IAM obrigatórias para a administração de cluster do DAX, consulte Controle de acesso do DAX.

Criar um cluster do DAX usando a AWS CLI

Esta seção descreve como criar um cluster do Amazon DynamoDB Accelerator (DAX) usando a AWS Command Line Interface (AWS CLI). Caso ainda não tenha feito isso, você deve instalar e configurar a AWS CLI. Para fazer isso, consulte os seguintes tópicos no Guia do usuário do AWS Command Line Interface:

- Instalar a AWS CLI
- Configurar a AWS CLI

> **Important**
>
> Para gerenciar os clusters do DAX usando a AWS CLI, instale ou faça upgrade para a versão 1.11.110 ou superior.

Todos os exemplos da AWS CLI usam a região `us-west-2` e IDs de conta fictícios.

Tópicos

- Etapa 1: criar um perfil de serviço do IAM para DAX para acessar o DynamoDB usando a AWS CLI
- Etapa 2: criar um grupo de sub-redes
- Etapa 3: criar um cluster do DAX usando a AWS CLI
- Etapa 4: configurar regras de entrada para grupo de segurança usando a AWS CLI

Etapa 1: criar um perfil de serviço do IAM para DAX para acessar o DynamoDB usando a AWS CLI

Para poder criar um cluster do Amazon DynamoDB Accelerator (DAX) você precisa criar uma função de serviço para ele. Uma função de serviço é uma função do AWS Identity and Access Management (IAM) que autoriza um serviço da AWS a atuar em seu nome. A função de serviço permite que o DAX acesse as tabelas do DynamoDB como se você mesmo estivesse acessando essas tabelas.

Nesta etapa, você criará uma política do IAM e anexará essa política a uma função do IAM. Isso permite atribuir a função a um cluster do DAX para que ele possa executar operações do DynamoDB em seu nome.

Como criar uma função de serviço do IAM para o DAX

1. Crie um arquivo denominado `service-trust-relationship.json` com o seguinte conteúdo:

   JSON

   ```
   {
       "Version": "2012-10-17"       ,
       "Statement": [
          {
               "Effect": "Allow",
               "Principal": {
                   "Service": "dax.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

2. Crie a função de serviço.

   ```
   aws iam create-role \
       --role-name DAXServiceRoleForDynamoDBAccess \
       --assume-role-policy-document file://service-trust-relationship.json
   ```

3. Crie um arquivo denominado `service-role-policy.json` com o seguinte conteúdo:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:*"
            ]
        }
    ]
}
```

Substitua *accountID* pelo seu ID de conta da AWS. Para encontrar o ID da conta da AWS, no canto superior direito do console, escolha seu ID de login. Seu ID de conta da AWS aparece no menu suspenso.

No nome de recurso da Amazon (ARN) do exemplo, *accountID* deve ser um número de 12 dígitos. Não use hifens ou qualquer outro sinal de pontuação.

4. Crie uma política do IAM para a função de serviço.

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

Na saída, anote o ARN da política que você criou, como no exemplo abaixo.

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. Anexe a política à função de serviço. Substitua *arn* no seguinte código pelo ARN real da função da etapa anterior.

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

Depois, especifique um grupo de sub-redes para a VPC padrão. Um grupo de sub-redes é uma coleção de uma ou mais sub-redes na sua VPC. Consulte Etapa 2: criar um grupo de sub-redes.

Etapa 2: criar um grupo de sub-redes

Siga este procedimento para criar um grupo de sub-redes para o cluster do Amazon DynamoDB Accelerator (DAX) usando a AWS Command Line Interface (AWS CLI).

> **Note**
>
> Se você já criou um grupo de sub-redes para sua VPC padrão, pode ignorar esta etapa.

O DAX foi projetado para ser executado dentro de um ambiente da Amazon Virtual Private Cloud (Amazon VPC). Se sua conta da AWS foi criada após 4 de dezembro de 2013, você já tem uma VPC padrão em cada região da AWS. Para obter mais informações, consulte VPC padrão e sub-redes padrão no Guia do usuário da Amazon VPC.

> **Note**
>
> A VPC com esse cluster do DAX pode conter outros recursos e até mesmo endpoints da VPC para os outros serviços, exceto o endpoint da VPC para ElastiCache, e pode gerar um erro nas operações do cluster do DAX.

Para criar um grupo de sub-redes

1. Para determinar o identificador da VPC padrão, insira o seguinte comando.

```
aws ec2 describe-vpcs
```

Na saída, anote o identificador da VPC padrão, como no seguinte exemplo.

```
vpc-12345678
```

2. Determine os IDs das sub-redes associadas à VPC padrão. Substitua *vpcID* pelo seu ID de VPC real. Por exemplo, `vpc-12345678`.

```
aws ec2 describe-subnets \
    --filters "Name=vpc-id,Values=vpcID" \
    --query "Subnets[*].SubnetId"
```

Na saída, observe os identificadores de sub-rede — por exemplo, `subnet-11111111`.

3. Crie o grupo de sub-redes. Certifique-se de especificar pelo menos um ID de sub-rede no parâmetro `--subnet-ids`.

```
aws dax create-subnet-group \
    --subnet-group-name my-subnet-group \
    --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
```

Para criar o cluster, consulte Etapa 3: criar um cluster do DAX usando a AWS CLI.

Etapa 3: criar um cluster do DAX usando a AWS CLI

Siga este procedimento para usar a AWS Command Line Interface (AWS CLI) para criar um cluster do Amazon DynamoDB Accelerator (DAX) em sua Amazon VPC padrão.

Para criar um cluster do DAX

1. Obtenha o nome de recurso da Amazon (ARN) para a sua função de serviço.

```
aws iam get-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --query "Role.Arn" --output text
```

Na saída, anote o ARN da função de serviço, como no seguinte exemplo.

```
arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
```

2. Crie um cluster do DAX. Substitua *roleARN* pelo ARN da etapa anterior.

```
aws dax create-cluster \
    --cluster-name mydaxcluster \
    --node-type dax.r4.large \
    --replication-factor 3 \
    --iam-role-arn roleARN \
    --subnet-group my-subnet-group \
    --sse-specification Enabled=true \
    --region us-west-2
```

Todos os nós do cluster são do tipo `dax.r4.large` (`--node-type`). Há três nós (`--replication-factor`): um nó primário e duas réplicas.

> **Note**
>
> Como `sudo` e `grep` são palavras-chave reservadas, você não pode criar um cluster do DAX com essas palavras no nome do cluster. Por exemplo, `sudo` e `sudocluster` são nomes de cluster inválidos.

Para visualizar o status do cluster, insira o seguinte comando,

```
aws dax describe-clusters
```

O status é mostrado na saída. Por exemplo, `"Status": "creating"`.

> **Note**
>
> A criação do cluster demora vários minutos. Quando o cluster estiver pronto, seu status mudará para `available`. Enquanto isso, prossiga para Etapa 4: configurar regras de entrada para grupo de segurança usando a AWS CLI e siga as instruções contidas ali.

Etapa 4: configurar regras de entrada para grupo de segurança usando a AWS CLI

Os nós no cluster do Amazon DynamoDB Accelerator (DAX) usam o grupo de segurança padrão para a Amazon VPC. Para o grupo de segurança padrão, você deve autorizar o tráfego de entrada

na porta TCP 8111 para clusters não criptografados ou a porta 9111 para clusters criptografados. Isso permitirá que as instâncias do Amazon EC2 em sua Amazon VPC acessem seu cluster do DAX.

> **Note**
>
> Se você tiver iniciado o cluster do DAX com um grupo de segurança diferente (que não seja o `default`), será necessário executar este procedimento para esse grupo.

Como configurar regras de entrada para o grupo de segurança

1. Para determinar o identificador do grupo de segurança padrão, digite o seguinte comando. Substitua *vpcID* pelo seu ID de VPC real (de Etapa 2: criar um grupo de sub-redes).

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   Na saída, anote o identificador do grupo de segurança. Por exemplo, `sg-01234567`.

2. Insira o seguinte. Substitua *sgID* pelo seu identificador de grupo de segurança real. Use a porta 8111 para clusters não criptografados e a porta 9111 para clusters criptografados.

   ```
   aws ec2 authorize-security-group-ingress \
       --group-id sgID --protocol tcp --port 8111
   ```

Criar um cluster do DAX usando o AWS Management Console

Esta seção descreve como criar um cluster do Amazon DynamoDB Accelerator (DAX) usando o AWS Management Console.

Tópicos

- Etapa 1: criar um grupo de sub-redes usando o AWS Management Console
- Etapa 2: criar um cluster do DAX usando o AWS Management Console
- Etapa 3: configurar regras de entrada para grupo de segurança usando a AWS Management Console

Etapa 1: criar um grupo de sub-redes usando o AWS Management Console

Siga este procedimento para criar um grupo de sub-redes para o cluster do Amazon DynamoDB Accelerator (DAX) usando o AWS Management Console.

> **Note**
>
> Se você já criou um grupo de sub-redes para sua VPC padrão, pode ignorar esta etapa.

O DAX foi projetado para ser executado dentro de um ambiente da Amazon Virtual Private Cloud (Amazon VPC). Se sua conta da AWS foi criada após 4 de dezembro de 2013, você já tem uma VPC padrão em cada região da AWS. Para obter mais informações, consulte VPC padrão e sub-redes padrão no Guia do usuário da Amazon VPC.

Como parte do processo de criação de um cluster do DAX, você deve especificar um grupo de sub-redes. Um grupo de sub-redes é uma coleção de uma ou mais sub-redes na VPC. Quando você cria o cluster do DAX, os nós são implantados nas sub-redes dentro do grupo de sub-redes.

> **Note**
>
> A VPC com esse cluster do DAX pode conter outros recursos e até mesmo endpoints da VPC para os outros serviços, exceto o endpoint da VPC para ElastiCache, e pode gerar um erro nas operações do cluster do DAX.

Como criar um grupo de sub-redes

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, em DAX, escolha Subnet groups (grupos de sub-rede).
3. Selecione Create subnet group (Criar grupo de sub-redes).
4. Na janela Create subnet group (Criar grupo de sub-redes), faça o seguinte:
    a. Name (Nome): insira um nome curto para o grupo de sub-redes.
    b. Description (Descrição): insira uma descrição para o grupo de sub-redes.
    c. VPC ID (ID da VPC): escolha o identificador do seu ambiente da Amazon VPC.
    d. Subnets (Sub-redes): escolha uma ou mais sub-redes na lista.

> **Note**
>
> As sub-redes são distribuídas por várias Zonas de disponibilidade. Se você planeja criar um cluster do DAX com vários nós (um nó primário e uma ou mais réplicas de leitura), convém escolher vários IDs de sub-redes. O DAX pode implantar os nós de cluster em várias zonas de disponibilidade. Se uma zona de disponibilidade se tornar indisponível, o DAX fará o failover em uma zona de disponibilidade sobrevivente. O cluster do DAX continuará a funcionar sem interrupção.

Quando estiver satisfeito com as configurações, escolha Create subnet group (Criar grupo de sub-rede).

Para criar o cluster, consulte Etapa 2: criar um cluster do DAX usando o AWS Management Console.

Etapa 2: criar um cluster do DAX usando o AWS Management Console

Siga este procedimento para criar um cluster do Amazon DynamoDB Accelerator (DAX) em sua Amazon VPC padrão.

Como criar um cluster do DAX

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, em DAX, escolha Clusters.
3. Selecione Criar cluster.
4. Na janela Create cluster (Criar cluster), faça o seguinte:
   a. Cluster name (Nome do cluster): insira um nome curto para o seu cluster do DAX.

      > **Note**
      >
      > Como `sudo` e `grep` são palavras-chave reservadas, você não pode criar um cluster do DAX com essas palavras no nome do cluster. Por exemplo, `sudo` e `sudocluster` são nomes de cluster inválidos.

   b. Cluster description (Descrição do cluster): insira uma descrição para o cluster.
   c. Node types (Tipos de nó): escolha o tipo de nó para todos os nós do cluster.

d. Cluster size (Tamanho do cluster): escolha o número de nós no cluster. Um cluster consiste em um nó primário e em até nove réplicas de leitura.

> **Note**
>
> Para criar um cluster com um único nó, escolha 1. O cluster será composto por um único nó primário.
> Para criar um cluster de vários nós, escolha um número entre 3 (um nó primário e duas réplicas de leitura) e 10 (um nó primário e nove réplicas de leitura).
>
> > **Important**
> >
> > Para uso em produção, é altamente recomendável usar o DAX com pelo menos três nós e posicionar cada nó em diferentes zonas de disponibilidade. São necessários três nós para que um cluster DAX seja tolerante a falhas.
> > Um cluster do DAX pode ser implantado com um ou dois nós para workloads de desenvolvimento ou de teste. Os clusters de um e dois nós não são tolerantes a falhas, portanto, não convém usar menos de três nós na produção. Se o cluster de um ou dois nós encontrar erros de software ou de hardware, ele poderá se tornar indisponível ou perder os dados armazenados em cache.

e. Escolha Próximo.

f. Subnet group — SelecioneEscolha existente e escolha o grupo de sub-redes que você criou em Etapa 1: criar um grupo de sub-redes usando o AWS Management Console.

g. Controle de acesso — Selecione o grupo de segurança padrão.

h. Zonas de disponibilidade (AZ) — Escolha Automatic.

i. Escolha Next (Próximo).

j. IAM service role for DynamoDB access (Função de serviço do IAM para acesso ao DynamoDB): selecione Create new (Criar novo) e insira as seguintes informações:

- IAM role name (Nome da função do IAM): insira um nome para a função do IAM, por exemplo, `DAXServiceRole`. O console cria uma nova função do IAM, e o cluster do DAX assume essa função em tempo de execução.
- Selecione a caixa ao lado de Create policy (Criar política).

- IAM role policy (Política da função do IAM): escolha Read/Write (Leitura/Gravação) Isso permite que o cluster do DAX execute operações de leitura e gravação no DynamoDB.
- Novo nome da política do IAM: esse campo será preenchido quando você inserir o nome do perfil do IAM. Você também pode inserir um nome para uma política do IAM; por exemplo, `DAXServicePolicy`. O console criará uma nova política do IAM e anexará essa política à função do IAM.
- Acesso às tabelas do DynamoDB: escolha Todas as tabelas.

k. Criptografia: escolha Ativar a criptografia em repouso e Ativar criptografia em trânsito. Para obter mais informações, consulte Criptografia em repouso do DAX e Criptografia em trânsito do DAX.

Uma função de serviço separada para o DAX acessar o Amazon EC2 também é necessária. O DAX cria automaticamente essa função de serviço para você. Para obter mais informações, consulte Usar funções vinculadas ao serviço para o DAX.

5. Quando as configurações estiverem como você deseja, escolha Próximo.
6. Grupo de parâmetros: escolha Escolher existente.
7. Janela de manutenção: escolha Sem preferência se você não tiver uma preferência quando as atualizações de software forem aplicadas ou escolha Especificar janela de tempo e forneça as opções de Dia da semana, Horário (UTC) e Começar dentro de (horas) para programar a janela de manutenção.
8. Tags: escolha Adicionar nova tag para inserir um par de chave-valor para fins de marcação.
9. Escolha Próximo.

Na tela Analisar e criar, você pode revisar todas as configurações. Se estiver tudo pronto para criar o cluster, escolha Criar cluster.

Na tela Clusters, o cluster do DAX estará listado com um status de Creating (Em criação).

> **Note**
>
> A criação do cluster demora vários minutos. Quando o cluster estiver pronto, seu status mudará para Available (Disponível).
> Enquanto isso, prossiga para Etapa 3: configurar regras de entrada para grupo de segurança usando a AWS Management Console e siga as instruções contidas ali.

Etapa 3: configurar regras de entrada para grupo de segurança usando a AWS Management Console

O cluster do Amazon DynamoDB Accelerator (DAX) comunica-se via porta TCP 8111 (para clusters não criptografados) ou 9111 (para clusters criptografados), portanto, você deve autorizar o tráfego de entrada nessa porta. Isso permitirá que as instâncias do Amazon EC2 em sua Amazon VPC acessem seu cluster do DAX.

> **Note**
>
> Se você tiver iniciado o cluster do DAX com um grupo de segurança diferente (que não seja o `default`), será necessário executar este procedimento para esse grupo.

Como configurar regras de entrada para o grupo de segurança

1. Abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. No painel de navegação, selecione Grupos de segurança.
3. Escolha o grupo de segurança default (padrão). No menu Actions (Ações), escolha Edit inbound rules (Editar regras de entrada).
4. Escolha Add Rule (Adicionar regra) e insira as informações a seguir:
   - Port Range (Intervalo de portas): insira 8111 (se o cluster não for criptografado) ou 9111 (se o cluster for criptografado).
   - Origem: deixe como Personalizado e escolha o campo de pesquisa à direita. Um menu suspenso será exibido. Escolha o identificador para o grupo de segurança padrão.
5. Escolha Salvar regras para salvar as alterações.
6. Para atualizar o nome no console, acesse a propriedade Nome e selecione a opção Editar que é exibida.

Modelos de consistência do DAX e do DynamoDB

O Amazon DynamoDB Accelerator (DAX) é um serviço de armazenamento em cache por gravação simultânea (write-through) projetado para simplificar o processo de adição de um cache a tabelas do DynamoDB. Como o DAX opera separadamente do DynamoDB, é importante compreender os

modelos de consistência do DAX e do DynamoDB para garantir que as aplicações se comportem conforme esperado.

Em muitos casos de uso, a maneira como a aplicação usa o DAX afeta a consistência dos dados dentro do cluster do DAX e a consistência dos dados entre o DAX e o DynamoDB.

Tópicos

- Consistência entre nós de cluster do DAX
- Comportamento do cache de itens do DAX
- Comportamento do cache de consultas DAX
- Leituras altamente consistentes e transacionais
- Armazenamento em cache negativo
- Estratégias para gravações

Consistência entre nós de cluster do DAX

Para obter alta disponibilidade para sua aplicação, recomendamos provisionar o cluster do DAX com pelo menos três nós. Além disso, coloque esses nós em várias zonas de disponibilidade dentro de uma região.

Quando o cluster do DAX estiver em execução, ele replicará os dados entre todos os nós do cluster (supondo que você tenha provisionado mais de um nó). Considere uma aplicação que realiza uma operação `UpdateItem` bem-sucedida usando o DAX. Essa ação faz com que o cache de itens no nó primário seja modificado com o novo valor. Esse valor é, então, replicado em todos os outros nós no cluster. Essa replicação é eventualmente consistente e costuma demorar menos de um segundo para ser concluída.

Nesse cenário, é possível que dois clientes leiam a mesma chave do mesmo cluster do DAX, mas recebam valores diferentes, dependendo do nó que cada cliente acessou. Os nós estarão todos consistentes quando a atualização tiver sido totalmente replicada por todos os nós do cluster. (Esse comportamento é semelhante à natureza final consistente do DynamoDB.)

Se você estiver criando uma aplicação que usa o DAX, essa aplicação deverá ser projetada para tolerância a dados finais consistentes.

Comportamento do cache de itens do DAX

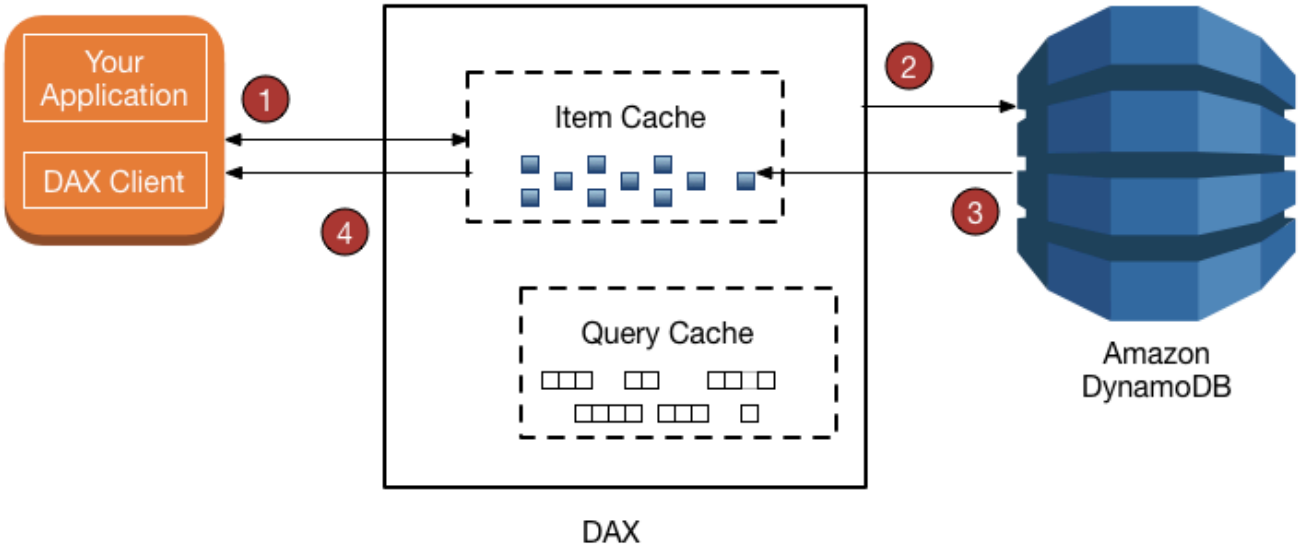
Cada cluster do DAX tem dois caches distintos: um cache de itens e um cache de consultas. Para ter mais informações, consulte DAX: como ele funciona.

Esta seção aborda as implicações de consistência da leitura e da gravação no cache de itens do DAX.

Consistência de leituras

Com o DynamoDB, por padrão, a operação `GetItem` executa uma leitura final consistente. Suponha que você use `UpdateItem` com o cliente do DynamoDB. Se você tentar ler o mesmo item imediatamente, poderá ver os dados no estado em que estavam antes da atualização. Isso acontece devido ao atraso da propagação em todos os locais de armazenamento do DynamoDB. A consistência normalmente é atingida em segundos. Portanto, se você repetir a leitura, provavelmente verá o item atualizado.

Ao usar `GetItem` com o cliente do DAX, a operação (neste caso, uma leitura final consistente) ocorre conforme mostrado abaixo.



1. O cliente do DAX emite uma solicitação `GetItem`. O DAX tenta ler o item solicitado do cache de itens. Se o item estiver no cache (acerto no cache), o DAX o retornará à aplicação.

2. Se o item não estiver disponível (erro de cache), o DAX realizará uma operação final consistente `GetItem` no DynamoDB.
3. O DynamoDB retornará o item solicitado, e o DAX o armazenará no cache de itens.
4. O DAX retornará o item para a aplicação.
5. (Não mostrado) Se o cluster do DAX contiver mais de um nó, o item será replicado em todos os outros nós do cluster.

O item permanece no cache de itens do DAX, sujeito à configuração de vida útil (TTL) e ao algoritmo de Least Recently Used (LRU – Menos usado recentemente) do cache. Para ter mais informações, consulte DAX: como ele funciona.

No entanto, durante esse período, o DAX não lê novamente o item no DynamoDB. Se outra pessoa atualizar o item usando um cliente do DynamoDB, ignorando totalmente o DAX, uma solicitação `GetItem` usando o cliente do DAX produzirá resultados diferentes da mesma solicitação `GetItem` usando o cliente do DynamoDB. Nesse cenário, o DAX e o DynamoDB manterão valores inconsistentes para a mesma chave até que o TTL do item do DAX expire.

Se uma aplicação modificar dados em uma tabela subjacente do DynamoDB, ignorando o DAX, a aplicação precisará antecipar e tolerar as inconsistências de dados que possam surgir.

> **Note**
>
> Além do `GetItem`, o cliente do DAX também oferece suporte a solicitações `BatchGetItem`. `BatchGetItem` é essencialmente um wrapper ao redor de uma ou mais solicitações `GetItem` e, dessa forma, o DAX trata cada uma delas como uma operação `GetItem` individual.

Consistência de gravações

O DAX é um cache write-through, o que simplifica o processo de manter o cache de itens do DAX consistente com as tabelas do DynamoDB subjacentes.

O cliente do DAX oferece suporte às mesmas operações da API de gravação que o DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` e `TransactWriteItems`). Quando você usa essas operações com o cliente do DAX, os itens são modificados tanto no DAX quanto no DynamoDB. O DAX atualizará os itens em seu cache de itens, independentemente do valor de TTL desses itens.

Por exemplo, suponha que você emita uma solicitação `GetItem` do cliente do DAX para ler um item da tabela `ProductCatalog`. (A chave de partição é `Id` e não há uma chave de classificação). Você recupera o item cujo `Id` é `101`. O valor`QuantityOnHand` para esse item é `42`. O DAX armazena o item em seu cache de itens com um TTL específico. Para este exemplo, suponha que o TTL é de 10 minutos. Três minutos depois, outra aplicação usa o cliente do DAX para atualizar o mesmo item de forma que o valor de `QuantityOnHand` agora é `41`. Supondo que o item não seja atualizado novamente, qualquer leitura subsequente do mesmo item durante os próximos dez minutos retornará o valor armazenado em cache a `QuantityOnHand` (`41`).

Como o DAX processa gravações

O DAX foi projetado para aplicações que exigem leituras de alta performance. Por ser um cache write-through, o DAX passa suas gravações para o DynamoDB de forma síncrona e, em seguida, replica assíncrona e automaticamente as atualizações resultantes para o cache de itens em todos os nós do cluster. Você não precisa gerenciar a lógica de invalidação do cache, pois o DAX lida com ela automaticamente.

O DAX oferece suporte às seguintes operações de gravação: `PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` e `TransactWriteItems`.

Quando você envia uma solicitação `PutItem`, `UpdateItem`, `DeleteItem` ou `BatchWriteItem` ao DAX:

- O DAX envia a solicitação para o DynamoDB.
- O DynamoDB responde ao DAX, confirmando que a gravação foi bem-sucedida.
- O DAX grava o item em seu cache de itens.
- O DAX retorna uma resposta de êxito ao solicitante.

Quando você envia uma solicitação `TransactWriteItems` ao DAX:

- O DAX envia a solicitação para o DynamoDB.
- O DynamoDB responde ao DAX, confirmando que a transação foi concluída.
- O DAX retorna uma resposta de êxito ao solicitante.
- Em segundo plano, o DAX faz uma solicitação `TransactGetItems` para cada item na solicitação `TransactWriteItems` para armazenar o item no cache de itens. `TransactGetItems` é usada para garantir o isolamento serializável.

Se houver falha em uma gravação no DynamoDB por qualquer motivo, inclusive controle de utilização, o item não será armazenado em cache no DAX. A exceção da falha é retornada ao solicitante. Isso garante que os dados sejam gravados no cache do DAX somente se forem gravados primeiro com êxito no DynamoDB.

> **Note**
>
> Toda gravação no DAX altera o estado do cache de itens. No entanto, as gravações no cache de itens não afeta o cache de consultas. (O cache de itens e o cache de consultas do DAX têm finalidades diferentes e operam de forma independente um do outro.)

Comportamento do cache de consultas DAX

O DAX armazena em cache os resultados das solicitações `Query` e `Scan` em seu cache de consultas. No entanto, esses resultados não afetam de forma alguma o cache de itens. Quando sua aplicação emite uma solicitação `Query` ou `Scan` com o DAX, o conjunto de resultados é salvo no cache de consultas, e não no cache de itens. Você não pode "aquecer" o cache de itens executando uma operação `Scan` porque o cache de itens e o cache de consultas são entidades separadas.

Consistência de consulta-atualização-consulta

As atualizações no cache de itens, ou na tabela subjacente do DynamoDB, não invalidam nem modificam os resultados armazenados no cache de consultas.

Para ilustrar, considere o seguinte cenário. Um aplicativo está trabalhando com a tabela `DocumentRevisions`, que tem `DocId` como chave de partição e `RevisionNumber` como chave de classificação.

1. Um cliente emite uma solicitação `Query` para `DocId 101` para todos os itens cujo `RevisionNumber` é maior ou igual a `5`. O DAX armazena o conjunto de resultados no cache de consultas e retorna esse conjunto ao usuário.
2. O cliente emite uma solicitação de `PutItem` de `DocId 101` com um valor de `RevisionNumber` igual a `20`.
3. O cliente emite a mesma `Query` descrita na etapa 1 (`DocId 101` e `RevisionNumber >= 5`).

Neste cenário, o conjunto de resultados armazenado em cache para a `Query` emitida na etapa 3 será idêntico ao conjunto de resultados que foi armazenado em cache na etapa 1. Isso acontece

porque o DAX não invalida os conjuntos de resultados de `Query` ou `Scan` com base em atualizações em itens individuais. A operação `PutItem` da etapa 2 é refletida no cache de consultas do DAX apenas quando o TTL de `Query` expira.

Seu aplicativo deve considerar o valor do TTL do cache de consultas e por quanto tempo pode tolerar resultados inconsistentes entre o cache de consultas e o cache de itens.

Leituras altamente consistentes e transacionais

Para realizar uma solicitação de `GetItem`, `BatchGetItem`, `Query` ou `Scan` fortemente consistente, defina o parâmetro `ConsistentRead` como true. O DAX transfere solicitações de leitura fortemente consistente ao DynamoDB. Ao receber uma resposta do DynamoDB, o DAX retorna os resultados ao cliente, mas não armazena os resultados em cache. O DAX não pode servir leituras fortemente consistentes por conta própria, pois ele não está firmemente acoplado ao DynamoDB. Por esse motivo, todas as leituras subsequentes do DAX precisariam ser leituras finais consistentes. E todas as leituras fortemente consistentes subsequentes precisariam ser passadas para o DynamoDB.

O DAX trata solicitações `TransactGetItems` da mesma forma como trata leituras fortemente consistentes. O DAX passa todas as solicitações `TransactGetItems` para o DynamoDB. Ao receber uma resposta do DynamoDB, o DAX retorna os resultados ao cliente, mas não armazena os resultados em cache.

Armazenamento em cache negativo

O DAX oferece suporte a entradas de cache negativas, tanto no cache de itens quanto no cache de consultas. Uma entrada de cache negativa ocorre quando o DAX não consegue localizar os itens solicitados em uma tabela subjacente do DynamoDB. Em vez de gerar um erro, o DAX armazena em cache um resultado vazio e retorna esse resultado ao usuário.

Por exemplo, suponha que uma aplicação envie uma solicitação `GetItem` a um cluster do DAX e que não haja itens correspondentes no cache de itens do DAX. Isso faz com que o DAX leia o item correspondente na tabela subjacente do DynamoDB. Se o item não existir no DynamoDB, o DAX armazenará um item vazio no cache de itens e retornará esse item vazio para a aplicação. Agora, suponha que a aplicação envie outra solicitação `GetItem` para o mesmo item. O DAX encontrará o item vazio no cache de itens e o retornará imediatamente para a aplicação. Ele não consulta o DynamoDB.

Uma entrada de cache negativa permanece no cache de itens do DAX até que o TTL do item expire, a LRU seja invocada ou o item seja modificado via `PutItem`, `UpdateItem` ou `DeleteItem`.

O cache de consulta do DAX manipula resultados de cache negativos de maneira semelhante. Se uma aplicação executar uma operação `Query` ou `Scan` e o cache de consultas do DAX não contiver um resultado armazenado em cache, o DAX enviará a solicitação para o DynamoDB. Se não houver itens correspondentes no conjunto de resultados, o DAX armazenará um conjunto de resultados vazio no cache de consultas e retornará esse conjunto vazio à aplicação. Solicitações `Query` ou `Scan` subsequentes produzem o mesmo conjunto de resultados (vazio), até que o TTL desse conjunto de resultados tenha expirado.

Estratégias para gravações

O comportamento de gravação simultânea do DAX é apropriado para muitos padrões de aplicações. No entanto, existem alguns padrões de aplicativo em que um modelo de gravação simultânea pode não ser apropriado.

Para aplicações sensíveis à latência, a gravação por meio do DAX incorre em um salto de rede extra. Portanto, uma gravação no DAX é um pouco mais lenta que uma gravação diretamente no DynamoDB. Se a sua aplicação for sensível à latência de gravação, você poderá reduzir essa latência gravando diretamente no DynamoDB. Para ter mais informações, consulte Write-around (gravação direta).

Para aplicações que fazem uso intensivo de gravações (como aqueles que executam carregamento de dados em massa), talvez não seja desejável gravar todos os dados via DAX porque apenas uma porcentagem pequena deles é lida pela aplicação. Quando você grava grandes quantidades de dados via DAX, ele deve chamar seu algoritmo LRU de forma a liberar espaço no cache para que os novos itens sejam lidos. Isso diminui a eficácia do DAX como um cache de leitura.

Quando você grava um item no DAX, o estado do cache de itens é alterado para acomodar o novo item. (Por exemplo, o DAX talvez precise remover dados antigos do cache de itens para liberar espaço para o novo item.) O novo item permanece no cache de itens, sujeito ao algoritmo LRU do cache e à configuração de TTL do cache. Enquanto o item persistir no cache de itens, o DAX não lerá o item novamente no DynamoDB.

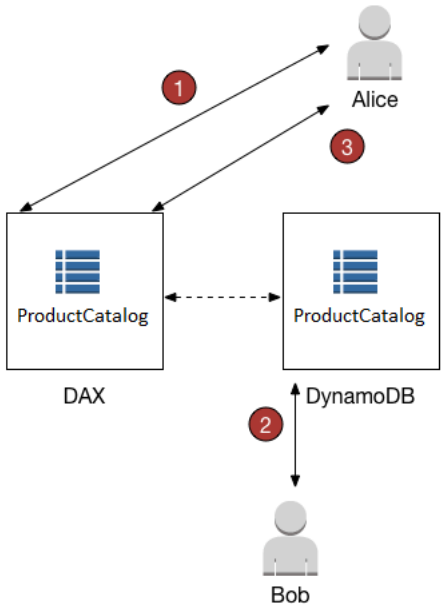
Gravação simultânea

O cache de itens do DAX implementa uma política de gravação simultânea (write-through). Para ter mais informações, consulte Como o DAX processa gravações.

Quando você grava um item, o DAX garante que o item em cache seja sincronizado com o item existente no DynamoDB. Isto é útil para aplicativos que precisam repetir a leitura de um item logo

depois de gravá-lo. No entanto, se outras aplicações gravarem diretamente em uma tabela do DynamoDB, o cache de itens do DAX não estará mais em sincronia com o DynamoDB.

Para ilustrar isso, considere dois usuários (Alice e Bob) que estão trabalhando com a tabela `ProductCatalog`. Alice acessa a tabela usando o DAX, mas Bob ignora o DAX e acessa a tabela diretamente no DynamoDB.



1. Alice atualiza um item na tabela `ProductCatalog`. O DAX encaminha a solicitação ao DynamoDB, e a atualização é bem-sucedida. Em seguida, o DAX grava o item em seu cache de itens e retorna uma resposta bem-sucedida a Alice. Desse ponto em diante, até que o item seja finalmente removido do cache, qualquer usuário que ler o item no DAX o verá com a atualização de Alice.
2. Pouco depois, Bob atualiza o mesmo item do `ProductCatalog` gravado por Alice. No entanto, Bob atualiza o item diretamente no DynamoDB. O DAX não atualiza automaticamente o cache de itens em resposta às atualizações feitas no DynamoDB. Portanto, os usuários do DAX não veem a atualização de Bob.
3. Alice lê novamente o item do DAX. O item está no cache de itens e, portanto, o DAX o retorna para Alice sem acessar a tabela do DynamoDB.

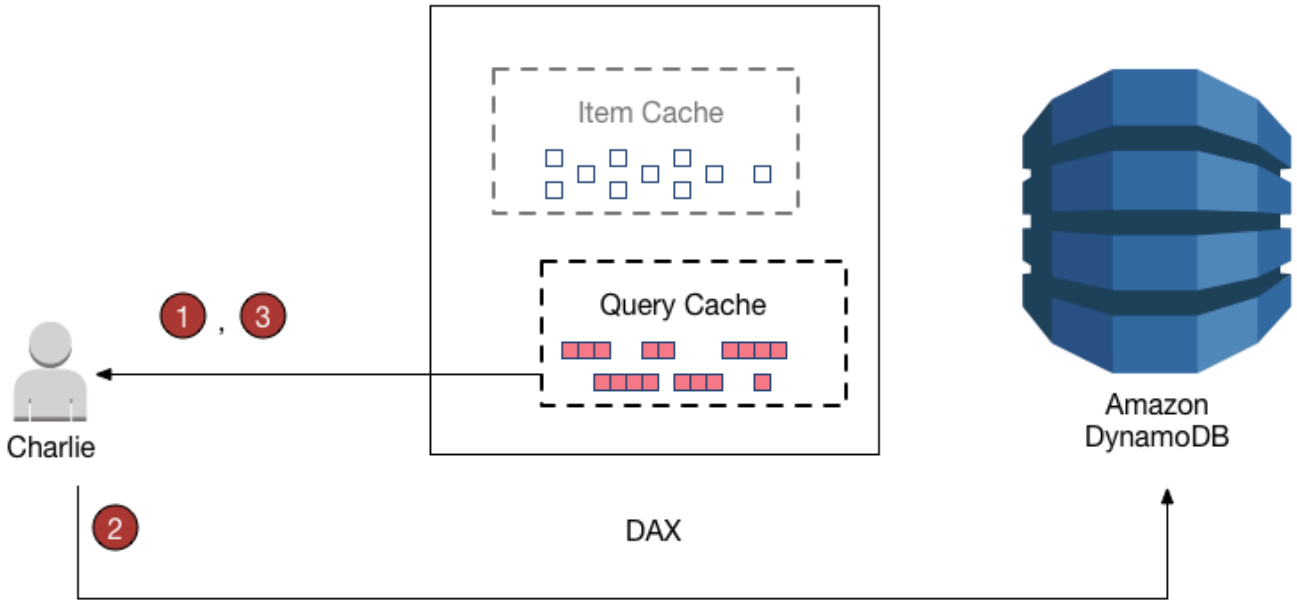
Nesse cenário, Alice e Bob veem representações diferentes do mesmo item do `ProductCatalog`. Esse será o caso até que o DAX remova o item do cache de itens ou até que outro usuário atualize o mesmo item novamente usando o DAX.

Write-around (gravação direta)

Se a sua aplicação precisa gravar grandes quantidades de dados (como um carregamento em massa de dados), talvez faça sentido ignorar o DAX e gravar os dados diretamente no DynamoDB. Essa estratégia de gravação direta (write-around) reduz a latência de gravação. No entanto, o cache de itens não permanece em sincronia com os dados no DynamoDB.

Se você optar por usar uma estratégia de gravação direta (write-around), lembre-se de que o DAX preenche o cache de itens sempre que as aplicações usam o cliente do DAX para ler os dados. Isso pode ser vantajoso em alguns casos porque garante que apenas os dados lidos com mais frequência sejam armazenados em cache (não os dados gravados com mais frequência).

Por exemplo, considere um usuário (Charlie) que deseja trabalhar com uma tabela diferente, a tabela `GameScores` usando o DAX. A chave de partição de `GameScores` é `UserId` e, portanto, todas as pontuações de Charlie teriam o mesmo `UserId`.



1. Charlie deseja recuperar todas as suas pontuações e, para isso, envia uma `Query` para o DAX. Supondo que essa consulta não foi emitida antes, o DAX a encaminha ao DynamoDB para processamento. Ele armazena os resultados no cache de consultas do DAX e retorna os

resultados a Charlie. O conjunto de resultados permanece disponível no cache de consultas até ser removido.

2. Agora, suponha que Charlie jogue uma partida de Meteor Blasters e obtenha uma pontuação elevada. Charlie envia uma solicitação de `UpdateItem` ao DynamoDB, modificando um item na tabela `GameScores`.
3. Por fim, Charlie decide executar novamente sua `Query` anterior para recuperar todos os dados de `GameScores`. Charlie não vê sua alta pontuação do jogo Meteor Blasters nos resultados. Isso ocorre porque os resultados da consulta vêm do cache de consultas, e não do cache de itens. Os dois caches são independentes um do outro e, portanto, uma alteração em um cache não afeta o outro.

O DAX não atualiza conjuntos de resultados no cache de consultas com os dados mais atuais do DynamoDB. Cada conjunto de resultados no cache de consultas é atual no momento em que a operação `Query` ou `Scan` foi executada. Portanto, os resultados da `Query` de Charlie não refletem sua operação `PutItem`. Esse será o caso até que o DAX remova o conjunto de resultados do cache de consultas.

Desenvolver com o cliente do DynamoDB Accelerator (DAX)

Para usar o DAX via uma aplicação, use o cliente do DAX para sua linguagem de programação. O cliente do DAX foi projetado para causar o mínimo de interrupção em suas aplicações existentes do Amazon DynamoDB, exigindo apenas algumas modificações simples no código.

> **Note**
>
> Clientes do DAX para várias linguagens de programação estão disponíveis no seguinte site:
>
> - http://dax-sdk.s3-website-us-west-2.amazonaws.com

Esta seção demonstra como iniciar uma instância do Amazon EC2 na sua Amazon VPC padrão, conectar-se a essa instância e executar uma aplicação de exemplo. Ela também fornece informações sobre como modificar sua aplicação existente para que ela possa usar seu cluster do DAX.

Tópicos

- Tutorial: executar uma aplicação de exemplo usando o DynamoDB Accelerator (DAX)
- Como modificar uma aplicação existente para usar o DAX

Tutorial: executar uma aplicação de exemplo usando o DynamoDB Accelerator (DAX)

Este tutorial demonstra como iniciar uma instância do Amazon EC2 na sua nuvem privada virtual (VPC) padrão, conectar-se à instância e executar uma aplicação de exemplo que usa o Amazon DynamoDB Accelerator (DAX).

> **Note**
>
> Para completar este tutorial, você deverá ter um cluster do DAX em execução na sua VPC padrão. Se você ainda não criou um cluster do DAX, consulte Criar um cluster do DAX para obter instruções.

Tópicos

- Etapa 1: iniciar uma instância do Amazon EC2
- Etapa 2: criar um usuário e uma política
- Etapa 3: configurar uma instância do Amazon EC2
- Etapa 4: executar uma aplicação de amostra

Etapa 1: iniciar uma instância do Amazon EC2

Quando seu cluster do Amazon DynamoDB Accelerator (DAX) estiver disponível, você poderá inicializar uma instância do Amazon EC2 em sua Amazon Virtual Private Cloud (Amazon VPC) padrão. Você poderá então instalar e executar o software cliente do DAX nessa instância.

Para iniciar uma instância do EC2

1. Faça login no AWS Management Console e abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. Selecione Launch Instance (Executar instância) e faça o seguinte:

   Etapa 1: Escolher uma imagem de máquina da Amazon (AMI)

   1. Na lista de AMIs, localize Amazon Linux AMI (AMI do Amazon Linux) e escolha Select (Selecionar).

Etapa 2: escolher um tipo de instância

1. Na lista de tipos de instância, escolha t2.micro.
2. Escolha Next: Configure Instance Details (Próximo: configurar detalhes da instância).

Etapa 3: configurar detalhes da instância

1. Em Network (Rede), escolha a nuvem privada virtual (VPC) padrão.
2. Escolha Next: Add Storage (Próximo: adicionar armazenamento).

Etapa 4: adicionar armazenamento

1. Ignore esta etapa escolhendo Next: Add tags (Próximo: adicionar tags).

Etapa 5: adicionar tags

1. Ignore esta etapa escolhendo Próximo: configurar o grupo de segurança.

Etapa 6: configurar o grupo de segurança

1. Escolha Select an existing security group (Selecionar um grupo de segurança existente).
2. Na lista de grupos de segurança, escolha default (padrão). Este é o grupo de segurança padrão para sua VPC.
3. Escolha Próximo: revisar e executar.

Etapa 7: revisar a execução da instância

1. Escolha Launch (Executar).

3. Na janela Select an existing key pair or create a new key pair (Selecionar um par de chaves existente ou criar um novo par de chaves), siga um destes procedimentos:

   - Se você não tiver um par de chaves do Amazon EC2, escolha Create a new key pair (Criar um novo par de chaves) e siga as instruções. É solicitado que você faça download de um arquivo de chave privada (arquivo `.pem`). Você precisará deste arquivo mais tarde quando fizer login na instância do Amazon EC2.

- Se você já tiver um par de chaves existente do Amazon EC2, vá para Select a key pair (Selecionar um par de chaves) e escolha o seu par de chaves na lista. Você já deverá ter o arquivo de chave privada (arquivo `.pem`) disponível para fazer login na instância do Amazon EC2.

4. Depois de configurar seu par de chaves, escolha Launch instances (Executar instâncias).
5. No painel de navegação do console, escolha EC2 Dashboard (Painel do EC2) e, depois, escolha a instância que você executou. No painel inferior, na guia Description (Descrição), localize o Public DNS (DNS público) da instância, por exemplo: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`. Anote esse nome DNS público, pois você precisará dele para Etapa 3: configurar uma instância do Amazon EC2.

> **Note**
>
> Serão necessários alguns minutos para que a sua instância do Amazon EC2 se torne disponível. Enquanto isso, prossiga para Etapa 2: criar um usuário e uma política e siga as instruções contidas ali.

Etapa 2: criar um usuário e uma política

Nesta etapa, você criará um usuário com uma política que concede acesso ao cluster do Amazon DynamoDB Accelerator (DAX) e ao DynamoDB usando o AWS Identity and Access Management. Assim, você poderá executar aplicações que interagem com seu cluster do DAX.

Inscrever-se para uma Conta da AWS

Se você ainda não tem uma Conta da AWS, siga as etapas abaixo para criar uma.

Como cadastrar uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica ou uma mensagem de texto e inserir um código de verificação pelo teclado do telefone.

   Quando você se inscreve para uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática

recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

A AWS envia um e-mail de confirmação depois que o processo de inscrição é concluído. A qualquer momento, é possível exibir as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário-raiz em tarefas cotidianas.

Proteger o Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao escolher a opção Usuário-raiz e inserir o endereço de e-mail da Conta da AWS. Na próxima página, insira a senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Fazer login como usuário-raiz no Guia do usuário do Início de Sessão da AWS.
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para sua Conta da AWS de usuário-raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilita o Centro de Identidade do IAM.

   Para obter instruções, consulte Habilitar o AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.
2. No Centro de Identidade do IAM, conceda o acesso administrativo a um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como a fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  Para obter ajuda para fazer login usando um usuário do Centro de Identidade do IAM, consulte Fazer login no portal de acesso da AWS, no Guia do usuário do Início de Sessão da AWS.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Criar um conjunto de permissões no Guia do usuário do AWS IAM Identity Center.
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Adicionar grupos no Guia do usuário do AWS IAM Identity Center.

Para conceder acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos no AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Criação de um conjunto de permissões no Guia do usuário do AWS IAM Identity Center.
- Usuários gerenciados no IAM com provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criando um perfil para um provedor de identidades de terceiros (federação) no Guia do Usuário do IAM.
- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Criação de um perfil para um usuário do IAM no Guia do usuário do IAM.
  - (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Para usar o editor de políticas JSON para criar uma política

1. Faça login no AWS Management Console e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. No painel de navegação à esquerda, escolha Políticas.

   Se essa for a primeira vez que você escolhe Políticas, a página Bem-vindo às políticas gerenciadas será exibida. Escolha Começar.
3. Na parte superior da página, escolha Criar política.
4. Na seção Editor de políticas, escolha a opção JSON.
5. Insira ou cole um documento de política JSON. Para obter detalhes sobre a linguagem da política do IAM, consulte a referência de política JSON do IAM.
6. Resolva os avisos de segurança, erros ou avisos gerais gerados durante a validação de política e depois escolha Próximo.

   > **Note**
   >
   > Você pode alternar entre as opções de editor Visual e JSON a qualquer momento. Porém, se você fizer alterações ou escolher Próximo no editor Visual, o IAM poderá reestruturar a política a fim de otimizá-la para o editor visual. Para obter mais informações, consulte Restruturação de política no Guia do usuário do IAM.
7. (Opcional) Ao criar ou editar uma política no AWS Management Console, você pode gerar um modelo de política JSON ou YAML que pode ser usado em modelos do AWS CloudFormation.

   Para isso, no Editor de políticas, escolha Ações e depois escolha Gerar modelo do CloudFormation. Para saber mais sobre o AWS CloudFormation, consulte Referência de tipos de recurso do AWS Identity and Access Management no Guia do usuário do AWS CloudFormation.
8. Quando terminar de adicionar as permissões à política, escolha Avançar.
9. Na página Revisar e criar, insira um Nome de política e uma Descrição (opcional) para a política que você está criando. Revise Permissões definidas nessa política para ver as permissões que são concedidas pela política.
10. (Opcional) Adicione metadados à política associando tags como pares de chave-valor. Para obter mais informações sobre o uso de tags no IAM, consulte Tags para recursos do AWS Identity and Access Management no Guia do usuário do IAM.

11. Escolha Criar política para salvar sua nova política.

Documento de política: copie e cole o documento a seguir para criar a política JSON.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Etapa 3: configurar uma instância do Amazon EC2

Quando sua instância do Amazon EC2 estiver disponível, você poderá fazer login na instância e prepará-la para uso.

> **Note**
>
> As etapas a seguir assumem que você está se conectando à sua instância do Amazon EC2 de um computador que executa o Linux. Para conferir outras maneiras de se conectar, consulte Conecte-se à sua instância do Linux no Guia do usuário do Amazon EC2.

Como configurar a instância do EC2

1. Abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. Use o comando `ssh` para fazer login em sua instância do Amazon EC2, conforme mostrado no exemplo a seguir.

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   Você precisará especificar seu arquivo de chave privada (arquivo `.pem`) e o nome DNS público da instância. (Consulte Etapa 1: iniciar uma instância do Amazon EC2.)

   O ID de login é `ec2-user`. Nenhuma senha é necessária.
3. Depois de fazer login em sua instância do EC2, configure suas credenciais da AWS, conforme mostrado abaixo. Insira o ID da chave de acesso da AWS e a chave secreta (de Etapa 2: criar um usuário e uma política) e defina o nome da região padrão como a sua região atual. (No exemplo a seguir, o nome de região padrão é `us-west-2`.)

   ```
   aws configure

   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]:
   ```

Após iniciar e configurar sua instância do Amazon EC2, você poderá testar a funcionalidade do DAX usando uma das aplicações de exemplo disponíveis. Para obter mais informações, consulte Etapa 4: executar uma aplicação de amostra.

Etapa 4: executar uma aplicação de amostra

Para ajudar a testar a funcionalidade do Amazon DynamoDB Accelerator (DAX), você pode executar uma das aplicações de exemplo disponíveis em sua instância do Amazon EC2.

Tópicos

- Node.js e DAX
- DAX SDK for Go
- Java e DAX
- .NET e DAX
- Python e DAX

Node.js e DAX

Configuração do cliente padrão para Node.js

Ao configurar o cliente SDK JavaScript do DAX, você pode personalizar vários parâmetros para otimizar a performance, o tratamento da conexão e a resiliência a erros. A tabela a seguir descreve as configurações padrão que controlam como o cliente interage com o cluster DAX, como valores de tempo limite, mecanismos de repetição, gerenciamento de credenciais e opções de monitoramento de integridade. Para ter mais informações, consulte DynamoDBClient Operations.

Padrões do cliente SDK JS do DAX

Parameter	Tipo	Descrição
`region` optional	`string`	A Região da AWS a ser usada para o cliente DAX (exemplo: “us-east-1”). É um parâmetro obrigatório caso não seja fornecido por meio da variável de ambiente.
`endpoint` obrigatório	`string`	O endpoint do cluster ao qual o SDK se conecta. Exemplos:

Parameter	Tipo	Descrição
		Não criptografado: dax-cluster-name.region.amazonaws.com Criptografado: daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`requestTimeout` padrão 6.000 ms	`number`	Isso define o tempo máximo que o cliente aguardará por uma resposta do DAX.
`writeRetries` padrão 1	`number`	O número de tentativas para solicitações de gravação que falharam.
`readRetries` padrão 1	`number`	O número de tentativas para solicitações de leitura que falharam.
`maxRetries` padrão 1	`number`	A quantidade máxima de tentativas após solicitações com falha. Se readRetries/writeRetries estiverem definidos, a configuração em readRetries e writeRetries terá prioridade sobre maxRetries.
`connectTimeout` padrão 10.000 ms	`number`	O tempo limite (em milissegundos) para estabelecer conexão com qualquer um dos nós do cluster.

Parameter	Tipo	Descrição
`maxConcurrentConnections` padrão 100	`number`	Limita o número total de conexões simultâneas que uma instância cliente pode criar por nó em um cluster do DAX.
`maxRetryDelay` padrão 7.000 ms	`number`	Quando o servidor DAX indicar que a recuperação é necessária definindo o sinalizador `waitForRecoveryBeforeRetrying` como verdadeiro, o cliente fará uma pausa antes de tentar novamente. Durante esses períodos de recuperação, o parâmetro `maxRetryDelay` determina o tempo máximo de espera entre as tentativas. Essa configuração específica de recuperação só se aplica quando o servidor DAX está no modo de recuperação. Em todos os outros cenários, o comportamento de tentativa segue um dos dois padrões: um atraso exponencial com base na contagem de tentativas (controlado pelos parâmetros `writeRetries`, `readRetries` ou `maxRetries`) ou uma nova tentativa imediata, dependendo do tipo de exceção.

Parameter	Tipo	Descrição
`credentials` optional	`AwsCredentialIdentity` | `AwsCredentialIdentityProvider`	As credenciais da AWS a serem usadas para autenticar solicitações. Isso pode ser fornecido como um AwsCredentialIdentity ou um AwsCredentialIdentityProvider. Se não for fornecido, o SDK da AWS usará automaticamente a cadeia de provedores de credenciais padrão. Exemplo: `{ accessKeyId: 'AKIA...', secretAccessKey: '...', sessionToken: '...' }` * @default usa a cadeia de provedores de credenciais padrão da AWS.
`healthCheckInterval` padrão 5.000 ms	`number`	O intervalo (em milissegundos) entre as verificações de integridade do cluster. Um intervalo menor fará verificações com maior frequência.
`healthCheckTimeout` padrão 1.000 ms	`number`	O tempo limite (em milissegundos) para a conclusão da verificação de integridade.

Parameter	Tipo	Descrição
`skipHostnameVerification` padrão falso	`boolean`	Ignore a verificação do nome do host das conexões TLS. Isso não tem impacto em clusters não criptografados. O padrão é realizar a verificação do nome do host. Se definida como True, a verificação será ignorada. Você precisa entender a implicação de desativá-la, que é a impossibilidade de autenticar o cluster ao qual você está se conectando.
`unhealthyConsecutiveErrorCount` padrão 5	`number`	Define o número de erros consecutivos necessários para sinalizar que o nó não está íntegro no intervalo de verificação de integridade.
`clusterUpdateInterval` padrão 4.000 ms	`number`	Exibe o intervalo entre as sondagens dos membros do cluster para alterações de associação.
`clusterUpdateThreshold` padrão 125	`number`	Exibe o limite abaixo do qual o cluster não será sondado para alterações de associação.
`credentailProvider` opcional | padrão nulo	`AwsCredentialIdentityProvider`	Provedor definido pelo usuário para credenciais da AWS usadas para autenticar solicitações ao DAX.

Configuração de paginação para DaxDocument

Name	Tipo	Detalhes
`client`	DaxDocument	Instância do tipo DaxDocument.
`pageSize`	número	Determina o número de itens por página.
`startingToken` Opcional	alumínio	A LastEvaluatedKey da resposta anterior pode ser usada para solicitações subsequentes.

Quanto ao uso da paginação, consulte the section called “TryDax.js”.

Migrar para o DAX Node.js SDK V3

Este guia de migração ajudará você a fazer a transição de aplicações DAX Node.js existentes. O novo SDK requer o Node.js 18 ou posterior e introduz várias alterações importantes na forma como você estruturará o código do DynamoDB Accelerator. Este guia explicará as principais diferenças, incluindo alterações de sintaxe, novos métodos de importação e padrões de programação assíncrona atualizados.

Uso do DAX Node.js V2

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var client = new AWS.DynamoDB.DocumentClient();

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
```

```
    endpoints: [process.argv[2]],
    region: region,
  });
  client = new AWS.DynamoDB.DocumentClient({ service: dax });
}

// Make Get Call using Dax
var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
client.get(params, function (err, data) {
    if (err) {
        console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
    } else {
        console.log(data);
    }
});
```

Uso do DAX Node.js V3

Para usar o DAX Node.js V3, a versão 18 ou posterior do Node é a versão preferencial. Para mudar para o Node 18, use o seguinte:

```
// Import AWS DAX V3
import { DaxDocument } from '@amazon-dax-sdk/lib-dax';

// Import AWS SDK V3 DynamoDBDocument ~ DocumentClient in V2
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';
import { DynamoDBClient } from '@aws-sdk/client-dyanmodb';

// Create DynamoDBDocument
var client = DynamoDBDocument.from(new DynamoDB({region: 'us-west-2'});

// Override DynamoDBDocument Client with DaxDocument
if (process.argv.length > 2) {
  client = new DaxDocument({
    endpoints: [process.argv[2]],
    region: 'us-west-2',
```

```
  });
}

var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
// Dax Shifted it's API Calls to await/promise
try {
  const results = await client.get(params);
  console.log(results);
} catch (err) {
  console.error(err)
}
```

O DAX SDK para Node.js v3.x é compatível com o AWS SDK para Node.js v3.x. O DAX SDK para Node.js v3.x aceita o uso de clientes agregados. Observe que o DAX não aceita a criação de clientes básicos. Para ter mais detalhes sobre os recursos não aceitos, consulte the section called “Recursos que não estão em paridade com o AWS SDK para V3”.

Siga estas etapas para executar a aplicação de exemplo em Node.js na instância do Amazon EC2.

Para executar o exemplo em Node.js para DAX

1. Configure o Node.js na instância do Amazon EC2 da seguinte forma:

   a. Instale o gerenciador de versão de nó (nvm).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. Use o nvm para instalar o Node.js.

   ```
   nvm install 18
   ```

   c. Usar nvm para utilizar o Node 18

   ```
   nvm use 18
   ```

   d. Verifique se o Node.js está instalado e funcionando corretamente.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

Isso deve exibir a seguinte mensagem.

```
Running Node.js v18.x.x
```

2. Instale o cliente Node.js do DaxDocument usando o gerenciador de pacotes de nó (npm).

```
npm install @amazon-dax-sdk/lib-dax
```

Código de exemplo do TryDax

Se quiser avaliar os benefícios de performance do DynamoDB Accelerator (DAX), siga estas etapas para executar um teste de amostra que compara os tempos de operação de leitura entre o DynamoDB padrão e um cluster do DAX.

1. Depois de configurar seu espaço de trabalho e instalar o `lib-dax` como uma dependência, copie the section called “TryDax.js” no projeto.
2. Execute o programa no cluster do DAX. Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

   - Usando o console do DynamoDB: escolha seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Usando a AWS CLI: insira o comando a seguir.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     O endpoint do cluster é mostrado na saída, como no exemplo a seguir.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

3. Agora execute o programa especificando o endpoint do cluster como um parâmetro de linha de comandos.

```
node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Você deve ver uma saída semelhante a:

```
Attempting to create table; please wait...
Successfully created table. Table status: ACTIVE
Writing data to the table...
Writing 20 items for partition key:  1
Writing 20 items for partition key:  2
Writing 20 items for partition key:  3
...
Running GetItem Test
        Total time: 153555.10 µs - Avg time: 383.89 µs
        Total time: 44679.96 µs - Avg time: 111.70 µs
        Total time: 36885.86 µs - Avg time: 92.21 µs
        Total time: 32467.25 µs - Avg time: 81.17 µs
        Total time: 32202.60 µs - Avg time: 80.51 µs
Running Query Test
        Total time: 14869.25 µs - Avg time: 2973.85 µs
        Total time: 3036.31 µs - Avg time: 607.26 µs
        Total time: 2468.92 µs - Avg time: 493.78 µs
        Total time: 2062.53 µs - Avg time: 412.51 µs
        Total time: 2178.22 µs - Avg time: 435.64 µs
Running Scan Test
        Total time: 2395.88 µs - Avg time: 479.18 µs
        Total time: 2207.16 µs - Avg time: 441.43 µs
        Total time: 2443.14 µs - Avg time: 488.63 µs
        Total time: 2038.24 µs - Avg time: 407.65 µs
        Total time: 1972.17 µs - Avg time: 394.43 µs
Running Pagination Test
Scan Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
```

```
...
Query Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Attempting to delete table; please wait...
Successfully deleted table.
```

Anote as informações de tempo. O número de microssegundos necessários para os testes `GetItem`, `Query` e `Scan`.

4. Nesse caso, você executou os programas no cluster do DAX. Agora você executará o programa novamente, desta vez no DynamoDB.
5. Execute o programa novamente, mas, desta vez, sem o URL do endpoint do cluster como um parâmetro de linha de comandos.

```
node TryDax.js
```

Observe o restante da saída e anote as informações de tempo. Os tempos decorridos para `GetItem`, `Query` e `Scan` devem ser muito mais baixos com o DAX em comparação ao DynamoDB.

Recursos que não estão em paridade com o AWS SDK para V3

- Clientes básicos: o Dax Node.js V3 não aceita clientes básicos.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
const regularParams = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
```

```
};
// The DynamoDB client supports the send operation.
const dynamoResult = await dynamoDBClient.send(new GetCommand(regularParams));

// However, the DaxDocument client does not support the send operation.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

const params = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};

// This will throw an error - send operation is not supported for DAX. Please refer
 to documentation.
const result = await daxClient.send(new GetCommand(params));
console.log(result);
```

- Middleware Stack: o Dax Node.js V3 não comporta o uso de funções de Middleware.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
// The DynamoDB client supports the middlewareStack.
dynamoDBClient.middlewareStack.add(
  (next, context) =>> async (args) => {
    console.log("Before operation:", args);
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);

// However, the DaxDocument client does not support the middlewareStack.
const daxClient = new DaxDocument({
```

```
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

// This will throw an error - custom middleware and middlewareStacks are not
 supported for DAX. Please refer to documentation.
daxClient.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("Before operation:", args);
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);
```

TryDax.js

```
import { DynamoDB, waitUntilTableExists, waitUntilTableNotExists } from "@aws-sdk/
client-dynamodb";
import { DaxDocument, daxPaginateScan, daxPaginateQuery } from "@amazon-dax-sdk/lib-
dax";
import { DynamoDBDocument, paginateQuery, paginateScan } from "@aws-sdk/lib-dynamodb";

const region = "us-east-1";
const tableName = "TryDaxTable";

// Determine the client (DynamoDB or DAX)
let client = DynamoDBDocument.from(new DynamoDB({ region }));
if (process.argv.length > 2) {
  client = new DaxDocument({ region, endpoint: process.argv[2] });
}

// Function to create table
async function createTable() {
  const dynamodb = new DynamoDB({ region });
  const params = {
    TableName: tableName,
```

```
    KeySchema: [
      { AttributeName: "pk", KeyType: "HASH" },
      { AttributeName: "sk", KeyType: "RANGE" },
    ],
    AttributeDefinitions: [
      { AttributeName: "pk", AttributeType: "N" },
      { AttributeName: "sk", AttributeType: "N" },
    ],
    ProvisionedThroughput: { ReadCapacityUnits: 25, WriteCapacityUnits: 25 },
  };

  try {
    console.log("Attempting to create table; please wait...");
    await dynamodb.createTable(params);
    await waitUntilTableExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully created table. Table status: ACTIVE");
  } catch (err) {
    console.error("Error in table creation:", err);
  }
}

// Function to insert data
async function writeData() {
  console.log("Writing data to the table...");
  const someData = "X".repeat(10);
  for (let ipk = 1; ipk <= 20; ipk++) {
    console.log("Writing 20 items for partition key: ", ipk)
    for (let isk = 1; isk <= 20; isk++) {
      try {
        await client.put({ TableName: tableName, Item: { pk: ipk, sk: isk,
 someData } });
      } catch (err) {
        console.error("Error inserting data:", err);
      }
    }
  }
}

// Function to test GetItem
async function getItemTest() {
  console.log("Running GetItem Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
```

```
    const promises = [];
    for (let ipk = 1; ipk <= 20; ipk++) {
      for (let isk = 1; isk <= 20; isk++) {
        promises.push(client.get({ TableName: tableName, Key: { pk: ipk, sk: isk } }));
      }
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 400).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Query
async function queryTest() {
  console.log("Running Query Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.query(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Scan
async function scanTest() {
  console.log("Running Scan Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
```

```
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        FilterExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.scan(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Pagination
async function paginationTest() {
  console.log("Running Pagination Test");
  console.log("Scan Pagination");
  const scanParams = { TableName: tableName };
  const paginator = process.argv.length > 2 ? daxPaginateScan : paginateScan;
  for await (const page of paginator({ client, pageSize: 3 }, scanParams)) {
    console.log(page.Items);
  }

  console.log("Query Pagination");
  const queryParams = {
    TableName: tableName,
    KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
    ExpressionAttributeValues: { ":pkval": 1, ":skval1": 1, ":skval2": 10 },
  };
  const queryPaginator = process.argv.length > 2 ? daxPaginateQuery : paginateQuery;
  for await (const page of queryPaginator({ client, pageSize: 3 }, queryParams)) {
    console.log(page.Items);
  }
}

// Function to delete the table
async function deleteTable() {
  const dynamodb = new DynamoDB({ region });
  console.log("Attempting to delete table; please wait...")
  try {
```

```
    await dynamodb.deleteTable({ TableName: tableName });
    await waitUntilTableNotExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully deleted table.");
  } catch (err) {
    console.error("Error deleting table:", err);
  }
}

// Execute functions sequentially
(async function () {
  await createTable();
  await writeData();
  await getItemTest();
  await queryTest();
  await scanTest();
  await paginationTest();
  await deleteTable();
})();
```

DAX SDK for Go

Siga este procedimento para executar o exemplo do SDK for Go para Amazon DynamoDB Accelerator (DAX) em sua instância do Amazon EC2.

Para executar o exemplo do SDK for Go para DAX

1. Configure o SDK for Go em sua instância do Amazon EC2:

   a. Instale a linguagem de programação Go (`Golang`).

   ```
   sudo yum install -y golang
   ```

   b. Verifique se o Golang está instalado e funcionando corretamente.

   ```
   go version
   ```

   Uma mensagem semelhante a esta deve ser mostrada.

   ```
   go version go1.23.4 linux/amd64
   ```

2. Instale o aplicativo de exemplo do Golang.

```
go get github.com/aws-samples/sample-aws-dax-go-v2
```

3. Execute os seguintes programas do Golang. O primeiro programa cria uma tabela do DynamoDB chamada `TryDaxGoTable`. O segundo programa grava dados na tabela.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command put-item
```

4. Execute os seguintes programas do Golang.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command scan
```

   Anote as informações de tempo: o número de milissegundos necessários para os testes `GetItem`, `Query` e `Scan`.

5. Na etapa anterior, você executou os programas no endpoint do DynamoDB. Agora, execute os programas novamente, mas, desta vez, as operações `GetItem`, `Query` e `Scan` são processadas pelo cluster do DAX.

   Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

   - Usando o console do DynamoDB: escolha seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Usando a AWS CLI: insira o comando a seguir.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

O endpoint do cluster é mostrado na saída, como no exemplo a seguir.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Agora, execute os programas novamente, mas, desta vez, especifique o endpoint do cluster como um parâmetro de linha de comando.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-batch-get -endpoint my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com:8111
```

Observe o restante da saída e anote as informações de tempo. Os tempos decorridos para `GetItem`, `Query` e `Scan` devem ser significativamente mais baixos com o DAX do que com o DynamoDB.

6. Execute o seguinte programa Golang para excluir a `TryDaxGoTable`.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command delete-table
```

Recursos que não estão em paridade com o AWS SDK para Go V2

Pilha de middleware: o DAX Go V2 não aceita o uso de pilhas de middleware por meio de APIoptions. Para ter mais informações, consulte Customizing the AWS SDK para Go v2 Client Requests with Middleware.

Exemplo:

```
// Custom middleware implementation
type customSerializeMiddleware struct{}
// ID returns the identifier for the middleware
func (m *customSerializeMiddleware) ID() string {
    return "CustomMiddleware"
}
// HandleSerialize implements the serialize middleware handler
func (m *customSerializeMiddleware) HandleSerialize(
    ctx context.Context,
    in middleware.SerializeInput,
    next middleware.SerializeHandler,
) (
    out middleware.SerializeOutput,
    metadata middleware.Metadata,
    err error,
) {
    // Add your custom logic here before the request is serialized
    fmt.Printf("Executing custom middleware for request: %v\n", in)
    // Call the next handler in the middleware chain
    return next.HandleSerialize(ctx, in)
}

func executeGetItem(ctx context.Context) error {
    client, err := initItemClient(ctx)
```

```
    if err != nil {
        os.Stderr.WriteString(fmt.Sprintf("failed to initialize client: %v\n", err))
        return err
    }

    st := time.Now()
    for c := 0; c < iterations; c++ {
        for i := 0; i < pkMax; i++ {
            for j := 0; j < skMax; j++ {
                // Create key using attributevalue.Marshal for type safety
                pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
                if err != nil {
                    return fmt.Errorf("error marshaling pk: %v", err)
                }
                sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
                if err != nil {
                    return fmt.Errorf("error marshaling sk: %v", err)
                }
                key := map[string]types.AttributeValue{
                    "pk": pk,
                    "sk": sk,
                }
                in := &dynamodb.GetItemInput{
                    TableName: aws.String(table),
                    Key:       key,
                }

                // Custom middleware option
                customMiddleware := func(o *dynamodb.Options) {
                    o.APIOptions = append(o.APIOptions, func(stack *middleware.Stack)
error {
                        // Add custom middleware to the stack
                        return stack.Serialize.Add(&customSerializeMiddleware{},
middleware.After)
                    })
                }

                // Apply options to the GetItem call
                out, err := client.GetItem(ctx, in, customMiddleware)
                if err != nil {
                    return err
                }
                writeVerbose(out)
            }
```

```
        }
    }
    d := time.Since(st)
    os.Stdout.WriteString(fmt.Sprintf("Total Time: %v, Avg Time: %v\n", d, d/
iterations))
    return nil
}
```

Saída:

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Configuração de cliente padrão para Go

Este guia explicará as opções de configuração que permitem ajustar a performance, o gerenciamento de conexões e o comportamento de registro em log do cliente DAX. Ao entender as configurações padrão e como personalizá-las, você pode otimizar a interação da aplicação Go com o DAX.

Nesta seção

- Padrões do cliente DAX Go SDK
- Criação de cliente

Padrões do cliente DAX Go SDK

Parameter	Tipo	Descrição
`Region` obrigatório	`string`	A Região da AWS a ser usada para o cliente DAX (exemplo: “us-east-1”). É um parâmetro obrigatório caso não seja fornecido por meio do ambiente.
`HostPorts` obrigatório	`[] string`	Lista de endpoints do cluster DAX aos quais o SDK se conecta.

Parameter	Tipo	Descrição
		Por exemplo: Não criptografado: dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com Criptografado: daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`MaxPendingConnectionsPerHost` padrão 10	`number`	O número de tentativas de conexão simultâneas. (As conexões podem estar no processo de ser estabelecidas simultaneamente.)
`ClusterUpdateThreshold` padrão 125 * time.Millisecond	`time.Duration`	O tempo mínimo que deve decorrer entre as atualizações do cluster.
`ClusterUpdateInterval` padrão 4 * time.Second	`time.Duration`	O intervalo no qual o cliente atualizará automaticamente as informações do cluster DAX.
`IdleConnectionsReapDelay` padrão 30 * time.Second	`time.Duration`	O intervalo no qual o cliente fechará conexões ociosas no cliente DAX.
`ClientHealthCheckInterval` padrão 5 * time.Second	`time.Duration`	O intervalo no qual o cliente realizará verificações de integridade nos endpoints do cluster DAX.

Parameter	Tipo	Descrição
`Credentials` padrão	`aws.CredentialsProvider`	As credenciais da AWS usadas pelo cliente DAX para autenticar solicitações no serviço DAX. Consulte Credentials and Credential Providers.
`DialContext` padrão	`func`	Uma função personalizada usada pelo cliente DAX para estabelecer conexões com o cluster DAX.
`SkipHostnameVerification` padrão falso	bool	Ignore a verificação do nome do host das conexões TLS. Essa configuração afeta somente clusters criptografados. Quando definida como True, ela desativa a verificação do nome do host. Quando você desabilita a verificação, não consegue autenticar a identidade do cluster ao qual está se conectando, o que gera riscos de segurança. Por padrão, a verificação do nome do host fica habilitada.
`RouteManagerEnabled` padrão falso	`bool`	Esse sinalizador é usado para remover rotas que enfrentam erros de rede.

Parameter	Tipo	Descrição
`RequestTimeout` padrão 60 * time.Second	`time.Duration`	Isso define o tempo máximo que o cliente aguardará por uma resposta do DAX. Prioridade: tempo limite do contexto (se definido) > `RequestTimmeout` (se definido) > `RequestTimeout` padrão de 60 s.
`WriteRetries` padrão 2	`number`	O número de tentativas para solicitações de gravação que falharam.
`ReadRetries` padrão 2	`number`	O número de tentativas para solicitações de leitura que falharam.
`RetryDelay` padrão 0	`time.Duration`	O atraso para erros não limitados (em segundos) nas tentativas de repetição quando uma solicitação falha.
`Logger` optional	`logging.Logger`	Logger é uma interface para registrar em log entradas em determinadas classificações.

Parameter	Tipo	Descrição
`LogLevel` padrão utils.LogOff	`number`	Esse nível de log é definido somente para DAX. Ele pode ser importado usando github.com/aws/aws-dax-go-v2/tree/main/dax/utils. `const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithRequestRetries LogLevelType = 2 )`

> **Note**
>
> Em `time.Duration`, a unidade padrão é nanossegundo. Se não especificarmos nenhuma unidade para nenhum parâmetro, ele considerará isso como nanossegundos: `daxCfg.ClusterUpdateInterval = 10` significa 10 nanossegundos. (`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond` significa 10 milissegundos).

Criação de cliente

Para criar um cliente DAX:

- Crie a configuração do DAX e, depois, crie o cliente DAX usando a configuração do DAX. Assim, você poderá substituir uma configuração do DAX, se necessário.

```
import (
"github.com/aws/aws-dax-go-v2/dax/utils"
"github.com/aws/aws-dax-go-v2/dax"
)
```

```
// Non - Encrypted : 'dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.
// Encrypted : daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.

config := dax.DefaultConfig()
config.HostPorts = []string{endpoint}
config.Region = region
config.LogLevel = utils.LogDebug
daxClient, err := dax.New(config)
```

Migrar para o DAX Go SDK V2

Este guia de migração ajudará você a fazer a transição de aplicações DAX Go existentes.

Uso do V1 DAX Go SDK

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-dax-go/dax"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
)

func main() {
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create session
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String(region),
    })
    if err != nil {
        fmt.Printf("Failed to create session: %v\n", err)
        os.Exit(1)
    }

    // Configure DAX client
    cfg := dax.DefaultConfig()
    cfg.HostPorts = []string{endpoint}
```

```
    cfg.Region = region

    // Create DAX client
    daxClient, err := dax.New(cfg)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]*dynamodb.AttributeValue{
            "pk": {
                N: aws.String("1"),
            },
            "sk": {
                N: aws.String("1"),
            },
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Uso do V2 DAX Go SDK

```
package main

import (
    "context"
    "fmt"
    "os"
```

```
    "github.com/aws/aws-dax-go-v2/dax"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/aws"
)

func main() {
    ctx := context.Background()
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create DAX config
    config := dax.DefaultConfig()
    // Specify Endpoint and Region
    config.HostPorts = []string{endpoint}
    config.Region = region
    // Enabling logging
    config.LogLevel = utils.LogDebug
    // Create DAX client
    daxClient, err := dax.New(config)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create key using attributevalue.Marshal for type safety
    pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
    if err != nil {
        return fmt.Errorf("error marshaling pk: %v", err)
    }
    sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
    if err != nil {
        return fmt.Errorf("error marshaling sk: %v", err)
    }

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]types.AttributeValue{
             "pk": pk,
             "sk": sk,
```

```
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(ctx, input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Para ter mais detalhes sobre o uso de API, consulte Amostras da AWS.

Java e DAX

O DAX SDK para Java 2.x é compatível com o AWSSDK para Java 2.x. Ele foi criado com base no Java 8+ e inclui suporte a E/S sem bloqueio. Para ter informações sobre como usar o DAX com o AWS SDK para Java 1.x, consulte Uso do DAX com o AWS SDK for Java 1.x.

Como usar o cliente como uma dependência do Maven

Siga estas etapas para usar o cliente do SDK do DAX para Java em seu aplicativo como uma dependência.

1. Faça download do Apache Maven e instale-o. Para obter mais informações, consulte Download do Apache Maven e Instalação do Apache Maven.
2. Adicione a dependência do cliente Maven ao arquivo Project Object Model (POM) da aplicação. Neste exemplo, substitua *x.x.x* pelo número da versão real do cliente.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

Código de exemplo do TryDax

Depois de configurar seu espaço de trabalho e adicionar o DAX SDK como uma dependência, copieTryDax.java para seu projeto.

Execute o código usando este comando.

```
java -cp classpath TryDax
```

Você deve ver saída semelhante ao seguinte:

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
...

Running GetItem and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1-100 and sort keys 1-10
  Total time: 4390.240 ms - Avg time: 4.390 ms
  Total time: 3097.089 ms - Avg time: 3.097 ms
  Total time: 3273.463 ms - Avg time: 3.273 ms
  Total time: 3353.739 ms - Avg time: 3.354 ms
  Total time: 3533.314 ms - Avg time: 3.533 ms
Query test - partition key 1-100 and sort keys between 2 and 9
  Total time: 475.868 ms - Avg time: 4.759 ms
  Total time: 423.333 ms - Avg time: 4.233 ms
  Total time: 460.271 ms - Avg time: 4.603 ms
  Total time: 397.859 ms - Avg time: 3.979 ms
  Total time: 466.644 ms - Avg time: 4.666 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Anote as informações de tempo — o número de milissegundos necessários para os testes `GetItem` e `Query`. Neste caso, você executou o programa no endpoint do DynamoDB. Agora você executará o programa novamente, desta vez em seu cluster do DAX.

Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

- Usando o console do DynamoDB, selecione seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- Usando a AWS CLI, insira o comando seguir.

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  A porta, o endereço e o URL do endpoint do cluster aparecem na saída, como no exemplo a seguir.

  ```
  {
      "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
      "Port": 8111,
      "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
  }
  ```

Agora execute o programa novamente, mas, desta vez, especifique o URL do endpoint do cluster como um parâmetro de linha de comando.

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Observe o restante da saída e anote as informações de tempo. Os tempos decorridos para `GetItem` e `Query` devem ser significativamente mais baixos com o DAX do que com o DynamoDB.

Métricas do SDK

Com o DAX SDK para Java 2.x, você pode coletar métricas sobre os clientes de serviço em sua aplicação e analisar a saída no Amazon CloudWatch. Consulte Habilitar métricas do SDK para obter mais informações.

> **Note**
>
> O DAX SDK para Java coleta somente métricas `ApiCallSuccessful` e `ApiCallDuration`.

TryDax.java

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
    public static void main(String[] args) throws Exception {
        DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
                .build();

        DynamoDbAsyncClient daxClient = null;
        if (args.length >= 1) {
            daxClient = ClusterDaxAsyncClient.builder()
                    .overrideConfiguration(Configuration.builder()
                            .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
                            .build())
                    .build();
        }

        String tableName = "TryDaxTable";
```

```
        System.out.println("Creating table...");
        createTable(tableName, ddbClient);

        System.out.println("Populating table...");
        writeData(tableName, ddbClient, 100, 10);

        DynamoDbAsyncClient testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        getItemTest(tableName, testClient, 100, 10, 5);

        // Query
        queryTest(tableName, testClient, 100, 2, 9, 5);

        System.out.println("Deleting table...");
        deleteTable(tableName, ddbClient);
    }

    private static void createTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("Attempting to create table; please wait...");

            client.createTable(CreateTableRequest.builder()
                    .tableName(tableName)
                    .keySchema(KeySchemaElement.builder()
                            .keyType(KeyType.HASH)
                            .attributeName("pk")
                            .build(), KeySchemaElement.builder()
                            .keyType(KeyType.RANGE)
                            .attributeName("sk")
                            .build())
                    .attributeDefinitions(AttributeDefinition.builder()
                            .attributeName("pk")
                            .attributeType(ScalarAttributeType.N)
                            .build(), AttributeDefinition.builder()
```

```
                        .attributeName("sk")
                        .attributeType(ScalarAttributeType.N)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build()).get();
        client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
                .tableName(tableName)
                .build()).get();
        System.out.println("Successfully created table.");

    } catch (Exception e) {
        System.err.println("Unable to create table: ");
        e.printStackTrace();
    }
}

private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
    try {
        System.out.println("\nAttempting to delete table; please wait...");
        client.deleteTable(DeleteTableRequest.builder()
                .tableName(tableName)
                .build()).get();
        client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
                .tableName(tableName)
                .build()).get();
        System.out.println("Successfully deleted table.");

    } catch (Exception e) {
        System.err.println("Unable to delete table: ");
        e.printStackTrace();
    }
}

private static void writeData(String tableName, DynamoDbAsyncClient client, int
pkmax, int skmax) {
    System.out.println("Writing data to the table...");

    int stringSize = 1000;
    StringBuilder sb = new StringBuilder(stringSize);
    for (int i = 0; i < stringSize; i++) {
        sb.append('X');
    }
    String someData = sb.toString();
```

```
        try {
            for (int ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (int isk = 1; isk <= skmax; isk++) {
                    client.putItem(PutItemRequest.builder()
                            .tableName(tableName)
                            .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData",
attr(someData)))
                            .build()).get();
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    private static AttributeValue attr(int n) {
        return AttributeValue.builder().n(String.valueOf(n)).build();
    }

    private static AttributeValue attr(String s) {
        return AttributeValue.builder().s(s).build();
    }

    private static void getItemTest(String tableName, DynamoDbAsyncClient client, int
pk, int sk, int iterations) {
        long startTime, endTime;
        System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-"
+ sk);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (int ipk = 1; ipk <= pk; ipk++) {
                    for (int isk = 1; isk <= sk; isk++) {
                        client.getItem(GetItemRequest.builder()
                                .tableName(tableName)
                                .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
                                .build()).get();
                    }
                }
            } catch (Exception e) {
```

```
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            for (int ipk = 1; ipk <= pk; ipk++) {
                try {
                    // Pagination API for Query.
                    client.queryPaginator(QueryRequest.builder()
                            .tableName(tableName)
                            .keyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                            .expressionAttributeValues(Map.of(":pkval", attr(ipk),
":skval1", attr(sk1), ":skval2", attr(sk2)))
                            .build()).items().subscribe((item) -> {
                    }).get();
                } catch (Exception e) {
                    System.err.println("Unable to query table:");
                    e.printStackTrace();
                }
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk);
        }
    }

    private static void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
(1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
1000000.0));
    }
```

```
}
```

.NET e DAX

Siga estas etapas para executar o exemplo .NET na sua instância do Amazon EC2.

> **Note**
>
> Este tutorial usa o SDK .NET 6, mas também funciona com o SDK .NET Core. Ele mostra como você pode executar um programa na sua Amazon VPC padrão para acessar seu cluster do Amazon DynamoDB Accelerator (DAX). Se você preferir, pode usar o AWS Toolkit for Visual Studio para escrever um aplicativo .NET e implantá-lo em sua VPC.
> Para obter mais informações, consulte Criação e implantação de aplicações do Elastic Beanstalk no .NET usando o AWS Toolkit for Visual Studio no Guia do desenvolvedor do AWS Elastic Beanstalk.

Como executar o exemplo do .NET para o DAX

1. Acesse a página de downloads da Microsoft e baixe o SDK do .NET 6 (ou .NET Core) para Linux mais recente. O arquivo obtido por download é o `dotnet-sdk-N.N.N-linux-x64.tar.gz`.
2. Extraia os arquivos do SDK.

   ```
   mkdir dotnet
   tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
   ```

   Substitua *N.N.N* pelo número de versão real do SDK do .NET (por exemplo: `6.0.100`).
3. Verifique a instalação.

   ```
   alias dotnet=$HOME/dotnet/dotnet
   dotnet --version
   ```

   Isso deve imprimir o número da versão do SDK .NET.

   > **Note**
   >
   > Em vez de obter o número da versão, é possível que você receba o seguinte erro:

erro: libunwind.so.8: não é possível abrir o arquivo do objeto compartilhado: o arquivo ou diretório não existe
Para resolver o erro, instale o pacote `libunwind`.

```
sudo yum install -y libunwind
```

Depois disso, você deve conseguir executar o comando `dotnet --version` sem nenhum erro.

4. Crie um novo projeto .NET.

```
dotnet new console -o myApp
```

Ele levará alguns minutos executando uma configuração isolada. Ao concluir, execute o projeto de exemplo.

```
dotnet run --project myApp
```

Você deverá receber a seguinte mensagem: `Hello World!`

5. O arquivo `myApp/myApp.csproj` contém os metadados do projeto. Para usar o cliente do DAX em sua aplicação, modifique o arquivo para que seja semelhante ao seguinte.

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net6.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. Baixe o código-fonte do programa de exemplo (arquivo `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Quando o download for concluído, extraia os arquivos de origem.

```
unzip TryDax.zip
```

7. Agora, execute os programas de exemplo, um por vez. Para cada programa, você copiará seu conteúdo em `myApp/Program.cs` e, seguida, executará o projeto `MyApp`.

   Execute os seguintes programas .NET. O primeiro programa cria uma tabela do DynamoDB chamada `TryDaxTable`. O segundo programa grava dados na tabela.

   ```
   cp TryDax/dotNet/01-CreateTable.cs myApp/Program.cs
   dotnet run --project myApp

   cp TryDax/dotNet/02-Write-Data.cs myApp/Program.cs
   dotnet run --project myApp
   ```

8. Depois, execute alguns programas para executar as operações `GetItem`, `Query` e `Scan` no cluster do DAX. Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

   - Usando o console do DynamoDB: escolha seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Usando a AWS CLI: insira o comando a seguir.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     O endpoint do cluster é mostrado na saída, como no exemplo a seguir.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Agora execute os seguintes programas, especificando o endpoint do cluster como um parâmetro de linha de comando. (Substitua o endpoint de exemplo pelo endpoint do seu cluster do DAX real.)

```
cp TryDax/dotNet/03-GetItem-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/04-Query-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/05-Scan-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
```

Anote as informações de tempo: o número de milissegundos necessários para os testes `GetItem`, `Query` e `Scan`.

9. Execute o seguinte programa .NET para excluir a `TryDaxTable`.

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

Para obter mais informações sobre esses programas, consulte as seguintes seções:

- 01-CreateTable.cs
- 02-Write-Data.cs
- 03-GetItem-Test.cs
- 04-Query-Test.cs
- 05-Scan-Test.cs
- 06-DeleteTable.cs

01-CreateTable.cs

O programa `01-CreateTable.cs` cria uma tabela (`TryDaxTable`). Os demais programas .NET nesta seção dependem dessa tabela.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new CreateTableRequest()
            {
                TableName = tableName,
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement{ AttributeName = "pk",KeyType = "HASH"},
                    new KeySchemaElement{ AttributeName = "sk",KeyType = "RANGE"}
                },
                AttributeDefinitions = new List<AttributeDefinition>() {
                    new AttributeDefinition{ AttributeName = "pk",AttributeType = "N"},
                    new AttributeDefinition{ AttributeName = "sk",AttributeType  = "N"}
                },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 10
                }
            };

            var response = await client.CreateTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

02-Write-Data.cs

O programa `02-Write-Data.cs` grava dados de teste em `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            string someData = new string('X', 1000);
            var pkmax = 10;
            var skmax = 10;

            for (var ipk = 1; ipk <= pkmax; ipk++)
            {
                Console.WriteLine($"Writing {skmax} items for partition key: {ipk}");
                for (var isk = 1; isk <= skmax; isk++)
                {
                    var request = new PutItemRequest()
                    {
                        TableName = tableName,
                        Item = new Dictionary<string, AttributeValue>()
                       {
                            { "pk", new AttributeValue{N = ipk.ToString() } },
                            { "sk", new AttributeValue{N = isk.ToString() } },
                            { "someData", new AttributeValue{S = someData } }
                       }
                    };

                    var response = await client.PutItemAsync(request);
                }
            }

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
```

```
        }
    }
}
```

03-GetItem-Test.cs

O programa `03-GetItem-Test.cs` executa operações `GetItem` em `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 1;
            var sk = 10;
            var iterations = 5;

            var startTime = System.DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                for (var ipk = 1; ipk <= pk; ipk++)
                {
```

```
                    for (var isk = 1; isk <= sk; isk++)
                    {
                        var request = new GetItemRequest()
                        {
                            TableName = tableName,
                            Key = new Dictionary<string, AttributeValue>() {
                            {"pk", new AttributeValue {N = ipk.ToString()} },
                            {"sk", new AttributeValue {N = isk.ToString() } }
                        }
                        };
                        var response = await client.GetItemAsync(request);
                        Console.WriteLine($"GetItem succeeded for pk: {ipk},sk:
 {isk}");
                    }
                }
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

04-Query-Test.cs

O programa `04-Query-Test.cs` executa operações `Query` em `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
```

```
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");


            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 5;
            var sk1 = 2;
            var sk2 = 9;
            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new QueryRequest()
                {
                    TableName = tableName,
                    KeyConditionExpression = "pk = :pkval and sk between :skval1
and :skval2",
                    ExpressionAttributeValues = new Dictionary<string,
AttributeValue>() {
                            {":pkval", new AttributeValue {N = pk.ToString()} },
                            {":skval1", new AttributeValue {N = sk1.ToString()} },
                            {":skval2", new AttributeValue {N = sk2.ToString()} }
                    }
                };
                var response = await client.QueryAsync(request);
                Console.WriteLine($"{i}: Query succeeded");

            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
```

```
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

05-Scan-Test.cs

O programa `05-Scan-Test.cs` executa operações `Scan` em `TryDaxTable`.

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
```

```
                var request = new ScanRequest()
                {
                    TableName = tableName
                };
                var response = await client.ScanAsync(request);
                Console.WriteLine($"{i}: Scan succeeded");
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

06-DeleteTable.cs

O programa `06-DeleteTable.cs` exclui `TryDaxTable`. Execute esse programa depois de concluir o teste.

```
using System;
using System.Threading.Tasks;
using Amazon.DynamoDBv2.Model;
using Amazon.DynamoDBv2;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new DeleteTableRequest()
            {
```

```
                TableName = tableName
            };

            var response = await client.DeleteTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

Python e DAX

Siga este procedimento para executar a aplicação de exemplo em Python na instância do Amazon EC2.

Para executar o exemplo de Python para DAX

1. Instale o cliente do Python do DAX usando o utilitário `pip`.

   ```
   pip install amazon-dax-client
   ```

2. Baixe o código-fonte do programa de exemplo (arquivo `.zip`).

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Quando o download for concluído, extraia os arquivos de origem.

   ```
   unzip TryDax.zip
   ```

3. Execute os seguintes programas do Python. O primeiro programa cria uma tabela do Amazon DynamoDB chamada `TryDaxTable`. O segundo programa grava dados na tabela.

   ```
   python 01-create-table.py
   python 02-write-data.py
   ```

4. Execute os seguintes programas do Python.

   ```
   python 03-getitem-test.py
   ```

```
python 04-query-test.py
python 05-scan-test.py
```

Anote as informações de tempo: o número de milissegundos necessários para os testes `GetItem`, `Query` e `Scan`.

5. Na etapa anterior, você executou os programas no endpoint do DynamoDB. Agora execute os programas novamente, mas, desta vez, as operações `GetItem`, `Query` e `Scan` são processadas pelo cluster do DAX.

   Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

   - Usando o console do DynamoDB: escolha seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Usando a AWS CLI: insira o comando a seguir.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     O endpoint do cluster é mostrado na saída, como neste exemplo.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Execute os programas novamente, mas, desta vez, especifique o endpoint do cluster como um parâmetro de linha de comando.

   ```
   python 03-getitem-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com
   python 04-query-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com
   python 05-scan-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
   ```

Observe o restante da saída e anote as informações de tempo. Os tempos decorridos para `GetItem`, `Query` e `Scan` devem ser significativamente mais baixos com o DAX do que com o DynamoDB.

6. Execute o seguinte programa do Python para excluir a `TryDaxTable`.

```
python 06-delete-table.py
```

Para obter mais informações sobre esses programas, consulte as seguintes seções:

- 01-create-table.py
- 02-write-data.py
- 03-getitem-test.py
- 04-query-test.py
- 05-scan-test.py
- 06-delete-table.py

01-create-table.py

O programa `01-create-table.py` cria uma tabela (`TryDaxTable`). Os demais programas Python nesta seção dependem dessa tabela.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
```

```
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

02-write-data.py

O programa `02-write-data.py` grava dados de teste em `TryDaxTable`.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
```

```
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

03-getitem-test.py

O programa `03-getitem-test.py` executa operações `GetItem` em `TryDaxTable`. Este exemplo é fornecido para a região eu-west-1.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
```

```
    if dyn_resource is None:
        dyn_resource = boto3.resource('dynamodb')

    table = dyn_resource.Table('TryDaxTable')
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(Key={
                    'partition_key': partition_key,
                    'sort_key': sort_key
                })
                print('.', end='')
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        'endpoint_url', nargs='?',
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.")
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url,
 region_name='eu-west-1') as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax)
    else:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the Boto3 client.")
        test_start, test_end = get_item_test(
            test_key_count, test_iterations)
    print(f"Total time: {test_end - test_start:.4f} sec. Average time: "
          f"{(test_end - test_start)/ test_iterations}.")
```

04-query-test.py

O programa `04-query-test.py` executa operações `Query` em `TryDaxTable`.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this value.
    :param sort_keys: The range of sort key values for the query. The query returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end
```

```
if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations, dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

05-scan-test.py

O programa `05-scan-test.py` executa operações `Scan` em `TryDaxTable`.

```
import argparse
import time
import sys
import amazondax
import boto3
```

```
def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
```

```
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

06-delete-table.py

O programa `06-delete-table.py` exclui `TryDaxTable`. Execute esse programa depois de concluir o teste de funcionalidade do Amazon DynamoDB Accelerator (DAX).

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

Como modificar uma aplicação existente para usar o DAX

Se você já tem uma aplicação Java que usa o Amazon DynamoDB, poderá modificá-la para que ela possa acessar seu cluster do DynamoDB Accelerator (DAX). Não é necessário reescrever toda a aplicação porque o cliente Java do DAX é muito semelhante ao cliente de nível inferior do DynamoDB incluído no AWS SDK for Java 2.x. Consulte Como trabalhar com itens no DynamoDB para obter detalhes.

> **Note**
>
> Este exemplo usa o AWS SDK for Java 2.x. Para a versão herdada do SDK for Java 1.x, consulte Modificar uma aplicação do SDK for Java 1.x existente para usar o DAX.

Para modificar o programa, substitua o cliente do DynamoDB por um cliente do DAX.

```
Region region = Region.US_EAST_1;

// Create an asynchronous DynamoDB client
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
                .region(region)
                .build();

// Create an asynchronous DAX client
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
                .overrideConfiguration(Configuration.builder()
                    .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
                    .region(region)
                    .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
 metric collection
                    .build())
                .build();
```

Também é possível usar a biblioteca de alto nível que faz parte do AWS SDK for Java 2.x substituindo-se o cliente DynamoDB por um cliente DAX.

```
Region region = Region.US_EAST_1;
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
            .region(region)
            .build())
        .build();

DynamoDbEnhancedAsyncClient enhancedClient = DynamoDbEnhancedAsyncClient.builder()
        .dynamoDbClient(dax)
        .build();
```

Para obter mais informações, consulte Mapeamento de itens em tabelas do DynamoDB.

Gerenciar clusters do DAX

Esta seção aborda algumas das tarefas comuns de gerenciamento de clusters do Amazon DynamoDB Accelerator (DAX).

Tópicos

- Permissões do IAM para gerenciar um cluster do DAX
- Escalar um cluster do DAX
- Personalizar as configurações do cluster do DAX
- Configurar definições de TTL
- Compatibilidade com marcação para o DAX
- Integração do AWS CloudTrail
- Excluir um cluster do DAX

Permissões do IAM para gerenciar um cluster do DAX

Quando você administra um cluster do DAX usando o AWS Management Console ou a AWS Command Line Interface (AWS CLI), é altamente recomendável restringir o escopo das ações que os usuários podem executar. Ao fazer isso, você pode ajudar a reduzir os riscos, enquanto segue o princípio do privilégio mínimo.

A seguinte discussão aborda o controle de acesso das APIs de gerenciamento do DAX. Para obter mais informações consulte Amazon DynamoDB Accelerator na Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB).

> **Note**
>
> Para obter informações mais detalhadas sobre o gerenciamento de permissões do AWS Identity and Access Management (IAM), consulte:
>
> - IAM e criação de clusters do DAX: Criar um cluster do DAX.
> - IAM e operações de plano de dados do DAX: Controle de acesso do DAX.

Para as APIs de gerenciamento do DAX, não é possível incluir um recurso específico no escopo das ações da API. O elemento `Resource` deve ser definido como `"*"`. Isso é diferente das operações da API do plano de dados do DAX, como `GetItem`, `Query` e `Scan`. As operações de plano de dados são expostas por meio do cliente Java do DAX, e essas operações podem incluir recursos específicos no escopo.

Para ilustrar, considere o seguinte documento de política do IAM:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Suponha que a intenção dessa política seja permitir chamadas de API de gerenciamento do DAX para o cluster `DAXCluster01` – e apenas para esse cluster.

Agora, suponha que um usuário emita o seguinte comando da AWS CLI.

```
aws dax describe-clusters
```

Esse comando falha com uma exceção de Não autorizado, pois a chamada à API `DescribeClusters` subjacente não pode ser colocada no escopo de um cluster específico. Embora a política seja sintaticamente válida, o comando falha porque o elemento `Resource` deve ser definido como `"*"`. No entanto, se o usuário executar um programa que envie chamadas do plano de dados do DAX (como `GetItem` ou `Query`) para o `DAXCluster01`, essas chamadas serão bem-sucedidas. Isso ocorre porque as APIs do plano de dados do DAX podem incluir recursos específicos no escopo (nesse caso, `DAXCluster01`).

Para escrever uma única política abrangente do IAM que inclua as APIs de gerenciamento do DAX e as APIs do plano de dados do DAX, sugerimos incluir duas instruções distintas no documento da política. Uma dessas instruções deve lidar com as APIs de plano de dados do DAX, enquanto a outra instrução lida com as APIs de gerenciamento do DAX.

A política de exemplo a seguir mostra essa abordagem. Observe como a declaração `DAXDataAPIs` é colocada no escopo do recurso `DAXCluster01`, mas o recurso para `DAXManagementAPIs` deve ser `"*"`. As ações mostradas em cada declaração são apenas para ilustração. É possível personalizá-las conforme necessário para seu aplicativo.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DAXDataAPIs",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]},
            {
            "Sid": "DAXManagementAPIs",
            "Action": [
                "dax:CreateParameterGroup",
                "dax:CreateSubnetGroup",
                "dax:DecreaseReplicationFactor",
                "dax:DeleteCluster",
                "dax:DeleteParameterGroup",
                "dax:DeleteSubnetGroup",
                "dax:DescribeClusters",
                "dax:DescribeDefaultParameters",
```

```
                "dax:DescribeEvents",
                "dax:DescribeParameterGroups",
                "dax:DescribeParameters",
                "dax:DescribeSubnetGroups",
                "dax:IncreaseReplicationFactor",
                "dax:ListTags",
                "dax:RebootNode",
                "dax:TagResource",
                "dax:UntagResource",
                "dax:UpdateCluster",
                "dax:UpdateParameterGroup",
                "dax:UpdateSubnetGroup"
                ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Escalar um cluster do DAX

Há duas opções disponíveis para escalar um cluster do DAX. A primeira opção é a escalabilidade horizontal, na qual você adiciona réplicas de leitura ao cluster. A segunda opção é a escalabilidade vertical, na qual você seleciona diferentes tipos de nó. Para obter conselhos sobre como abordar a escolha de um tamanho de cluster apropriado e um tipo de nó para seu aplicativo, consulte Guia de dimensionamento de clusters do DAX.

Escalabilidade horizontal

Com a escalabilidade horizontal, você pode melhorar o throughput para operações de leitura adicionando mais réplicas de leitura ao cluster. Um único cluster do DAX aceita até 10 réplicas de leitura, e você pode adicionar ou remover réplicas enquanto o cluster está em execução.

Ao adicionar um novo nó, é necessário sincronizar os dados de cache de outro nó do mesmo nível. Portanto, o tempo de adição varia com base no tamanho do cache e na workload da aplicação. Recomendamos que você escale previamente o cluster para atender aos picos de tráfego esperados. Para obter informações sobre diretrizes de dimensionamento correto e recomendações de monitoramento, consulte Guia de dimensionamento de clusters do DAX.

Os exemplos de AWS CLI a seguir mostram como aumentar ou diminuir o número de nós. O argumento `--new-replication-factor` especifica o número total de nós no cluster. Um dos nós é o nó primário, e os outros nós são réplicas de leitura.

```
aws dax increase-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 5
```

```
aws dax decrease-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 3
```

> **Note**
>
> O status do cluster muda para `modifying` quando você modifica o fator de replicação. O status muda para `available` quando a modificação é concluída.

Escala vertical

Se você tiver um grande conjunto de dados de trabalho, seu aplicativo poderá se beneficiar do uso de maiores tipos de nós. Os nós maiores podem permitir que o cluster armazene mais dados na memória, o que reduz os erros de cache e melhora o desempenho geral do aplicativo. (Todos os nós em um cluster do DAX devem ser do mesmo tipo.)

Se o cluster do DAX tiver uma alta taxa de operações de gravação ou falhas de cache, sua aplicação também poderá se beneficiar do uso de tipos maiores de nó. Operações de gravação e erros de cache consomem recursos no nó primário do cluster. Portanto, usar tipos maiores de nós pode aumentar o desempenho do nó primário e permitir um throughput mais alto para esses tipos de operações.

Não é possível modificar os tipos de nós em um cluster do DAX em execução. Em vez disso, você deve criar um novo cluster com o tipo de nó desejado. Para obter uma lista dos tipos de nó compatíveis, consulte Nós.

Você pode criar um novo cluster do DAX usando via AWS Management Console, AWS CloudFormation, AWS CLI ou AWS SDK. (Para a AWS CLI, use o parâmetro `--node-type` para especificar o tipo de nó.)

Personalizar as configurações do cluster do DAX

Quando você cria um cluster do DAX, as configurações padrão a seguir são usadas:

- Remoção automática de cache habilitada com vida útil (TTL) de 5 minutos
- Nenhuma preferência para zonas de disponibilidade
- Nenhuma preferência para janelas de manutenção
- Notificações desabilitadas

Para novos clusters, você pode personalizar as configurações durante a criação. Para fazer isso no AWS Management Console, desmarque Use default settings (Usar configurações padrão) para modificar os seguintes ajustes:

- Network and Security (Rede e segurança): permite que você execute nós de cluster do DAX individuais em diferentes zonas de disponibilidade dentro da região da AWS atual. Se você escolher No Preference (Sem preferência), os nós serão distribuídos entre as zonas de disponibilidade automaticamente.
- Parameter Group (Grupo de parâmetros): um conjunto nomeado de parâmetros que são aplicados a cada nó no cluster. Você pode usar um grupo de parâmetros para especificar o comportamento do TTL do cache. Você pode alterar o valor de qualquer parâmetro determinado em um grupo de parâmetros (exceto o grupo de parâmetros padrão `default.dax.1.0`) a qualquer momento.
- Maintenance Window (Janela de manutenção): um período semanal durante o qual atualizações e patches são aplicados aos nós do cluster. Você pode escolher o dia e a hora de início e a duração da janela de manutenção. Se você escolher No Preference (Sem preferência), a janela de manutenção será selecionada aleatoriamente em um bloco de tempo de 8 horas por região. Para obter mais informações, consulte Janela de manutenção.

> **Note**
>
> Parameter Group (Grupo de parâmetros) e Maintenance Window (Janela de manutenção) também podem ser modificados a qualquer momento em um cluster em execução.

Quando um evento de manutenção ocorre, o DAX pode notificar você usando o Amazon Simple Notification Service (Amazon SNS). Para configurar notificações, escolha uma opção no seletor

Topic for SNS notification (Tópico de notificação do SNS). Você pode criar um novo tópico do Amazon SNS ou usar um tópico existente.

Para obter informações sobre como criar um tópico do SNS e assiná-lo, consulte Conceitos básicos do Amazon SNS no Guia do desenvolvedor do Amazon Simple Notification Service.

Configurar definições de TTL

O DAX mantém dois caches para dados que ele lê do DynamoDB:

- Cache de itens: para itens recuperados via `GetItem` ou `BatchGetItem`.
- Cache de consultas: para conjuntos de resultados recuperados via `Query` ou `Scan`.

Para obter mais informações, consulte Cache de itens e Cache de consultas.

O TTL padrão de cada um desses caches é 5 minutos. Se desejar usar configurações de TTL diferentes, você poderá iniciar um cluster do DAX usando um grupo de parâmetros personalizados. Para fazer isso no console, escolha DAX | Parameter groups (DAX | grupos de parâmetros) no painel de navegação.

Você também pode realizar essas tarefas usando a AWS CLI. O exemplo a seguir mostra como iniciar um novo cluster DAX usando um grupo de parâmetros personalizados. Neste exemplo, o TTL do cache de itens é definido como 10 minutos, e o TTL do cache de consultas é definido como 3 minutos.

1. Crie um novo grupo de parâmetros.

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. Defina o TTL do cache de itens para 10 minutos (600000 milissegundos).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. Defina o TTL do cache de consultas para 3 minutos (180000 milissegundos).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
```

```
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. Verifique se os parâmetros foram configurados corretamente.

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

Agora você pode iniciar um novo cluster do DAX com este grupo de parâmetros.

```
aws dax create-cluster \
    --cluster-name MyNewCluster \
    --node-type dax.r3.large \
    --replication-factor 3 \
    --iam-role-arn arn:aws:iam::123456789012:role/DAXServiceRole \
    --parameter-group custom-ttl
```

> **Note**
>
> Não é possível modificar um grupo de parâmetros que está sendo usado por uma instância do DAX em execução.

Compatibilidade com marcação para o DAX

Muitos serviços da AWS, incluindo o DynamoDB, oferecem suporte à marcação com tags: a habilidade de rotular recursos com nomes definidos pelo usuário. Você pode atribuir tags a clusters do DAX, o que permite identificar rapidamente todos os seus recursos da AWS que têm a mesma tag ou categorizar os faturamentos da AWS pelas tags que você atribui.

Para obter mais informações, consulte Adicionar etiquetas e rótulos a recursos no DynamoDB.

Usar a AWS Management Console

Para gerenciar tags de cluster do DAX

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, em DAX, escolha Clusters.
3. Escolha o cluster com o qual você deseja trabalhar.

4. Escolha a guia Tags. Você pode adicionar, listar, editar ou excluir suas tags aqui.

   Quando as configurações estiverem conforme você deseja, escolha Apply Changes (Aplicar alterações).

Como usar o AWS CLI

Ao usar a AWS CLI para gerenciar tags de cluster do DAX, você deve primeiro determinar o nome do recurso da Amazon (ARN) do cluster. O exemplo a seguir mostra como determinar o ARN de um cluster chamado `MyDAXCluster`.

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

Na saída, o ARN será semelhante a este: `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

O exemplo a seguir mostra como atribuir tags ao cluster.

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

Para listar todas as tags de um cluster.

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

Para remover uma tag, especifique a chave dela.

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

Integração do AWS CloudTrail

O DAX é integrado ao AWS CloudTrail, o que permite auditar as atividades do cluster do DAX. É possível usar os logs do CloudTrail para visualizar todas as alterações que foram feitas no nível do

cluster. Também é possível ver as alterações em componentes do cluster, como nós, grupos de sub-redes e grupos de parâmetros. Para obter mais informações, consulte Registrar em log as operações do DynamoDB usando o AWS CloudTrail.

Excluir um cluster do DAX

Se você não estiver mais usando um cluster do DAX, deverá excluí-lo para evitar ser cobrado por recursos não utilizados.

É possível excluir um cluster do DAX usando o console ou a AWS CLI. Veja um exemplo a seguir.

```
aws dax delete-cluster --cluster-name mydaxcluster
```

Monitorar o DynamoDB Accelerator

O monitoramento é uma parte importante para manter a confiabilidade, a disponibilidade e a performance do Amazon DynamoDB Accelerator (DAX) e das soluções da AWS. Você deve coletar dados de monitoramento de todas as partes de sua solução da AWS para facilitar a depuração de uma falha multipontos, caso ela ocorra.

Antes de começar a monitorar o DAX, crie um plano de monitoramento que inclua as respostas para as seguintes perguntas:

- Quais são seus objetivos de monitoramento?
- Quais recursos você vai monitorar?
- Com que frequência você vai monitorar esses recursos?
- Quais ferramentas de monitoramento você usará?
- Quem realizará o monitoramento das tarefas?
- Quem deve ser notificado quando algo der errado?

Tópicos

- Ferramentas de monitoramento para o DynamoDB Accelerator
- Monitorar o com o Amazon CloudWatch
- Registrar operações do DAX usando o AWS CloudTrail

Ferramentas de monitoramento para o DynamoDB Accelerator

O AWS fornece ferramentas que você pode usar para monitorar o Amazon DynamoDB Accelerator (DAX). É possível configurar algumas dessas ferramentas para fazer o monitoramento para você, e algumas delas exigem intervenção manual. Recomendamos que as tarefas de monitoramento sejam automatizadas ao máximo possível.

Tópicos

- Ferramentas de monitoramento automatizadas
- Ferramentas de monitoramento manual

Ferramentas de monitoramento automatizadas

Você pode usar as seguintes ferramentas de monitoramento automatizadas para observar o DAX e gerar relatórios quando algo estiver errado:

- Amazon CloudWatch Alarms: observe uma única métrica ao longo de um período que você especificar e realize uma ou mais ações com base no valor da métrica em relação a um limite ao longo de vários períodos. A ação é uma notificação enviada para um tópico do Amazon Simple Notification Service (Amazon SNS) ou uma política do Amazon EC2 Auto Scaling. Os alarmes do CloudWatch não invocam ações simplesmente por estarem em um estado específico. O estado deve ter sido alterado e mantido por um número específico de períodos. Para ter mais informações, consulte Monitorar métricas no DynamoDB com o Amazon CloudWatch.
- Amazon CloudWatch Logs: monitore, armazene e acesse seus arquivos de log do AWS CloudTrail ou de outras origens. Para ter mais informações, consulte Monitoring Log Files no Guia do usuário do Amazon CloudWatch.
- Amazon CloudWatch Events: faça correspondência de eventos e direcione-os a uma ou mais funções ou streams de destino para fazer alterações, capturar informações de estado e realizar ações corretivas. Para obter mais informações, consulte O que é o Amazon CloudWatch Events? no Guia do usuário do Amazon CloudWatch.
- Monitoramento de log AWS CloudTrail: compartilhe arquivos de log entre contas, monitore os arquivos de log do CloudTrail em tempo real enviando-os para o CloudWatch Logs, escreva aplicações de processamento de logs em Java e confirme se os arquivos de log não foram alterados após a entrega pelo CloudTrail. Para obter mais informações, consulte Trabalhando com arquivos de log do CloudTrail no Guia do usuário do AWS CloudTrail.

Ferramentas de monitoramento manual

Outra parte importante do monitoramento do DAX é o monitoramento manual dos itens que os alarmes do CloudWatch não abrangem. O DAX, o CloudWatch, o Trusted Advisor e outros painéis do AWS Management Console fornecem uma visão rápida do estado do ambiente da AWS. Recomendamos verificar também os arquivos de registro do DAX.

- O painel do DAX mostra o seguinte:
  - Integridade de serviço
- A página inicial do CloudWatch mostra o seguinte:
  - Alertas e status atual
  - Gráficos de alertas e recursos
  - Estado de integridade do serviço

  Além disso, é possível usar o CloudWatch para fazer o seguinte:
  - Criar painéis personalizados para monitorar os serviços de seu interesse.
  - Colocar em gráfico dados de métrica para solucionar problemas e descobrir tendências.
  - Pesquisar e procurar todas as métricas de recursos da AWS.
  - Criar e editar alertas para ser notificado sobre problemas.

Monitorar o com o Amazon CloudWatch

É possível monitorar o DynamoDB Accelerator (DAX) usando o Amazon CloudWatch, que coleta e processa dados brutos do DAX e os transforma em métricas legíveis quase em tempo real. Essas estatísticas são gravadas durante um período de duas semanas. Você pode acessar as informações históricas para obter uma perspectiva melhor sobre o desempenho do aplicativo web ou do serviço. Por padrão, os dados de métrica do DAX são enviados para o CloudWatch automaticamente. Para obter mais informações, consulte O que é o Amazon CloudWatch? no Guia do usuário do Amazon CloudWatch.

Tópicos

- Como usar as métricas do DAX?
- Visualizar métricas e dimensões do DAX
- Criar alarmes do CloudWatch para monitorar o DAX
- Monitoramento da produção

Como usar as métricas do DAX?

As métricas informadas pelo DAX fornecem informações que você pode analisar de diferentes maneiras. A lista a seguir mostra alguns usos comuns para as métricas. Essas são sugestões para você começar, e não uma lista abrangente.

Como?	Métricas relevantes
Determinar se ocorreu algum erro do sistema	Monitore `FaultRequestCount` para determinar se alguma solicitação resultou em um código HTTP 500 (erro de servidor) . Isso pode indicar um erro de serviço interno do DAX ou um HTTP 500 na métrica SystemErrors da tabela subjacente.
Determinar se ocorreu algum erro do usuário	Monitore `ErrorRequestCount` para determinar se alguma solicitação resultou em um código HTTP 400 (erro de cliente). Se você vir a contagem de erros aumentando, investigue e verifique se você está enviando solicitações de clientes corretas.
Determinar se ocorreu alguma ausência no cache	Monitore `ItemCacheMisses` para determinar o número de vezes que um item não foi encontrado no cache e `QueryCacheMisses` e `ScanCacheMisses` para determinar o número de vezes que uma consulta ou um resultado de verificação não foi encontrado no cache.
Monitorar taxas de acerto do cache	Use a Matemática de métricas do CloudWatch para definir uma métrica da taxa de acertos do cache usando expressões matemáticas. Por exemplo, para o cache do item, você pode usar a expressão m1/SUM([m1, m2])*100, em que m1 é a métrica `ItemCacheHits` e m2 é a métrica `ItemCacheMisses` para seu cluster. Para os caches de verificação e consulta, você pode seguir o mesmo padrão usando a métrica do cache de verificação e consulta correspondente.

Visualizar métricas e dimensões do DAX

Quando você interage com o Amazon DynamoDB, ele envia as seguintes métricas e dimensões ao Amazon CloudWatch. É possível usar os procedimentos a seguir para visualizar as métricas do DynamoDB Accelerator (DAX).

Para visualizar métricas (console)

As métricas são agrupadas primeiro pelo namespace do serviço e, em seguida, por várias combinações de dimensão dentro de cada namespace.

1. Abra o console do CloudWatch em https://console.aws.amazon.com/cloudwatch/.
2. No painel de navegação, selecione Métricas.
3. Selecione o namespace DAX.

Para visualizar as métricas (AWS CLI)

- Em um prompt de comando, use o seguinte comando.

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

Métricas e dimensões do DAX

As seções a seguir contêm as métricas e as dimensões que o DAX envia para o CloudWatch.

Métricas do DAX

As métricas a seguir estão disponíveis no DAX. O DAX envia métricas ao CloudWatch somente quando elas têm um valor diferente de zero.

> **Note**
>
> O Amazon CloudWatch agrega as seguintes métricas do DAX em intervalos de um minuto:
>
> - `CPUUtilization`
> - `CacheMemoryUtilization`
> - `NetworkBytesIn`
> - `NetworkBytesOut`

- BaselineNetworkBytesInUtilization
- BaselineNetworkBytesOutUtilization
- NetworkPacketsIn
- NetworkPacketsOut
- GetItemRequestCount
- BatchGetItemRequestCount
- BatchWriteItemRequestCount
- DeleteItemRequestCount
- PutItemRequestCount
- UpdateItemRequestCount
- TransactWriteItemsCount
- TransactGetItemsCount
- ItemCacheHits
- ItemCacheMisses
- QueryCacheHits
- QueryCacheMisses
- ScanCacheHits
- ScanCacheMisses
- TotalRequestCount
- ErrorRequestCount
- FaultRequestCount
- FailedRequestCount
- QueryRequestCount
- ScanRequestCount
- ClientConnections
- EstimatedDbSize
- EvictedSize
- CPUCreditUsage
- CPUCreditBalance
- CPUSurplusCreditBalance

- `CPUSurplusCreditsCharged`

Nem todas as estatísticas, como `Average` ou `Sum`, são aplicáveis a todas as métricas. No entanto, todos esses valores estão disponíveis por meio do console do DAX ou usando o console do CloudWatch, AWS CLI ou AWS SDKs para todas as métricas. Na tabela a seguir, cada métrica tem uma lista de estatísticas válidas aplicáveis a essa métrica.

Métrica	Descrição
`CPUUtilization`	O percentual de utilização da CPU do nó ou cluster. Unidades: `Percent` Estatística válida: • `Minimum` • `Maximum` • `Average`
`CacheMemoryUtilization`	O percentual de memória cache disponível que está em uso pelo cache de item e cache de consulta no nó ou cluster. Os dados armazenados em cache começam a ser despejados antes que a utilização da memória atinja 100% (consulte a métrica `EvictedSize`). Se `CacheMemoryUtilization` atinge 100% em qualquer nó, as solicitações de gravação serão limitadas e você deverá considerar a mudança para um cluster com um tipo de nó maior. Unidades: `Percent` Estatística válida: • `Minimum` • `Maximum` • `Average`

Métrica	Descrição
`NetworkBytesIn`	O número de bytes recebidos em todas as interfaces de rede pelo nó ou cluster. Unidades: `Bytes` Estatística válida: • `Minimum` • `Maximum` • `Average`
`NetworkBytesOut`	O número de bytes enviados em todas as interfaces de rede pelo nó ou cluster. Essa métrica identifica o volume de tráfego de saída em termos do número de bytes em um único nó ou cluster. Unidades: `Bytes` Estatística válida: • `Minimum` • `Maximum` • `Average`

Métrica	Descrição
`BaselineNetworkBytesInUtilization`	A porcentagem da largura de banda de rede de linha de base consumida em determinado momento para o tráfego de entrada. Para referência, 50% significa que metade da largura de banda de rede disponível para tráfego de entrada está sendo usada. Unidades: `Percent` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`BaselineNetworkBytesOutUtilization`	A porcentagem da largura de banda da rede de linha de base consumida em determinado momento para o tráfego de saída. Para referência, 50% significa que metade da largura de banda da rede disponível para tráfego de saída está sendo usada. Unidades: `Percent` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`NetworkPacketsIn`	O número de pacotes recebidos em todas as interfaces de rede pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average`
`NetworkPacketsOut`	O número de pacotes enviados em todas as interfaces de rede pelo nó ou cluster. Essa métrica identifica o volume de tráfego de saída em termos do número de pacotes em um único nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average`
`GetItemRequestCount`	O número de solicitações `GetItem` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`BatchGetItemRequestCount`	O número de solicitações `BatchGetItem` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`BatchWriteItemRequestCount`	O número de solicitações `BatchWriteItem` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`DeleteItemRequestCount`	O número de solicitações `DeleteItem` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`PutItemRequestCount`	O número de solicitações `PutItem` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`UpdateItemRequestCount`	O número de solicitações `UpdateItem` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TransactWriteItemsCount`	O número de solicitações `TransactWriteItems` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`TransactGetItemsCount`	O número de solicitações `TransactGetItems` manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ItemCacheHits`	O número de vezes que um item foi retornado do cache pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`ItemCacheMisses`	O número de vezes que um item não estava no cache do nó ou cluster e precisou ser recuperado do DynamoDB. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`QueryCacheHits`	O número de vezes que um resultado de consulta foi retornado do cache do nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`QueryCacheMisses`	O número de vezes que um resultado de consulta não estava no cache do nó ou cluster e precisou ser recuperado do DynamoDB. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanCacheHits`	O número de vezes que um resultado de verificação foi retornado do cache do nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`ScanCacheMisses`	O número de vezes que um resultado de verificação não estava no cache do nó ou cluster e precisou ser recuperado do DynamoDB. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TotalRequestCount`	O número total de solicitações manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`ErrorRequestCount`	Número total de solicitações que resultaram em um erro de usuário relatado pelo nó ou cluster. As solicitações que foram limitadas pelo nó ou cluster estão incluídas. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ThrottledRequestCount`	O número total de solicitações limitadas pelo nó ou cluster. As solicitações que foram limitadas pelo DynamoDB não são incluídas e podem ser monitoradas usando métricas do DynamoDB. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`FaultRequestCount`	Número total de solicitações que resultaram em um erro interno relatado pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`FailedRequestCount`	Número total de solicitações que resultaram em erro relatado pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`QueryRequestCount`	O número de solicitações de consulta manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanRequestCount`	O número de solicitações de varredura manipuladas pelo nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`ClientConnections`	O número de conexões simultâneas feitas pelos clientes ao nó ou cluster. Unidades: `Count` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`EstimatedDbSize`	Uma aproximação da quantidade de dados armazenados em cache no cache de item e no cache de consulta pelo nó ou cluster. Unidades: `Bytes` Estatística válida: • `Minimum` • `Maximum` • `Average`

Métrica	Descrição
`EvictedSize`	A quantidade de dados que foi removida pelo nó ou cluster para criar espaço para dados recém-solicitados. Se a taxa de erros aumentar e você observar que essa métrica também está crescendo, isso provavelmente significa que seu conjunto de trabalho aumentou. Você deve considerar migrar para um cluster com um tipo de nó maior. Unidades: `Bytes` Estatística válida: • `Minimum` • `Maximum` • `Average` • `Sum`

Métrica	Descrição
`CPUCreditUsage`	O número de créditos de CPU gastos pelo nó por utilização de CPU. Um crédito de CPU equivale a um vCPU em execução em 100% de utilização por um minuto ou a uma combinaçã o equivalente de vCPUs, utilização e tempo (por exemplo, um vCPU em execução a 50% de utilização por dois minutos ou dois vCPUs em execução a 25% de utilização por dois minutos). As métricas de crédito de CPU estão disponíveis a uma frequência de apenas 5 minutos. Se você especificar um período de mais cinco minutos, use a estatística `Sum` em vez da estatística `Average`. Unidades: `Credits (vCPU-minutes)` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`CPUCreditBalance`	O número de créditos ganhos de CPU que um nó acumulou desde que foi iniciado. Os créditos são acumulados no saldo de créditos após terem sido ganhos e são removidos do saldo de créditos quando são gastos. O saldo de créditos tem um limite máximo que é determinado pelo tamanho do nó do DAX. Depois que o limite for atingido, todos os novos créditos ganhos serão descartad os. Os créditos em `CPUCreditBalance` são disponibilizados para que o nó gaste e apresente intermitência com uma utilização de CPU acima da referência. Unidades: `Credits (vCPU-minutes)` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`CPUSurplusCreditBalance`	O número de créditos excedentes gastos por um nó do DAX quando seu valor `CPUCreditBalance` é zero. O valor `CPUSurplusCreditBalance` é pago pelos créditos de CPU ganhos. Se o número de créditos excedentes ultrapassar o número máximo de créditos que o nó pode ganhar em um período de 24 horas, os créditos excedentes gastos acima do limite máximo incorrerão em uma taxa adicional. Unidades: `Credits (vCPU-minutes)` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descrição
`CPUSurplusCreditsCharged`	O número de créditos excedentes gastos que não são pagos pelos créditos de CPU ganhos e que, portanto, incorrem em uma cobrança adicional. Os créditos excedentes ultrapassaram o número máximo de créditos que o nó pode obter em um período de 24 horas. Os créditos excedentes gastos acima do limite máximo são cobrados no final da hora ou quando o nó for terminado. Unidades: `Credits (vCPU-minutes)` Estatística válida: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

> **Note**
>
> As métricas `CPUCreditUsage`, `CPUCreditBalance`, `CPUSurplusCreditBalance` e `CPUSurplusCreditsCharged` estão disponíveis apenas para nós T3.

Dimensões para métricas do DAX

As métricas do DAX são qualificadas pelos valores para a conta, o ID de cluster ou a combinação do ID do cluster e o do nó. Você pode usar o console do CloudWatch para recuperar dados do DAX junto com qualquer uma das dimensões da tabela a seguir.

Dimensão	Descrição
`Account`	Fornece estatísticas agregadas em todos os nós em uma conta.
`ClusterId`	Limita os dados a um cluster.
`ClusterId, NodeId`	Limita os dados a um nó dentro de um cluster.

Criar alarmes do CloudWatch para monitorar o DAX

Você pode criar um alarme do Amazon CloudWatch que envia uma mensagem do Amazon Simple Notification Service (Amazon SNS) quando o alarme muda de estado. Um alarme observa uma única métrica por um período tempo que você especifica. Ele executa uma ou mais ações com base no valor da métrica em relação a um limite especificado ao longo de vários períodos. A ação é uma notificação que é enviada para um tópico do Amazon SNS ou uma política de Auto Scaling. Os alertas invocam ações apenas para alterações de estado mantidas. Os alarmes do CloudWatch não invocam ações só porque estão em um determinado estado. O estado deve ter sido alterado e mantido por uma quantidade especificada de períodos.

Como posso ser notificado de erros no cache de consulta?

1. Crie um tópico do Amazon SNS, `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`.

   Para obter mais informações, consulte Configurar Amazon Simple Notification Service no Guia do usuário do Amazon CloudWatch.

2. Crie o alarme.

```
aws cloudwatch put-metric-alarm \
    --alarm-name QueryCacheMissesAlarm  \
    --alarm-description "Alarm over query cache misses" \
    --namespace AWS/DAX \
    --metric-name QueryCacheMisses  \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 8 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --period 60 \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-west-2:522194210714:QueryMissAlarm
```

3. Teste o alarme.

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value ALARM
```

> **Note**
>
> Você também pode aumentar ou diminuir o limite para que seja adequado ao seu aplicativo. Você também pode usar a Matemática de métricas do CloudWatch para definir uma métrica da taxa de ausências no cache e definir um alarme para essa métrica.

Como posso recebe notificação se as solicitações causarem um erro interno no cluster?

1. Crie um tópico do Amazon SNS, `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`.

   Para obter mais informações, consulte Configurar Amazon Simple Notification Service no Guia do usuário do Amazon CloudWatch.

2. Crie o alarme.

```
aws cloudwatch put-metric-alarm \
```

```
    --alarm-name FaultRequestCountAlarm \
    --alarm-description "Alarm when a request causes an internal error" \
    --namespace AWS/DAX \
    --metric-name FaultRequestCount \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Teste o alarme.

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value ALARM
```

Monitoramento da produção

Você deve estabelecer uma referência de performance normal do DAX em seu ambiente, medindo a performance em vários momentos e em diferentes condições de carga. Ao monitorar o DAX, você deve pensar na possibilidade de armazenar os dados históricos de monitoramento. Esses dados armazenados fornecem a você uma linha de base com a qual comparar os dados de desempenho atuais, identificar padrões normais e anomalias de desempenho e criar métodos para solucionar problemas.

Para estabelecer uma linha de base, você deve monitorar, no mínimo, os seguintes itens durante o teste de carga e na produção:

- Utilização da CPU e solicitações limitadas, para que seja possível determinar se é necessário usar um tipo de nó maior no cluster. A utilização da CPU do cluster está disponível por meio da métrica `CPUUtilization` do CloudWatch. A estatística média dessa métrica fornece uma visão da utilização média da CPU em todos os nós do cluster. Para decisões de escalabilidade de cluster, recomendamos que você use a estatística máxima, que é a utilização máxima em todos os nós.

> **Note**
>
> A AWS melhorou o detalhamento da métrica `CPUUtilization`. Você pode observar alterações na métrica entre 17/5/2024 e 22/6/2024.

- A latência da operação (medida do lado do cliente) deve permanecer consistente dentro dos requisitos de latência da aplicação.
- As taxas de erro devem permanecer baixas, como pode ser visto nas métricas `ErrorRequestCount`, `FaultRequestCount` e `FailedRequestCount` do CloudWatch.
- Consumo de bytes de rede, para que você possa determinar se deve usar mais nós ou um tipo de nó maior no cluster. Para monitorar o consumo, é possível definir alertas nas métricas `BaselineNetworkBytesInUtilization` e `BaselineNetworkBytesOutUtilization` disponíveis no CloudWatch, que indicam o consumo percentual da largura de banda da rede disponível para seu tipo de instância, para tráfego de entrada e de saída, respectivamente.
- Utilização da memória cache e tamanho da remoção, para que você possa determinar se o tipo de nó do cluster tem memória suficiente para manter o conjunto de trabalho e, se não, mudar para um tipo de nó maior.

> **Note**
>
> No caso de um grande número de falhas e gravações no cache, a utilização da memória cache pode aumentar em até 100% e causar tempo de inatividade na disponibilidade.

- Conexões de cliente, para que você possa monitorar todos os picos inexplicáveis em conexões com o cluster.

Registrar operações do DAX usando o AWS CloudTrail

O Amazon DynamoDB Accelerator (DAX) é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações tomadas por um usuário, uma função ou um serviço da AWS no DAX.

Para saber mais sobre o DAX e o CloudTrail, consulte a seção do DynamoDB Accelerator (DAX) emRegistrar em log as operações do DynamoDB usando o AWS CloudTrail.

Instâncias expansíveis T3/T2 do DAX

O DAX permite que você escolha entre instâncias de desempenho fixo (como R4, R5 e R7) e instâncias de desempenho expansível (como T2 e T3). As instâncias expansíveis oferecem um nível básico de performance de CPU com capacidade de intermitência acima da referência.

A performance de referência e a capacidade de intermitência são governados por créditos de CPU. As instâncias expansíveis acumulam créditos de CPU continuamente a uma taxa determinada pelo tamanho da instância quando a workload está abaixo do limite de linha de base. Esses créditos podem então ser consumidos quando a workload aumenta. Um crédito de CPU oferece a performance de um núcleo de CPU completo por um minuto.

Muitas workloads não precisam de níveis consistentemente altos de CPU, mas se beneficiam significativamente de ter acesso total a CPUs muito rápidas quando precisam delas. As instâncias expansíveis são projetadas especificamente para esses casos de uso. Se você precisar de alta performance de CPU consistentemente para seu banco de dados, recomendamos usar instâncias de performance fixa.

Família de instâncias T2 do DAX

As instâncias T2 do DAX são instâncias expansíveis de uso geral que oferecem um nível básico de performance de CPU com capacidade de intermitência acima da referência. As instâncias T2 são uma boa opção workloads de teste e desenvolvimento que precisam de previsibilidade de preços. As instâncias do T2 do DAX são configuradas para o modo padrão, o que significa que se a instância estiver ficando sem créditos acumulados, a utilização da CPU será gradualmente reduzida para o nível de referência. Para obter mais informações sobre o modo padrão, consulte Modo padrão de instâncias expansíveis no Guia do usuário do Amazon EC2.

Família de instâncias T3 do DAX

As instâncias T3 do DAX são instâncias expansíveis de uso geral da próxima geração que oferecem um nível básico de performance de CPU com a capacidade de expandir o uso da CPU a qualquer momento e pelo tempo que for necessário. As instâncias T3 oferecem recursos equilibrados de computação, memória e rede e são ideais para workloads com uso moderado de CPU que experimentam picos temporários de uso. As instâncias T3 do DAX são configuradas para o modo ilimitado, o que significa que elas podem se expandir além da referência em uma janela de 24 horas por um custo adicional. Para obter mais informações sobre o modo ilimitado, consulte Modo ilimitado de instâncias expansíveis no Guia do usuário do Amazon EC2.

As instâncias T3 do DAX podem sustentar alta performance da CPU enquanto uma workload exigir. Na grande maioria das workloads de uso geral, as instâncias T3 fornecem ampla performance sem encargos adicionais. O preço por hora da instância T3 cobre automaticamente todos os picos temporários de uso quando a utilização média de CPU de uma instância T3 é menor ou igual à referência em uma janela de 24 horas.

Por exemplo, uma instância `dax.t3.small` recebe créditos continuamente a uma taxa de 24 créditos de CPU por hora. Esse recurso fornece performance básica equivalente a 20% de um núcleo de CPU (20% × 60 minutos = 12 minutos). Se a instância não usar os créditos recebidos, eles serão armazenados em seu saldo de créditos de CPU até um máximo de 576 créditos de CPU. Quando a instância `t3.small` precisar utilizar mais de 20% de um núcleo, ela deduzirá do seu saldo de créditos de CPU para lidar automaticamente com esse aumento.

Embora as instâncias T2 do DAX sejam restritas à performance básica, uma vez que o saldo de créditos da CPU é zerado, as instâncias T3 do DAX podem utilizar uma capacidade da referência mesmo quando o saldo de créditos da CPU é zero. Para a grande maioria das workloads, onde a utilização média de CPU é igual ou inferior à performance básica, o preço por hora básico para `t3.small` cobre todas as expansões de CPU. Se a instância for executada com uma utilização média de 25% da CPU (5% acima da referência) durante um período de 24 horas após o saldo de crédito da CPU ser zerado, um adicional de 11,52 centavos (9,6 centavos/VCPU-hora × 1 vCPU × 5% × 24 horas) será cobrado. Para obter detalhes de preço, consulte Preços do Amazon DynamoDB.

Controle de acesso do DAX

O DynamoDB Accelerator (DAX) foi projetado para funcionar em conjunto com o DynamoDB para adicionar de forma imperceptível uma camada de cache às suas aplicações. No entanto, o DAX e o DynamoDB têm mecanismos de controle de acesso separados. Embora esses serviços usem o AWS Identity and Access Management (IAM) para implementar suas respectivas políticas de segurança, os modelos de segurança para o DAX e o DynamoDB são diferentes.

É altamente recomendável entender ambos os modelos de segurança para que você possa implementar as medidas de segurança adequadas às suas aplicações que usam DAX.

Esta seção descreve os mecanismos de controle de acesso fornecidos pelo DAX e fornece exemplos de políticas do IAM que você pode adaptar às suas necessidades.

Com o DynamoDB, você pode criar políticas do IAM que limitam as ações que um usuário pode realizar em recursos individuais do DynamoDB. Por exemplo, você pode criar uma função de usuário

que permite que o usuário execute apenas ações somente leitura em uma determinada tabela do DynamoDB. (Para ter mais informações, consulte Gerenciamento de identidade e acesso no Amazon DynamoDB.) Por comparação, o modelo de segurança do DAX se concentra na segurança do cluster e na habilidade do cluster de realizar ações de API do DynamoDB em seu nome.

> **Warning**
>
> Se, no momento, você estiver usando funções e políticas do IAM para restringir o acesso aos dados de tabelas do DynamoDB, então, o uso do DAX pode subverter essas políticas. Por exemplo, um usuário poderia ter acesso a uma tabela do DynamoDB via DAX, mas não ter acesso explícito à mesma tabela acessando o DynamoDB diretamente. Para obter mais informações, consulte Gerenciamento de identidade e acesso no Amazon DynamoDB.
> O DAX não impõe separação em nível do usuário em dados no DynamoDB. Em vez disso, os usuários herdam as permissões da política do IAM do cluster do DAX quando acessam o cluster. Sendo assim, ao acessar as tabelas do DynamoDB via DAX, os únicos controles de acesso que estão em vigor são as permissões da política do IAM do cluster do DAX. Nenhuma outra permissão é reconhecida.
> Se você precisar de isolamento, recomendamos criar clusters do DAX adicionais e definir o escopo da política do IAM para cada cluster conforme necessário. Por exemplo, é possível criar vários clusters do DAX e permitir que cada cluster acesse apenas uma única tabela.

Perfil de serviço do IAM para o DAX

Ao criar um cluster do DAX, você deve associar o cluster a uma função do IAM. Isso é conhecido como função de serviço do cluster.

Suponha que você queria criar um novo cluster do DAX chamado DAXCluster01. Você poderia criar uma função de serviço chamada DAXServiceRole e associar a função ao DAXCluster01. A política para DAXServiceRole definiria as ações do DynamoDB que o DAXCluster01 poderia realizar, em nome dos usuários que interagem com o DAXCluster01.

Ao criar uma função de serviço, você deve especificar uma relação de confiança entre DAXServiceRole e o serviço DAX em si. Uma relação de confiança determina quais entidades podem assumir uma função e usar suas permissões. A seguir há um exemplo de documento de relação de confiança de DAXServiceRole:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Essa relação de confiança permite que um cluster do DAX assuma DAXServiceRole e realize chamadas à API do DynamoDB em seu nome.

As ações da API do DynamoDB permitidas são descritas em um documento da política do IAM que você anexa à função de serviço DAXServiceRole. Veja abaixo um exemplo de documento de política.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DaxAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
```

```
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
            ]
        }
    ]
}
```

Essa política permite que o DAX execute todas as ações da API do DynamoDB em uma tabela do DynamoDB. A ação `dynamodb:DescribeTable` é necessária para o DAX manter metadados sobre a tabela, e as outras são ações de leitura e gravação executadas em itens na tabela. A tabela chamada `Books` está na região us-west-2 e pertence ao ID `123456789012` da conta da AWS.

> **Note**
>
> O DAX comporta mecanismos para evitar o problema de representante confuso durante o acesso entre serviços. Para obter mais informações, consulte O problema confused deputy no Guia do usuário IAM.

Política do IAM para permitir acesso a cluster do DAX

Depois de criar um cluster do DAX, será necessário conceder permissões a um usuário para que ele possa acessar o cluster do DAX.

Por exemplo, suponha que você queira conceder acesso ao DAXCluster01 para uma usuária chamada Alice. Você primeiro criaria uma política do IAM (AliceAccessPolicy) que define os clusters do DAX e as ações da API do DAX que o destinatário pode acessar. Em seguida, você deveria conceder acesso anexando essa política à usuária Alice.

O documento de política a seguir oferece acesso total ao destinatário no DAXCluster01.

JSON

```
{
    "Version": "2012-10-17"      ,
    "Statement": [
        {
```

```
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

O documento de política permite o acesso ao cluster do DAX, mas não concede quaisquer permissões do DynamoDB. (As permissões do DynamoDB são atribuídas pela função de serviço do DAX.)

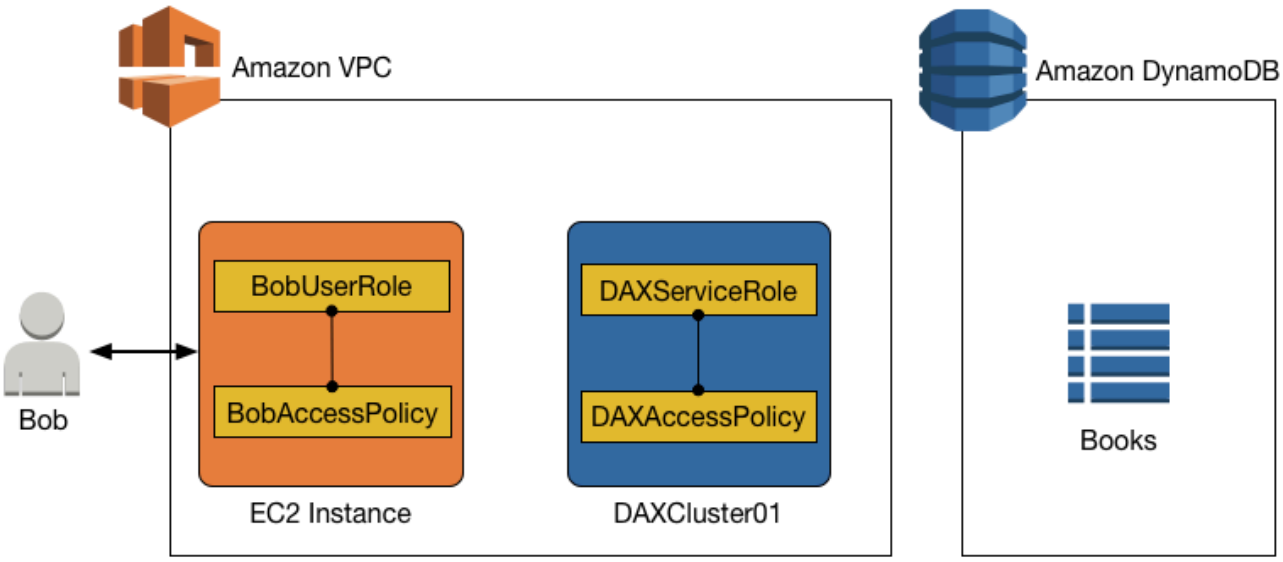
Para a usuária Alice, você deve primeiro criar `AliceAccessPolicy` com o documento de política mostrado anteriormente. Em seguida, você anexaria a política à Alice.

> **Note**
>
> Em vez de anexar a política a um usuário, você pode anexá-la a um perfil do IAM. Desse modo, todos os usuários que assumem essa função teriam as permissões que você definiu na política.

A política de usuário, junto com a função de serviço do DAX, determina os recursos e as ações de API do DynamoDB que o destinatário pode acessar via DynamoDB.

Estudo de caso: acesso ao DynamoDB e ao DAX

O cenário a seguir pode ajudar a conhecer melhor as políticas do IAM para uso com o DAX. (Vamos falar sobre este cenário em todo o resto desta seção.) O diagrama a seguir mostra uma visão geral de alto nível do cenário.

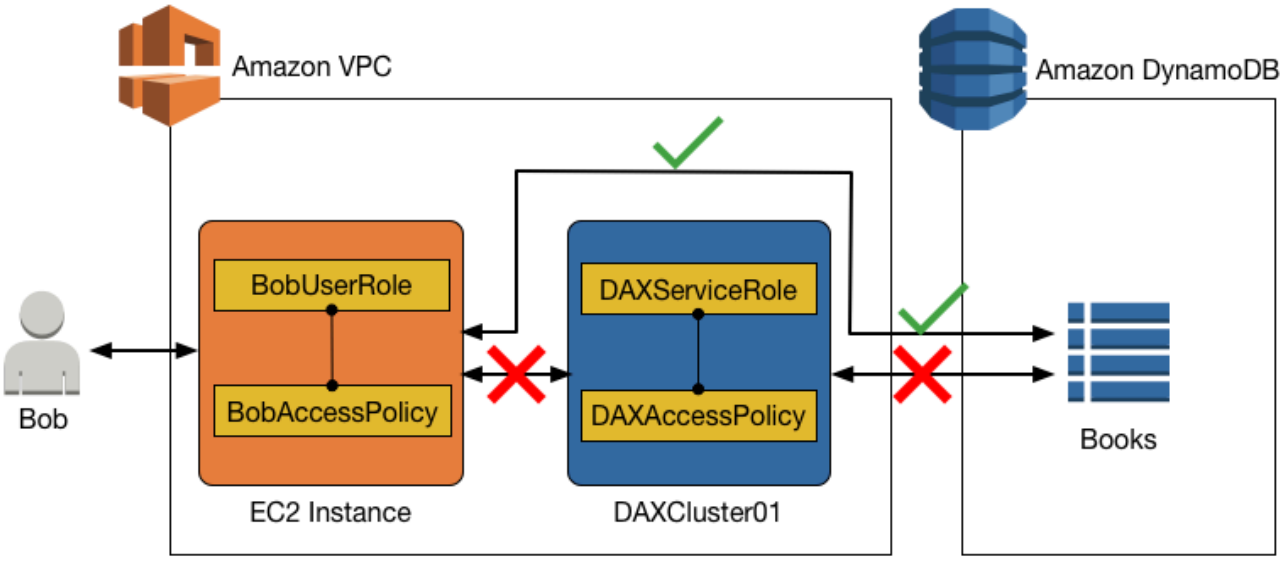


Nesse cenário, há as seguintes entidades:

- Um usuário (Bob).
- Um perfil do IAM (`BobUserRole`). Bob assume essa função no tempo de execução.
- Uma política do IAM (`BobAccessPolicy`). Essa política está anexada ao `BobUserRole`. O `BobAccessPolicy` define os recursos do DynamoDB e do DAX que o `BobUserRole` tem permissão para acessar.
- Um cluster do DAX (`DAXCluster01`).
- Uma função de serviço do IAM (`DAXServiceRole`). Essa função permite que o `DAXCluster01` acesse o DynamoDB.
- Uma política do IAM (`DAXAccessPolicy`). Essa política está anexada ao `DAXServiceRole`. O `DAXAccessPolicy` define as APIS e os recursos do DynamoDB que o `DAXCluster01` têm permissão para acessar.
- Uma tabela do DynamoDB (`Books`)

A combinação de elementos de política no `BobAccessPolicy` e `DAXAccessPolicy` determinam o que Bob pode fazer com a tabela `Books`. Por exemplo, Bob poderia conseguir acessar `Books` diretamente (usando o endpoint do DynamoDB), indiretamente (usando o cluster do DAX) ou ambos. Bob talvez também possa ler os dados de `Books`, gravar dados em `Books`, ou as duas coisas.

Acesso ao DynamoDB, mas sem acesso com o DAX



É possível permitir acesso direto a uma tabela do DynamoDB e, ao mesmo tempo, evitar o acesso indireto usando um cluster do DAX. Para acesso direto ao DynamoDB, as permissões do `BobUserRole` são determinadas pela `BobAccessPolicy` (que está anexada à função).

Acesso somente leitura ao DynamoDB (apenas)

Bob pode acessar o DynamoDB com `BobUserRole`. A política do IAM anexada a essa função (`BobAccessPolicy`) determina as tabelas do DynamoDB que `BobUserRole` pode acessar e quais APIs `BobUserRole` pode invocar.

Considere o documento de política a seguir para `BobAccessPolicy`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
```

```
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Quando esse documento está anexado a `BobAccessPolicy`, ele permite que `BobUserRole` acesse o endpoint do DynamoDB e execute operações somente leitura na tabela `Books`.

O DAX não aparece nessa política. Logo, o acesso via DAX é negado.

Acesso de leitura/gravação ao DynamoDB (apenas)

Se `BobUserRole` precisar de acesso de leitura/gravação ao DynamoDB, a política a seguir funcionará.

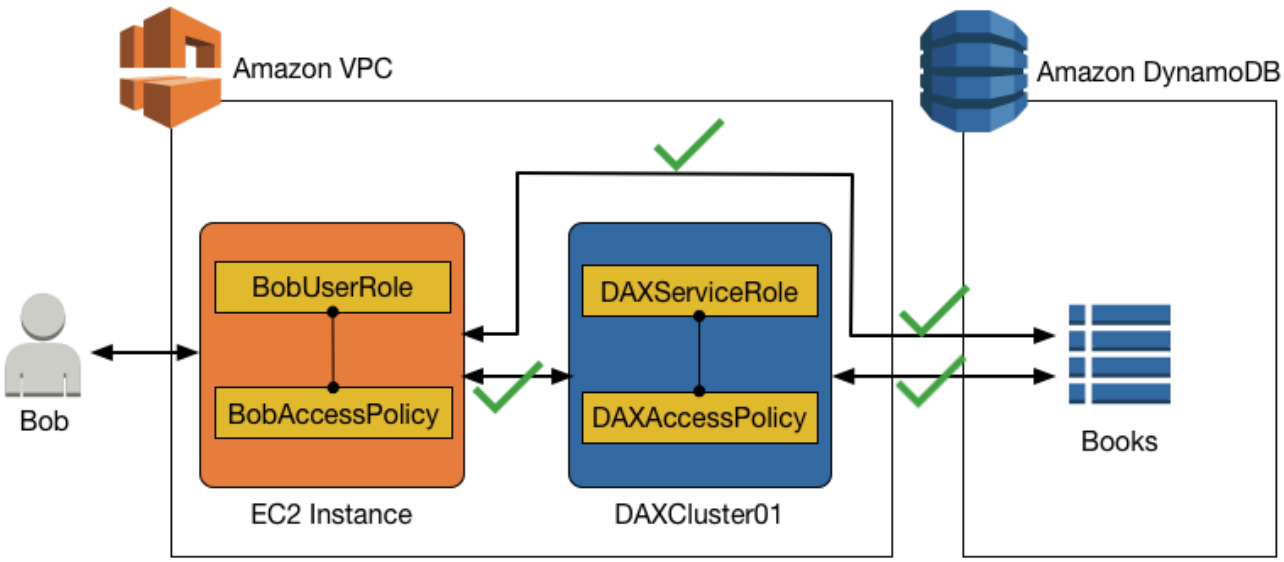
JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
```

```
    ]
}
```

Novamente, o DAX não aparece nessa política, portanto, o acesso via DAX é negado.

Acesso ao DynamoDB e ao DAX



Para permitir acesso a um cluster do DAX, você deve incluir ações específicas do DAX em uma política do IAM.

As ações específicas do DAX a seguir correspondem à suas equivalentes de nome semelhante na API do DynamoDB:

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax:DeleteItem`

- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

O mesmo é válido para a chave de condição `dax:EnclosingOperation`.

Acesso somente leitura ao DynamoDB e acesso somente leitura ao DAX

Suponha que Bob requeira acesso somente de leitura à tabela `Books`, tanto via DynamoDB quanto via DAX. A política a seguir (anexada a `BobUserRole`) confere esse acesso.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

A política tem uma instrução para acesso ao DAX (`DAXAccessStmt`) e outra instrução para o acesso ao DynamoDB (`DynamoDBAccessStmt`). Essas declarações permitem que Bob envie as solicitações `GetItem`, `BatchGetItem`, `Query` e `Scan` ao `DAXCluster01`.

No entanto, a função de serviço de `DAXCluster01` também exige acesso somente de leitura à tabela `Books` no DynamoDB. A política do IAM a seguir anexada à `DAXServiceRole` cumpriria essa exigência.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Acesso de leitura/gravação ao DynamoDB e somente de leitura com o DAX

Para uma determinada função de usuário, você pode fornecer acesso de leitura e gravação a uma tabela do DynamoDB e também permitir acesso somente de leitura via DAX.

Para Bob, a política do IAM para `BobUserRole` precisaria permitir ações de leitura e gravação ao DynamoDB na tabela `Books`, e ao mesmo tempo aceitar ações somente de leitura via `DAXCluster01`.

O seguinte exemplo de documento de política para `BobUserRole` conferiria esse acesso.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Além disso, `DAXServiceRole` exigiria uma política do IAM que permite que `DAXCluster01` realize ações somente de leitura na tabela `Books`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:DescribeTable"
           ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Acesso de leitura e gravação ao DynamoDB e acesso de leitura e gravação ao DAX

Agora, suponha que Bob precisasse de acesso de leitura/gravação à tabela `Books` diretamente do DynamoDB ou indiretamente do `DAXCluster01`. O seguinte documento de política anexado a `BobAccessPolicy` confere esse acesso.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
```

```
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Além disso, `DAXServiceRole` exigiria uma política do IAM que permitisse que `DAXCluster01` realizasse ações de leitura/gravação na tabela `Books`.

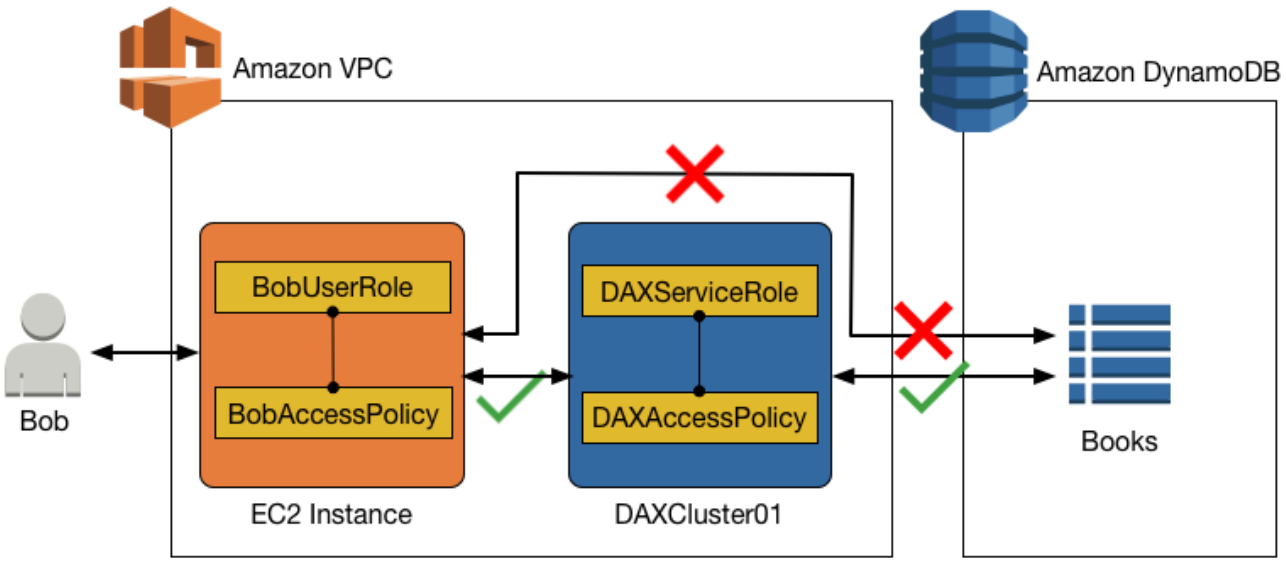
JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
```

```
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Acesso ao DynamoDB via DAX, mas sem acesso direto ao DynamoDB

Nesse cenário, Bob pode acessar a tabela `Books` via DAX, mas não tem acesso direto à tabela `Books` no DynamoDB. Portanto, quando Bob recebe acesso ao DAX, ele também obtém acesso a uma tabela do DynamoDB que, de outra forma, ele não poderia acessar. Ao configurar uma política do IAM para a função de serviço do DAX, lembre-se de que qualquer usuário que recebe acesso ao cluster do DAX por meio da política de acesso do usuário tem acesso às tabelas especificadas nessa política. Nesse caso, `BobAccessPolicy` ganha acesso às tabelas especificadas na `DAXAccessPolicy`.



Se você estiver usando funções e políticas do IAM para restringir o acesso aos dados e tabelas do DynamoDB, o uso do DAX pode subverter essas políticas. Na política a seguir, Bob tem acesso a uma tabela do DynamoDB via DAX mas não tem acesso direto explícito à mesma tabela no DynamoDB.

O documento de política a seguir (`BobAccessPolicy`) anexado a `BobUserRole` conferiria esse acesso.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        }
    ]
}
```

Nessa política de acesso, não há permissões para acessar o DynamoDB diretamente.

Junto com `BobAccessPolicy`, a `DAXAccessPolicy` a seguir confere ao `BobUserRole` o acesso à tabela `Books` do DynamoDB mesmo que `BobUserRole` não possa acessar diretamente a tabela `Books`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Como o exemplo mostra, ao configurar o controle de acesso para a política de acesso do usuário e a política de acesso do cluster do DAX, você tem que compreender totalmente o acesso de ponta a ponta para garantir que o princípio de privilégio mínimo seja observado. Garanta também que conceder acesso a um cluster do DAX para um usuário não subverte as políticas de controle de acesso estabelecidas anteriormente.

Criptografia em repouso do DAX

A criptografia em repouso do Amazon DynamoDB Accelerator (DAX) fornece uma camada adicional de proteção de dados, ajudando a proteger os dados contra acesso não autorizado ao armazenamento subjacente. O uso de criptografia em repouso para proteção de dados pode ser requerida por políticas organizacionais, regulamentações setoriais ou governamentais e exigências de conformidade. Você pode usar criptografia para aumentar a segurança dos dados dos aplicativos que são implantados na nuvem.

Com a criptografia em repouso, os dados persistentes do DAX em disco são criptografados usando Advanced Encryption Standard de 256 bits, também conhecida como AES-256. O DAX grava dados ao disco como parte das alterações de propagação do nó primário para as réplicas de leitura.

A criptografia em repouso do DAX integra-se automaticamente ao AWS Key Management Service (AWS KMS) para gerenciar a chave única de serviço padrão usada para criptografar seus clusters. Se uma chave de serviço padrão não existir quando você criar seu cluster do DAX criptografado, o AWS KMS criará automaticamente uma nova chave gerenciada pela AWS para você. Essa chave é usada com clusters criptografados que são criados no futuro. O AWS KMS integra hardware e software seguros e altamente disponíveis para oferecer um sistema de gerenciamento de chaves escalonado para a nuvem.

Assim que seus dados são criptografadas, o DAX lida de forma transparente com a descriptografia dos dados com um impacto mínimo sobre a performance. Você não precisa modificar seus aplicativos para usar a criptografia.

> **Note**
>
> O DAX não chama o AWS KMS para cada operação própria. O DAX usa a chave somente ao iniciar o cluster. Mesmo que o acesso seja revogado, o DAX ainda poderá acessar os dados até que o cluster seja desativado. As chaves AWS KMS especificadas pelo cliente não são aceitas.

A criptografia em repouso do DAX está disponível para os seguintes tipos de nó de cluster:

Família	Tipo de nó
Otimizado para memória (R4, R5 e R7)	dax.r4.large
	dax.r4.xlarge
	dax.r4.2xlarge
	dax.r4.4xlarge
	dax.r4.8xlarge
	dax.r4.16xlarge

Família	Tipo de nó
	dax.r5.large dax.r5.xlarge dax.r5.2xlarge dax.r5.4xlarge dax.r5.8xlarge dax.r5.12xlarge dax.r5.16xlarge dax.r5.24xlarge dax.r7i.large dax.r7i.xlarge dax.r7i.2xlarge dax.r7i.4xlarge dax.r7i.8xlarge dax.r7i.12xlarge dax.r7i.16xlarge dax.r7i.24xlarge
Uso geral (T2)	dax.t2.small dax.t2.medium
Uso geral (T3)	dax.t3.small dax.t3.medium

> **Important**
>
> A criptografia em repouso do DAX não é compatível com tipos de nós `dax.r3.*`.

Não é possível ativar ou desativar a criptografia em repouso após a criação e um cluster. Você deve recriar o cluster para ativar a criptografia em repouso caso não tenha sido ativada na criação.

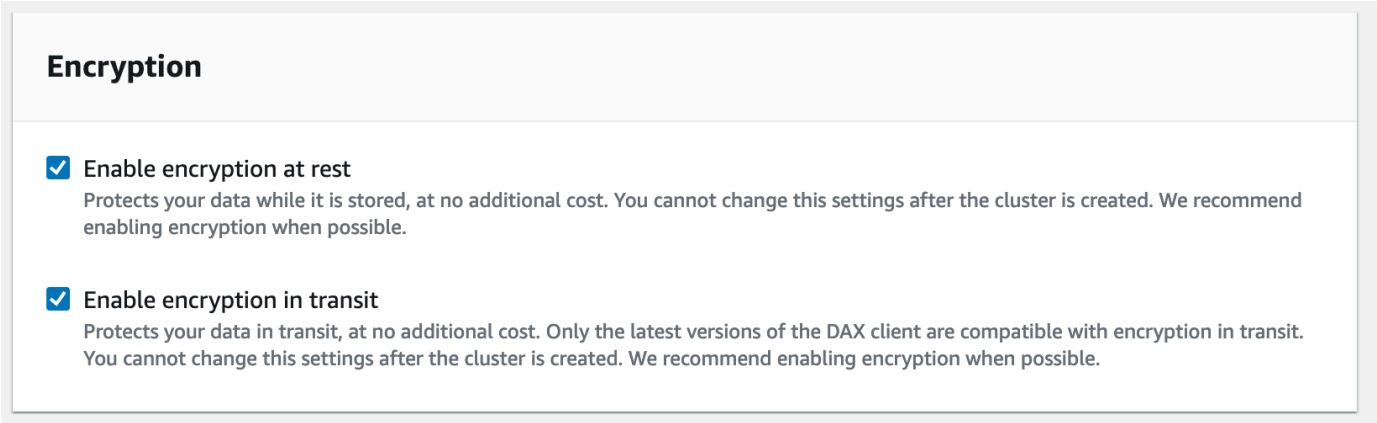
A criptografia em repouso do DAX é fornecida sem custo adicional (cobranças de uso da chave de criptografia AWS KMS se aplicam). Para obter informações sobre preços, consulte Preços do Amazon DynamoDB.

Habilitar a criptografia em repouso usando o AWS Management Console

Siga estas etapas para habilitar a criptografia em repouso do DAX em uma tabela usando o console.

Para habilitar a criptografia em repouso do DAX

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação no lado esquerdo do console, em DAX, selecione Clusters.
3. Selecione Criar cluster.
4. Em Cluster name (Nome do cluster), insira um nome curto para o seu cluster. Selecione o node type (tipo de nó) para todos os nós no cluster e, para o tamanho do cluster, use **3** nós.
5. Em Encryption (Criptografia), selecione Enable encryption (Habilitar criptografia).



6. Após selecionar a função do IAM, o grupo de sub-rede, os grupos de segurança e as configurações do cluster, selecione Launch cluster (Iniciar cluster).

Para confirmar se o cluster está criptografado, verifique os detalhes dele no painel Clusters. O status da criptografia deve ser ENABLED (ATIVADA).

Criptografia em trânsito do DAX

O Amazon DynamoDB Accelerator (DAX) oferece suporte à criptografia em trânsito de dados entre sua aplicação e seu cluster do DAX, permitindo a você usar o DAX em aplicações com requisitos rigorosos de criptografia.

Independentemente de você escolher ou não a criptografia em trânsito, o tráfego entre a aplicação e o cluster do DAX permanece em sua Amazon VPC. Esse tráfego é roteado para interfaces de rede elásticas com IPs privados em sua VPC que estão conectados aos nós do cluster. Com sua VPC como limite de confiança, você tem controle significativo sobre a segurança de seus dados por meio do uso de ferramentas padrão, como grupos de segurança, segmentação de sub-rede com ACLs de rede e rastreamento de fluxo de VPC. A criptografia em trânsito do DAX aumenta esse nível de confidencialidade de referência, garantindo que todas as solicitações e respostas entre a aplicação e o cluster sejam criptografadas por TLS (Transport Level Security) e que as conexões com o cluster possam ser autenticadas pela verificação de um certificado x509 de cluster. Os dados gravados em disco pelo DAX também podem ser criptografados se você escolher criptografia em repouso ao criar o cluster do DAX.

Usar criptografia em trânsito com DAX é fácil. Basta selecionar esta opção ao criar um novo cluster e usar uma versão recente de qualquer um dosclientes do DAX em sua aplicação. Os clusters que usam criptografia em trânsito não oferecem suporte a tráfego não criptografado, portanto, não há chance de configurar incorretamente sua aplicação e ignorar a criptografia. O cliente do DAX usará o certificado x509 do cluster para autenticar a identidade do cluster ao estabelecer as conexões, garantindo que suas solicitações do DAX vão para o destino pretendido. Todos os métodos de criação de clusters do DAX oferecem suporte à criptografia em trânsito: AWS Management Console, AWS CLI, todos os SDKs e AWS CloudFormation.

A criptografia em trânsito não pode ser habilitada em um cluster do DAX existente. Para usar criptografia em trânsito em uma aplicação do DAX existente, crie um novo cluster com criptografia em trânsito habilitada, mude o tráfego da aplicação para ele e exclua o cluster antigo.

Usar funções vinculadas ao serviço do IAM para o DAX

O Amazon DynamoDB Accelerator (DAX) usa funções vinculadas ao serviço do AWS Identity and Access Management (IAM). A função vinculada ao serviço é um tipo exclusivo de função do IAM

ligada diretamente ao DAX. As funções vinculadas a serviços são predefinidas pelo DAX e incluem todas as permissões que o serviço requer para chamar outros serviços da AWS em seu nome.

Uma função vinculada ao serviço facilita a configuração do DAX porque você não precisa adicionar as permissões necessárias manualmente. O DAX define as permissões das funções vinculadas ao serviço e, exceto se definido de outra forma, somente o DAX pode assumir suas funções. As permissões definidas incluem a política de confiança e a política de permissões. Essa política de permissões não pode ser anexada a nenhuma outra entidade do IAM.

É possível excluir as funções somente depois de primeiro excluir seus recursos relacionados. Isso protege seus recursos do DAX, pois você não pode remover por engano as permissões para acessar os recursos.

Para obter informações sobre outros serviços que oferecem suporte a funções vinculadas ao serviço, consulte Serviços da AWS compatíveis com o IAM no Guia do usuário do IAM. Procure os serviços que têm Yes (Sim) na coluna Service-linked roles (Funções vinculadas ao serviço). Selecione o link Yes (Sim) para exibir a documentação da função vinculada ao serviço para esse serviço.

Tópicos

- Permissões de função vinculada ao serviço para o DAX
- Criar uma função vinculada ao serviço do DAX
- Editar uma função vinculada ao serviço do DAX
- Excluir uma função vinculada ao serviço do DAX

Permissões de função vinculada ao serviço para o DAX

O DAX usa a função vinculada ao serviço chamada `AWSServiceRoleForDAX`. Essa função permite que o DAX chame serviços em nome do seu cluster do DAX.

> **Important**
>
> A função `AWSServiceRoleForDAX` vinculada ao serviço facilita a configuração e a manutenção de um cluster do DAX. No entanto, ainda é necessário conceder a cada cluster acesso ao DynamoDB para que você possa usá-lo. Para ter mais informações, consulte Controle de acesso do DAX.

A função vinculada ao serviço `AWSServiceRoleForDAX` confia nos seguintes serviços para assumir a função:

- `dax.amazonaws.com`

A política de permissões da função permite que o DAX conclua as seguintes ações nos recursos especificados:

- Ações em `ec2`:
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeNetworkInterfaces`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`
  - `DescribeVpcs`
  - `ModifyNetworkInterfaceAttribute`
  - `RevokeSecurityGroupIngress`

Você deve configurar permissões para que uma entidade do IAM (por exemplo, um usuário, grupo ou função) crie, edite ou exclua uma função vinculada a serviço. Para mais informações, consulte Permissões de perfil vinculado ao serviço no Guia do usuário do IAM.

Para permitir que uma entidade do IAM crie funções vinculadas ao serviço AWSServiceRoleForDAX

Adicione a seguinte declaração de política às permissões dessa entidade do IAM.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
```

```
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

Criar uma função vinculada ao serviço do DAX

Não é necessário criar manualmente uma função vinculada ao serviço. Quando você cria um cluster, o DAX cria uma função vinculada ao serviço para você.

> **Important**
>
> Se você usava o serviço DAX antes de 28 de fevereiro de 2018, quando ele começou a oferecer suporte às funções vinculadas ao serviço, o DAX criou a função `AWSServiceRoleForDAX` em sua conta. Para obter mais informações, consulte Uma nova função apareceu em minha conta da AWS no Guia do usuário do IAM.

Se você excluir essa função vinculada ao serviço e precisar criá-la novamente, poderá usar esse mesmo processo para recriar a função em sua conta. Quando você cria uma instância ou um cluster, o DAX cria uma função vinculada ao serviço para você novamente.

Editar uma função vinculada ao serviço do DAX

O DAX não permite editar a função vinculada ao serviço `AWSServiceRoleForDAX`. Depois que criar um perfil vinculado ao serviço, você não poderá alterar o nome do perfil, pois várias entidades podem fazer referência a ele. No entanto, será possível editar a descrição do perfil usando o IAM. Para obter mais informações, consulte Editar uma função vinculada a serviço no Guia do usuário do IAM.

Excluir uma função vinculada ao serviço do DAX

Se você não precisar mais usar um atributo ou serviço que requer uma função vinculada a serviço, é recomendável excluí-la. Dessa forma, você não tem uma entidade não utilizada que não seja monitorada ativamente ou mantida. No entanto, você deve excluir todos os seus clusters do DAX para poder excluir a função vinculada ao serviço.

Limpar uma função vinculada ao serviço

Antes de você poder usar o IAM para excluir uma função vinculada ao serviço, você deve primeiro confirmar que a função não tem sessões ativas e remover quaisquer recursos usados pela função.

Para verificar se a função vinculada ao serviço tem uma sessão ativa no console do IAM

1. Faça login em AWS Management Console e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. No painel de navegação do console do IAM, escolha Perfis. Em seguida, selecione o nome (não a caixa de seleção) da função AWSServiceRoleForDAX.
3. Na página Resumo para a função selecionada, escolha a guia Consultor de Acesso.
4. Na guia Consultor de Acesso, revise a atividade recente para a função vinculada ao serviço.

> **Note**
>
> Se não tiver certeza se o DAX está usando a função `AWSServiceRoleForDAX`, você poderá tentar excluir a função. Se o serviço estiver usando a função, a exclusão falhará, e você poderá visualizar as regiões em que a função está sendo usada. Se a função estiver sendo usada, você deverá excluir os clusters do DAX antes de excluir a função. Você não pode revogar a sessão de uma função vinculada ao serviço.

Para remover a função `AWSServiceRoleForDAX`, você deverá excluir primeiro todos os seus clusters do DAX.

Excluir todos os clusters do DAX

Use um destes procedimentos para excluir cada um de seus clusters do DAX.

Para excluir um cluster do DAX (console)

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, em DAX, escolha Clusters.
3. Escolha Ações e, em seguida, escolha Excluir.
4. Na caixa de diálogo Delete cluster confirmation (Confirmar exclusão de cluster), escolha Delete (Excluir).

Para excluir um cluster do DAX (AWS CLI)

Consulte delete-cluster na Referência de comandos da AWS CLI.

Para excluir um cluster do DAX (API)

Consulte DeleteCluster na Referência da API do Amazon DynamoDB.

Excluir uma função vinculada ao serviço

Como excluir manualmente a função vinculada a serviço usando o IAM

Use o console, a CLI ou a API do IAM para excluir a função vinculada ao serviço `AWSServiceRoleForDAX`. Para mais informações, consulte Excluir um perfil vinculado ao serviço no Guia do usuário do IAM.

Acessando o DAX em contas da AWS

Suponha que você tenha um cluster do DynamoDB Accelerator (DAX) em execução em uma conta da AWS (conta A) e que o cluster do DAX precise ser acessível de uma instância do Amazon Elastic Compute Cloud (Amazon EC2) em outra conta da AWS (conta B). Neste tutorial, isso é feito iniciando uma instância do EC2 na conta B com uma função do IAM da conta B. Então, você usa credenciais de segurança temporárias da instância do EC2 para assumir uma função do IAM da conta A. Por fim, você usa as credenciais de segurança temporárias da função do IAM da conta A para fazer chamadas de aplicação em uma conexão de emparelhamento da Amazon VPC para o cluster do DAX na conta A. Para realizar essas tarefas, é preciso ter acesso administrativo em ambas as contas da AWS.

> **Important**
>
> Não é possível que um cluster do DAX acesse uma tabela do DynamoDB de uma conta diferente.

Tópicos

- Configurar o IAM
- Configurar uma VPC
- Modificar o cliente do DAX para permitir acesso entre contas

Configurar o IAM

1. Crie um arquivo de texto chamado `AssumeDaxRoleTrust.json` com o conteúdo a seguir, que permite que o Amazon EC2 trabalhe em seu nome.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Na conta B, crie uma função que o Amazon EC2 possa usar ao executar instâncias.

```
aws iam create-role \
    --role-name AssumeDaxRole \
    --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. Crie um arquivo de texto chamado `AssumeDaxRolePolicy.json` com o seguinte conteúdo, que permite que o código executado na instância do EC2 na conta B assuma uma função do IAM na conta A. Substitua *`accountA`* pelo ID real da conta A.

   JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::111122223333:role/DaxCrossAccountRole"
        }
    ]
}
```

4. Adicione essa política à função que acabou de criar.

```
aws iam put-role-policy \
    --role-name AssumeDaxRole \
    --policy-name AssumeDaxRolePolicy \
    --policy-document file://AssumeDaxRolePolicy.json
```

5. Crie um perfil de instância para permitir que as instâncias usem a função.

```
aws iam create-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile
```

6. Associe a função ao perfil de instância.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile \
    --role-name AssumeDaxRole
```

7. Crie um arquivo de texto chamado `DaxCrossAccountRoleTrust.json` com o seguinte conteúdo, que permite que a conta B assuma uma função da conta A. Substitua *`accountB`* pelo ID real da conta B.

   JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. Na conta A, crie a função que a conta B poderá assumir.

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. Crie um arquivo de texto chamado `DaxCrossAccountPolicy.json` que permite acesso ao cluster do DAX. Substitua *`dax-cluster-arn`* pelo nome do recurso da Amazon (ARN) correto do cluster do DAX.

   JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-east-1:111122223333:cache/dax-
cluster-name"
        }
    ]
}
```

10. Na conta A, adicione a política à função.

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

Configurar uma VPC

1. Localize o grupo de sub-redes do cluster do DAX da conta A. Substitua *`cluster-name`* pelo nome do cluster do DAX que a conta B deve acessar.

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. Usando o *subnet-group*, encontre a VPC do cluster.

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. Usando o *vpc-id*, encontre o CIDR da VPC.

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. Na conta B, crie uma VPC usando um CIDR não sobreposto diferente daquele encontrado na etapa anterior. Em seguida, crie pelo menos uma sub-rede. Você pode usar o assistente de criação da VPC no AWS Management Console ou a AWS CLI.
5. Na conta B, solicite uma conexão de emparelhamento com a VPC da conta A conforme descrito em Criar e aceitar uma conexão de emparelhamento da VPC. Na conta A, aceite a conexão.
6. Na conta B, localize a nova tabela de roteamento da VPC. Substitua *vpc-id* pelo ID da VPC que você criou na conta B.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. Adicione uma rota para enviar o tráfego destinado ao CIDR da conta A para a conexão de emparelhamento da VPC. Lembre-se de substituir cada *espaço reservado de entrada de usuário* pelos valores corretos para suas contas.

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. Na conta A, localize a tabela de rotas do cluster do DAX usando o *vpc-id* encontrado anteriormente.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. Na conta A, adicione uma rota para enviar o tráfego destinado ao CIDR da conta B para a conexão de emparelhamento da VPC. Substitua cada *espaço reservado de entrada de usuário* pelos valores corretos para suas contas.

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. Na conta B, execute uma instância do EC2 na VPC que você criou anteriormente. Forneça o `AssumeDaxInstanceProfile`. Você pode usar o assistente de inicialização no AWS Management Console ou a AWS CLI. Anote o grupo de segurança da instância.

11. Na conta A, localize o grupo de segurança usado pelo cluster do DAX. Lembre-se de substituir *cluster-name* pelo nome do cluster do DAX.

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. Atualize o grupo de segurança do cluster do DAX para permitir o tráfego de entrada do grupo de segurança da instância do EC2 que você criou na conta B. Lembre-se de substituir os *espaços reservados de entrada do usuário* pelos valores corretos para suas contas.

```
aws ec2 authorize-security-group-ingress \
    --group-id accountA-security-group-id \
    --protocol tcp \
    --port 8111 \
    --source-group accountB-security-group-id \
    --group-owner accountB-id
```

Nesse ponto, uma aplicação na instância do EC2 da conta B pode usar o perfil de instância para assumir a função `arn:aws:iam::`*accountA-id*`:role/DaxCrossAccountRole` e usar o cluster do DAX.

Modificar o cliente do DAX para permitir acesso entre contas

> **Note**
>
> As credenciais do AWS Security Token Service (AWS STS) são temporárias. Alguns clientes lidam com a atualização automaticamente, enquanto outros exigem lógica adicional para atualizar as credenciais. Recomendamos seguir as orientações da documentação apropriada.

Java

Esta seção ajuda você a modificar seu código de cliente do DAX existente para permitir o acesso ao DAX entre contas. Caso ainda não tenha um código de cliente do DAX, você pode encontrar exemplos de código que funcionam no tutorial do Java e DAX.

1. Adicione as seguintes importações.

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. Obtenha um provedor de credenciais do AWS STS e crie um objeto de cliente do DAX. Lembre-se de substituir cada *espaço reservado de entrada de usuário* pelos valores corretos para suas contas.

```
AWSSecurityTokenService awsSecurityTokenService =
 AWSSecurityTokenServiceClientBuilder
     .standard()
     .withRegion(region)
     .build();

STSAssumeRoleSessionCredentialsProvider credentials =  new
 STSAssumeRoleSessionCredentialsProvider.Builder("arn:aws:iam::accountA:role/
RoleName", "TryDax")
     .withStsClient(awsSecurityTokenService)
     .build();

DynamoDB client = AmazonDaxClientBuilder.standard()
    .withRegion(region)
```

```
    .withEndpointConfiguration(dax_endpoint)
    .withCredentials(credentials)
    .build();
```

.NET

Esta seção ajuda você a modificar seu código de cliente do DAX existente para permitir o acesso ao DAX entre contas. Caso ainda não tenha um código de cliente do DAX, você pode encontrar exemplos de código que funcionam no tutorial do .NET e DAX.

1. Adicione o pacote AWSSDK.SecurityToken NuGet à solução.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. Use os pacotes `SecurityToken` e `SecurityToken.Model`.

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. Obtenha credenciais temporárias do `AmazonSimpleTokenService` e crie um objeto `ClusterDaxClient`. Lembre-se de substituir cada *espaço reservado de entrada de usuário* pelos valores corretos para suas contas.

```
IAmazonSecurityTokenService sts = new AmazonSecurityTokenServiceClient();

var assumeRoleResponse = sts.AssumeRole(new AssumeRoleRequest
{
    RoleArn = "arn:aws:iam::accountA:role/RoleName",
                RoleSessionName = "TryDax"
});

Credentials credentials = assumeRoleResponse.Credentials;

var clientConfig = new DaxClientConfig(dax_endpoint, port)
{
    AwsCredentials = assumeRoleResponse.Credentials

};

var client = new ClusterDaxClient(clientConfig);
```

Go

Esta seção ajuda você a modificar seu código de cliente do DAX existente para permitir o acesso ao DAX entre contas. Caso ainda não tenha um código de cliente do DAX, você poderá encontrar exemplos de código funcionais no GitHub.

1. Importe os pacotes de AWS STS e sessão.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sts"
    "github.com/aws/aws-sdk-go/aws/credentials/stscreds"
)
```

2. Obtenha credenciais temporárias do `AmazonSimpleTokenService` e crie um objeto de cliente do DAX. Lembre-se de substituir cada *`espaço reservado de entrada de usuário`* pelos valores corretos para suas contas.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String(region)},
)
if err != nil {
    return nil, err
}

stsClient := sts.New(sess)
arp := &stscreds.AssumeRoleProvider{
                Duration:     900 * time.Second,
                ExpiryWindow: 10 * time.Second,
                RoleARN:      "arn:aws:iam::accountA:role/role_name",
                Client:       stsClient,
                RoleSessionName: "session_name",
        }cfg := dax.DefaultConfig()

cfg.HostPorts = []string{dax_endpoint}
cfg.Region = region
cfg.Credentials = credentials.NewCredentials(arp)
daxClient := dax.New(cfg)
```

Python

Esta seção ajuda você a modificar seu código de cliente do DAX existente para permitir o acesso ao DAX entre contas. Caso ainda não tenha um código de cliente do DAX, você pode encontrar exemplos de código que funcionam no tutorial do Python e DAX.

1. Importe o `boto3`.

```
import boto3
```

2. Obtenha credenciais temporárias do `sts` e crie um objeto do `AmazonDaxClient`. Lembre-se de substituir cada *espaço reservado de entrada de usuário* pelos valores corretos para suas contas.

```
sts = boto3.client('sts')
stsresponse =
 sts.assume_role(RoleArn='arn:aws:iam::accountA:role/
RoleName',RoleSessionName='tryDax')
credentials = botocore.session.get_session()['Credentials']

dax = amazondax.AmazonDaxClient(session, region_name=region,
 endpoints=[dax_endpoint], aws_access_key_id=credentials['AccessKeyId'],
 aws_secret_access_key=credentials['SecretAccessKey'],
 aws_session_token=credentials['SessionToken'])
client = dax
```

Node.js

Esta seção ajuda você a modificar seu código de cliente do DAX existente para permitir o acesso ao DAX entre contas. Caso ainda não tenha um código de cliente do DAX, você pode encontrar exemplos de código que funcionam no tutorial do Node.js e DAX. Lembre-se de substituir cada *espaço reservado de entrada de usuário* pelos valores corretos para suas contas.

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
  return new Promise((resolve, reject) => {
    const sts = new AWS.STS();
```

```
    const roleParams = {
      RoleArn: 'arn:aws:iam::accountA:role/RoleName',
      RoleSessionName: 'tryDax',
    };
    sts.assumeRole(roleParams, (err, session) => {
      if(err) {
        reject(err);
      } else {
        resolve({
          accessKeyId: session.Credentials.AccessKeyId,
          secretAccessKey: session.Credentials.SecretAccessKey,
          sessionToken: session.Credentials.SessionToken,
        });
      }
    });
  });
};

const createDaxClient = async() => {
  const credentials = await getCredentials();
  const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
  return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
  client.get(...);
  ...
}).catch((error) => {
  console.log('Caught an error: ' + error);
});
```

Guia de dimensionamento de clusters do DAX

Este guia fornece recomendações para ajudar você a escolher um tamanho de cluster e um tipo de nó do Amazon DynamoDB Accelerator (DAX) para sua aplicação. Essas instruções fornecem orientações ao longo das etapas de estimar o tráfego do DAX da sua aplicação, selecionar uma configuração de cluster e testá-la.

Se você já tem um cluster do DAX e deseja avaliar se ele tem o número e o tamanho apropriados de nós, consulte Escalar um cluster do DAX.

Tópicos

- Visão geral
- Estimar tráfego
- Testes de carga

Visão geral

É importante dimensionar o cluster do DAX adequadamente para sua workload, quer você esteja criando um novo cluster ou mantendo um cluster existente. Conforme o tempo passa e a workload da sua aplicação muda, você deve revisar periodicamente suas decisões de escalabilidade para garantir que elas ainda sejam adequadas.

Normalmente o processo segue estas etapas:

1. Estimativa do tráfego. Nesta etapa, você fará previsões sobre o volume de tráfego que a aplicação enviará ao DAX, a natureza do tráfego (operações de leitura versus gravação) e a taxa de acertos do cache esperada.
2. Testes de carga Nesta etapa, você cria um cluster e envia tráfego para ele espelhando suas estimativas da etapa anterior. Repita essa etapa até encontrar uma configuração de cluster adequada.
3. Monitoramento da produção. Enquanto a aplicação estiver usando o DAX na produção, você deverá monitorar o cluster para validar continuamente se ele ainda está dimensionado corretamente conforme a workload muda ao longo do tempo.

Estimar tráfego

Existem três fatores principais que caracterizam uma workload típica do DAX:

- Taxa de acertos do cache
- Read capacity units (RCUs – Unidades de capacidade de leitura) por segundo
- Write capacity units (WCUs – Unidades de capacidade de gravação) por segundo

Estimar taxa de acertos de cache

Se você já tem um cluster do DAX, poderá usar as métricas do Amazon CloudWatch `ItemCacheHits` e `ItemCacheMisses`para determinar a taxa de acertos do cache. A taxa de acertos do cache é igual a `ItemCacheHits`/(`ItemCacheHits` + `ItemCacheMisses`). Se sua workload incluir operações `Query` ou `Scan`, você também deverá examinar as métricas `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits` e `ScanCacheMisses`. As taxas de acertos do cache variam de acordo com a aplicação e são fortemente influenciadas pela configuração de vida útil (TTL) do cluster. As taxas de acertos típicas de aplicações que usam o DAX são variam de 85 a 95%.

Estimar unidades de capacidade de leitura e gravação

Se você já tiver tabelas do DynamoDB para sua aplicação, consulte as métricas do CloudWatch`ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits`. Use a estatística `Sum` e divida pelo número de segundos no período.

Se você também já tiver um cluster do DAX, lembre-se de que a métrica `ConsumedReadCapacityUnits` do DynamoDB contabiliza apenas erros do cache. Portanto, para ter uma ideia das unidades de capacidade de leitura por segundo tratadas pelo cluster do DAX, divida o número pela taxa de erro do cache (ou seja, 1 - taxa de acertos do cache).

Se você ainda não tiver uma tabela do DynamoDB, consulte a documentação sobre unidades de capacidade de leitura e de gravação para estimar o tráfego com base na taxa de solicitação prevista da aplicação, nos itens acessados por solicitação e no tamanho do item.

Ao fazer estimativas de tráfego, considere o crescimento futuro e os picos esperados e inesperados em seu planejamento a fim de garantir que o cluster tenha espaço suficiente para aumentos do tráfego.

Testes de carga

A próxima etapa depois de estimar o tráfego é testar a configuração do cluster sob carga.

1. Para o teste inicial de carga, recomendamos que você comece com o tipo de nó `dax.r4.large`, a performance fixa de menor custo e o tipo de nó otimizado para memória.
2. Um cluster tolerante a falhas requer pelo menos três nós, distribuídos por três zonas de disponibilidade. Nesse caso, se uma zona de disponibilidade ficar indisponível, o número efetivo de zonas de disponibilidade será reduzido em um terço. Para o teste inicial de carga,

recomendamos começar com um cluster de dois nós, que simula a falha de uma zona de disponibilidade em um cluster de três nós.

3. Promova tráfego contínuo (conforme estimado na etapa anterior) para o cluster de teste durante o teste de carga.
4. Monitore a performance do cluster durante o teste de carga.

O ideal é que o perfil do tráfego que você promove durante o teste de carga seja o mais semelhante possível ao tráfego real da aplicação. Isso inclui a distribuição de operações (p. ex., 70 por cento de `GetItem`, 25 por cento de `Query` e 5 por cento de `PutItem`), a taxa de solicitação para cada operação, o número de itens acessados por solicitação e a distribuição de tamanhos dos itens. Para obter uma taxa de acerto do cache semelhante à taxa de acerto de cache esperada da sua aplicação, preste muita atenção à distribuição de chaves no tráfego de teste.

> **Note**
>
> Tenha cuidado ao carregar testes de tipos de nó T2 (`dax.t2.small` e `dax.t2.medium`). Os tipos de nó T2 fornecem performance de CPU com capacidade de intermitência que varia ao longo do tempo, dependendo do saldo de crédito da CPU do nó. Um cluster do DAX em execução em nós T2 pode parecer estar funcionando normalmente, mas se algum nó estiver com intermitência acima da performance de referência da instância, o nó estará gastando seu saldo de créditos de CPU acumulados. Quando o saldo de crédito está baixo, a performance é gradualmente reduzida para o nível de desempenho de linha de base.

Monitore o cluster do DAX durante o teste de carga para determinar se o tipo de nó que você está usando para o teste de carga é o tipo de nó certo para você. Além disso, durante um teste de carga, você deve monitorar a taxa de solicitações e a taxa de acertos do cache para garantir que sua infraestrutura de teste esteja realmente direcionando a quantidade de tráfego planejada.

Você deve prestar atenção ao consumo de bytes de rede do tipo de instância de cluster selecionado. Exceder a largura de banda de linha de base disponível para uma instância do Amazon EC2 é um indício de que o cluster pode não sustentar a workload da aplicação e precisa ser escalado.

Se o teste de carga indicar que a configuração do cluster selecionada não pode sustentar a workload da aplicação, você deve mudar para um tipo de nó maior, especialmente se perceber alta utilização da CPU no nó primário no cluster, taxas de remoção elevadas ou alta utilização de memória cache. Se as taxas de acerto forem consistentemente altas e a proporção do tráfego de leitura para

gravação for alta, convém considerar a adição de mais nós ao cluster. Consulte Escalar um cluster do DAX para obter orientações adicionais sobre quando usar um tipo de nó maior (escalabilidade vertical) ou adicionar mais nós (escalabilidade horizontal).

Você deve repetir o teste de carga depois de fazer alterações na configuração do cluster.

Modelagem de dados para tabelas do DynamoDB

Antes de nos aprofundarmos na modelagem de dados, é importante entender alguns fundamentos do DynamoDB. O DynamoDB é um banco de dados NoSQL de chave-valor que permite um esquema flexível. O conjunto de atributos de dados, além dos atributos de chave de cada item, pode ser uniforme ou distinto. O esquema de chaves do DynamoDB está na forma de uma chave primária simples, em que uma chave de partição identifica exclusivamente um item, ou na forma de uma chave primária composta, em que uma combinação de chave de partição e chave de classificação define exclusivamente um item. A chave de partição é em hash para determinar a localização física dos dados e recuperá-los. Portanto, é importante escolher um atributo de alta cardinalidade e escalável horizontalmente como chave de partição para garantir uma distribuição uniforme dos dados. O atributo de chave de classificação é opcional no esquema de chaves, e ter uma chave de classificação permite modelar relações de um para muitos e criar coleções de itens no DynamoDB. As chaves de classificação, também são chamadas de chaves de intervalo, são usadas para classificar itens em uma coleção de itens e também permitem operações flexíveis baseadas em intervalos.

Para ver mais detalhes e conhecer as práticas recomendadas sobre o esquema de chaves do DynamoDB, você pode consultar o seguinte:

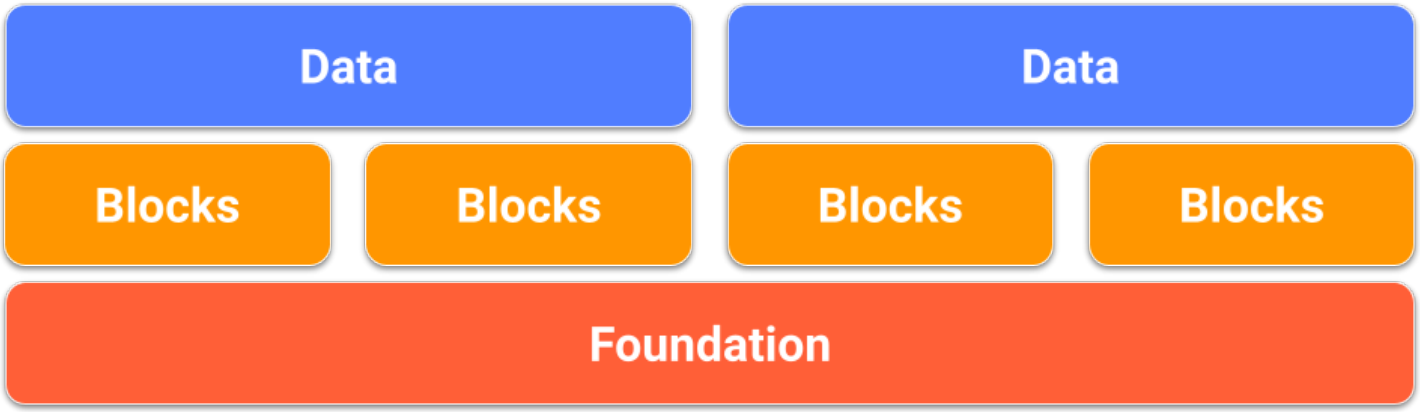
- the section called “Partições e distribuição de dados no DynamoDB”
- the section called “Design de chave de partição”
- the section called “Design de chaves de classificação”
- Escolher a chave de partição correta do DynamoDB

Os índices secundários geralmente são necessários para oferecer compatibilidade com padrões de consulta adicionais no DynamoDB. Eles são tabelas de sombra em que os mesmos dados são organizados por meio de um esquema de chaves diferente em comparação com a tabela base. Um índice secundário local (LSI) compartilha a mesma chave de partição da tabela base e permite ter uma chave de classificação alternativa, o que possibilita que ele compartilhe a capacidade da tabela base. Um índice secundário global (GSI) pode ter uma chave de partição diferente, bem como um atributo de chave de classificação diferente da tabela base, o que significa que o gerenciamento de throughput de um GSI é independente da tabela base.

Para ver mais detalhes sobre índices secundários e conhecer as práticas recomendadas, você pode consultar o seguinte:

- the section called “Trabalhar com índices”
- the section called “Índices secundários”

Vamos analisar a modelagem de dados um pouco mais de perto. A criação de um esquema flexível e altamente otimizado no DynamoDB, ou em qualquer banco de dados NoSQL, pode ser uma habilidade difícil de aprender. O objetivo deste módulo é ajudar você a desenvolver um fluxograma mental para criar um esquema que conduza você do caso de uso à produção. Começaremos com uma introdução à escolha básica de qualquer design: uma única tabela versus várias tabelas. Em seguida, analisaremos a variedade de padrões de design (componentes básicos) que podem ser usados para obter vários resultados organizacionais ou de performance para sua aplicação. Por fim, estamos incluindo uma variedade de pacotes completos de design de esquemas para diferentes casos de uso e setores.



Tópicos

- Coleções de itens: como modelar relações de um para muitos no DynamoDB
- Fundamentos da modelagem de dados no DynamoDB
- Componentes básicos da modelagem de dados no DynamoDB
- Pacotes de design de esquema de modelagem de dados no DynamoDB
- Práticas recomendadas para modelagem de dados relacionais no DynamoDB

Coleções de itens: como modelar relações de um para muitos no DynamoDB

No DynamoDB, uma coleção de itens é um grupo de itens que compartilham o mesmo valor de chave de partição, o que significa que os itens estão relacionados. As coleções de itens são o

mecanismo primário para modelar relações de um para muitos no DynamoDB. As coleções de itens só podem existir em tabelas ou índices configurados para usar uma chave primária composta.

> **Note**
>
> Podem existir coleções de itens em uma tabela base ou em um índice secundário. Para obter mais informações especificamente sobre como as coleções de itens interagem com índices, consulte Conjuntos de itens em índices secundários locais.

Considere a seguinte tabela que mostra três usuários diferentes e seus inventários no jogo:

Primary key		Attributes		
Partition key: PK	**Sort key: SK**			
account1234	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	inventory::weapons	data		
		{"weapons" : [{ "name" : "Sword of the Ancients", "type" : "sword", "gear score" : 320}]}		
	login-data	pw	state	last-login
		d1e8a70b5ccab1dc2f56bbf7e99f064a660c08e361a35751b9c483c88943d082	Active	1649276737
account1387	info	data		
		{"email" : "bot123@gmail.com"}		
	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	login-data	pw	state	last-login
		k2g8jk0m5ppab1dc2f56bbf7e99f064a660c08e361a35751b9c464r23943i082	Banned	1649456737
account1138	info	data		
		{"email" : "luh-3417@gmail.com" }		
	login-data	pw	state	last-login
		88A41A9A62B11CCC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4	Active	14275516966

Para alguns itens em cada coleção, a chave de classificação é uma concatenação composta de informações usadas para agrupar dados, como `inventory::armor`, `inventory::weapon` ou `info`. Cada coleção de itens pode ter uma combinação diferente desses atributos como a chave de classificação. O usuário `account1234` tem o item `inventory::weapons`, enquanto o usuário `account1387` não tem (porque ainda não o encontrou). O usuário `account1138` usa apenas dois itens para a chave de classificação (já que ainda não tem inventário) enquanto os outros usuários usam três.

O DynamoDB permite recuperar itens seletivamente dessas coleções para fazer o seguinte:

- Recuperar todos os itens de um usuário específico
- Recuperar apenas um item de um usuário específico
- Recuperar todos os itens de um tipo específico pertencentes a determinado usuário

Acelerar as consultas organizando os dados com coleções de itens

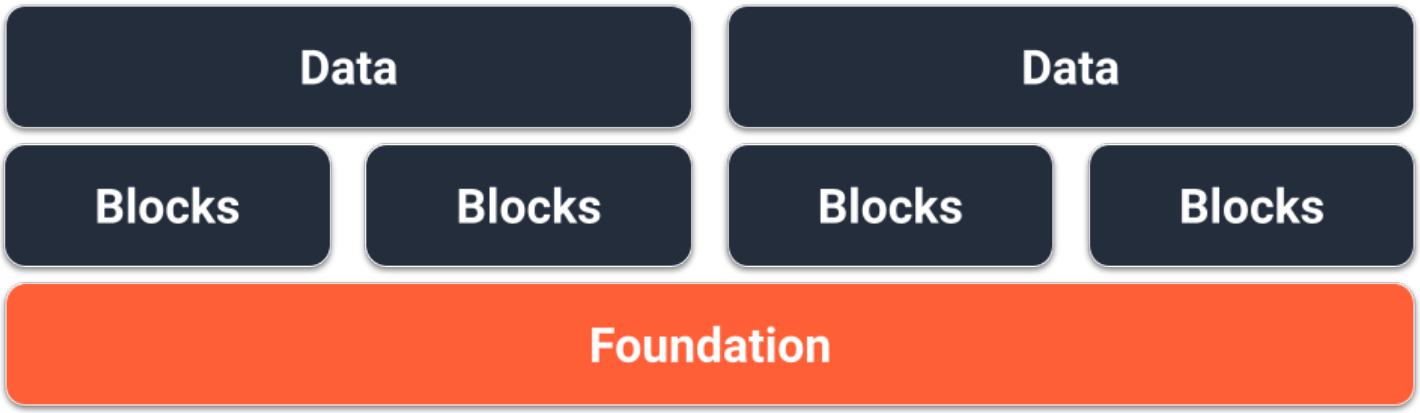
Nesse exemplo, cada um dos itens nas três coleções representa um jogador e o modelo de dados que escolhemos, com base nos padrões de acesso do jogo e do jogador. De quais dados o jogo precisa? Quando ele precisa desses dados? Com que frequência ele precisa dos dados? Qual é o custo de fazer isso dessa maneira? Essas decisões de modelagem de dados foram tomadas com base nas respostas a essas perguntas.

Nesse jogo, há uma página diferente apresentada ao jogador para o inventário de armas e outra página para a armadura. Quando o jogador abre o inventário, as armas são mostradas primeiro porque queremos que a página seja carregada muito rapidamente, enquanto as páginas de inventário subsequentes podem ser carregadas depois disso. Como cada um desses tipos de item pode ser bastante grande à medida que o jogador adquire mais itens no jogo, decidimos que cada página de inventário seria seu próprio item na coleção do jogador no banco de dados.

A seção a seguir fala mais sobre como você pode interagir com as coleções de itens por meio da operação `Query`.

Fundamentos da modelagem de dados no DynamoDB

Esta seção aborda a camada básica, examinando os dois tipos de design de tabela: tabela única e várias tabelas.



Conceitos básicos do design de tabela única

Uma das opções de base para o esquema do DynamoDB é o design de tabela única. O design de tabela única é um padrão que permite armazenar vários tipos (entidades) de dados em uma única tabela do DynamoDB. Seu objetivo é otimizar os padrões de acesso aos dados, melhorar a performance e reduzir os custos, eliminando a necessidade de manter várias tabelas e relacionamentos complexos entre elas. Isso é possível porque o DynamoDB armazena itens com a mesma chave de partição (conhecida como coleção de itens) nas mesmas partições. Nesse design, diferentes tipos de dados são armazenados como itens na mesma tabela e cada item é identificado por uma chave de classificação exclusiva.

Primary key		Attributes	
Partition key: PK	**Sort key: SK**		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Vantagens

- Oferece localidade de dados para atender a consultas para vários tipos de entidades em uma única chamada de banco de dados.
- Reduz os custos financeiros gerais e de latência das leituras:
  - Uma única consulta para dois itens com menos de 4 KB no total corresponde a 0,5 RCU com consistência final.

  - Duas consultas para dois itens com menos de 4 KB no total corresponde a 1 RCU com consistência final (0,5 RCU cada).
  - O tempo para exibir duas chamadas de banco de dados separadas normalmente será maior do que o de uma única chamada.
- Reduz o número de tabelas a serem gerenciadas:
  - Não é necessário manter permissões em vários perfis ou políticas do IAM.
  - O gerenciamento de capacidade da tabela é distribuído proporcionalmente entre todas as entidades, o que geralmente resulta em um padrão de consumo mais previsível.
  - O monitoramento requer menos alarmes.
  - As chaves de criptografia gerenciadas pelo cliente só precisam ser alternadas em uma única tabela.
- Suaviza o tráfego para a tabela:
  - Ao agregar vários padrões de uso à mesma tabela, o uso geral tende a ser mais suave (da mesma forma que a performance de um índice de ações tende a ser mais suave do que qualquer ação individual), o que é mais adequado para conseguir maior utilização com tabelas de modo provisionado.

Desvantagens

- A curva de aprendizado pode ser pronunciada devido ao design paradoxal em comparação com bancos de dados relacionais.
- Os requisitos de dados devem ser consistentes em todos os tipos de entidade.
  - Os backups são tudo ou nada; portanto, se alguns dados não forem essenciais, pense na possibilidade de mantê-los em uma tabela separada.
  - A criptografia da tabela é compartilhada entre todos os itens. Para aplicações de vários locatários com requisitos individuais de criptografia de locatário, seria necessária a criptografia do lado do cliente.
  - As tabelas com uma combinação de dados históricos e dados operacionais não serão tão beneficiadas com a habilitação da classe de armazenamento de acesso infrequente. Para obter mais informações, consulte Classes de tabelas do DynamoDB.
- Todos os dados alterados serão propagados para o DynamoDB Streams mesmo que seja necessário processar somente um subconjunto de entidades.

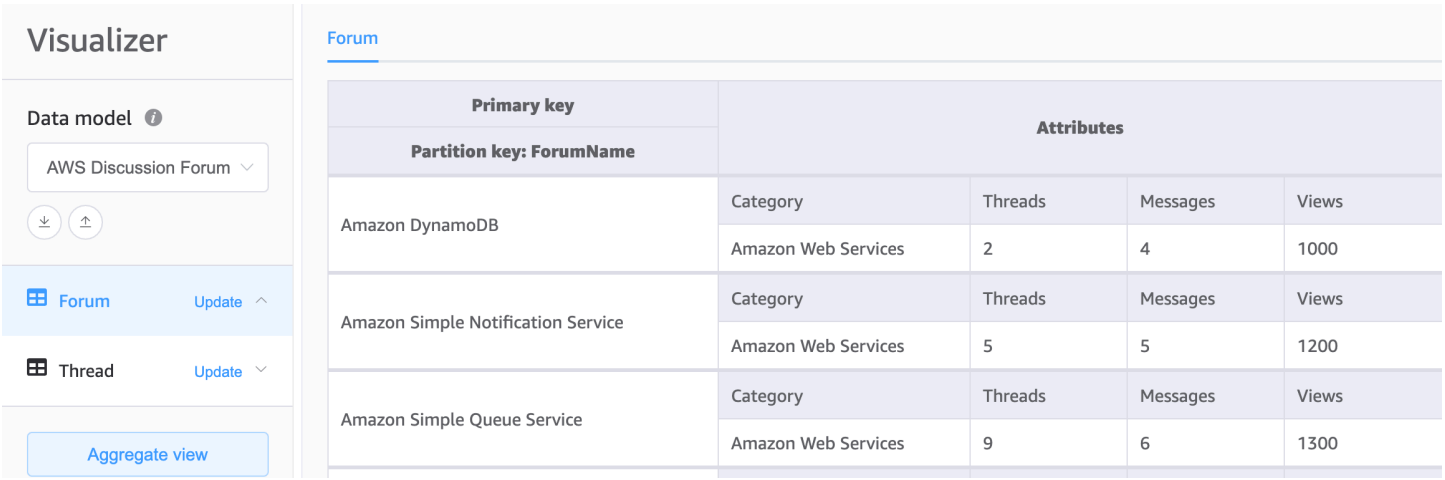
  - Graças aos filtros de eventos do Lambda, isso não afetará sua fatura ao usar o Lambda, mas haverá um custo adicional se você usar a Kinesis Consumer Library.
- Quando se usa o GraphQL, é mais difícil de implementar o design de tabela única.
- Ao usar clientes SDK de nível superior, como o `DynamoDBMapper` do Java ou o Enhanced Client, pode ser mais difícil processar resultados porque na mesma resposta pode haver itens associados a classes diferentes.

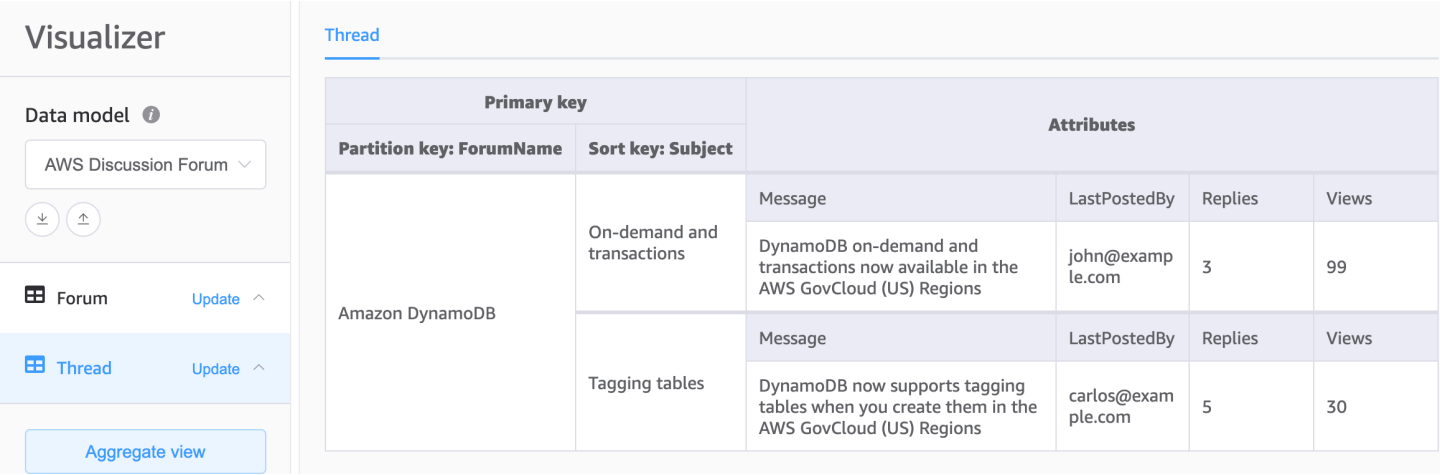
Quando usar

O design de tabela única funciona bem para aplicativos que frequentemente consultam vários tipos de entidades juntos ou precisam manter relacionamentos entre diferentes tipos de dados. É particularmente eficaz quando seus padrões de acesso se beneficiam da localidade de dados e quando você deseja minimizar a sobrecarga de gerenciar várias tabelas.

Conceitos básicos do design de várias tabelas

A segunda opção de base para o nosso esquema do DynamoDB é o design de várias tabelas. O design de várias tabelas é um padrão mais parecido com um design de banco de dados tradicional em que você armazena um único tipo (entidade) de dados em cada tabela do DynamoDB. Os dados em cada tabela ainda serão organizados por chave de partição para que a performance e a escalabilidade de um único tipo de entidade sejam otimizadas, mas as consultas em várias tabelas deverão ser feitas de forma independente.





Vantagens

- É mais simples de projetar para quem não está acostumado a trabalhar com design de tabela única.
- Possibilita uma implementação mais fácil de resolvedores do GraphQL devido ao mapeamento de cada resolvedor para uma única entidade (tabela).
- Permite requisitos de dados exclusivos em diferentes tipos de entidade:
  - É possível fazer backups de tabelas individuais de missão crítica.
  - É possível gerenciar a criptografia de cada tabela. Para aplicações de vários locatários com requisitos individuais de criptografia de locatário, tabelas de locatários separadas possibilitam que cada cliente tenha sua própria chave de criptografia.
  - Para obter todos os benefícios de redução de custos, a classe de armazenamento de acesso infrequente pode ser habilitada apenas nas tabelas com dados históricos. Para obter mais informações, consulte Classes de tabelas do DynamoDB.
- Cada tabela terá seu próprio fluxo de dados de alteração, permitindo que uma função do Lambda dedicada seja projetada para cada tipo de item, em vez de um único processador monolítico.

Desvantagens

- Para padrões de acesso que exigem dados em várias tabelas, diversas leituras do DynamoDB serão necessárias e talvez os dados precisem ser processados e unidos no código do cliente.
- As operações e o monitoramento de várias tabelas exigem mais alarmes do CloudWatch, e cada tabela deve ser escalada de forma independente.

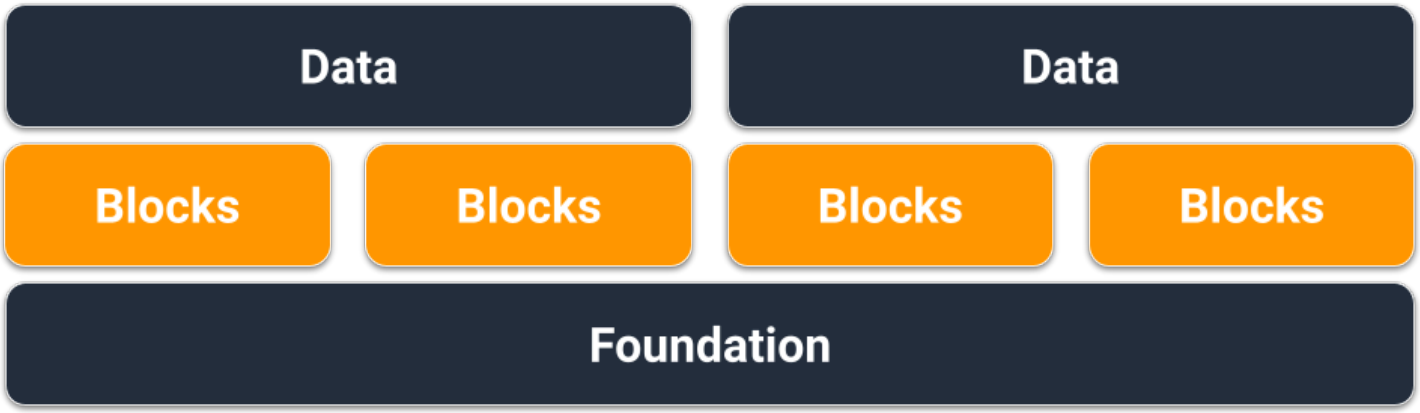
- As permissões de cada tabela precisarão ser gerenciadas separadamente. A adição de tabelas no futuro exigirá uma alteração em todos os perfis ou políticas essenciais do IAM.

Quando usar

Se os padrões de acesso da aplicação não precisarem consultar várias entidades ou tabelas ao mesmo tempo, o design de várias tabelas será uma abordagem adequada e suficiente.

Componentes básicos da modelagem de dados no DynamoDB

Esta seção abordará a camada de componentes básicos que oferecem os padrões de design a serem utilizados na aplicação.



Tópicos

- Componente básico: chave de classificação composta
- Componente básico: multilocação
- Componente básico: índice esparso
- Componente básico: vida útil
- Componente básico: vida útil para arquivamento
- Componente básico: particionamento vertical
- Componente básico: fragmentação de gravação

Componente básico: chave de classificação composta

Quando as pessoas pensam em NoSQL, é possível que elas também o imaginem como não relacional. Em última análise, não há motivo que impeça a inserção de relações em um esquema do

DynamoDB. Elas só parecem diferentes dos bancos de dados relacionais e das respectivas chaves externas. Um dos padrões mais importantes que podemos usar para desenvolver uma hierarquia lógica de nossos dados no DynamoDB é uma chave de classificação composta. A maneira mais comum de criar uma é separar cada camada da hierarquia (camada pai > camada filha > camada neta) com uma hashtag. Por exemplo, `PARENT#CHILD#GRANDCHILD#ETC`.

Primary key	
Partition key: PK	**Sort key: SK**
UserID	CART#ACTIVE#Apples
	CART#ACTIVE#Bananas
	CART#SAVED#Oranges
	CART#SAVED#Pears
	WISH#VEGGIES#Carrots

Embora uma chave de partição no DynamoDB sempre exija o valor exato para consulta de dados, podemos aplicar uma condição parcial à chave de classificação da esquerda para a direita, de modo semelhante a percorrer uma árvore binária.

No exemplo acima, temos uma loja de comércio eletrônico em que é necessário manter o carrinho de compras em todas as sessões do usuário. Sempre que o usuário fizer login, talvez queira ver o carrinho de compras completo, incluindo itens salvos para outro momento. Porém, quando ele entra na finalização da compra, somente os itens no carrinho ativo devem ser carregados para compra. Como ambas as `KeyConditions` solicitam explicitamente as chaves de classificação CART, os dados adicionais da lista de desejos simplesmente são ignorados pelo DynamoDB no momento da leitura. Embora os itens salvos e ativos façam parte do mesmo carrinho, precisamos tratá-los de forma distinta em diferentes partes da aplicação. Portanto, aplicar uma `KeyCondition` ao prefixo da chave de classificação é a melhor maneira de recuperar somente os dados necessários para cada parte da aplicação.

Principais atributos deste componente básico

- Os itens relacionados são armazenados lado a lado no mesmo local para tornar o acesso aos dados eficaz.

- Por meio de expressões `KeyCondition`, é possível recuperar seletivamente os subconjuntos da hierarquia, o que significa que não há desperdício de RCUs.
- Diferentes partes da aplicação podem armazenar os respectivos itens sob um prefixo específico, evitando itens sobrescritos ou gravações conflitantes.

Componente básico: multilocação

Muitos clientes usam o DynamoDB para hospedar dados de aplicações de vários locatários. Para essas circunstâncias, nosso objetivo é projetar o esquema de uma forma que mantenha todos os dados de um único locatário em sua própria partição lógica da tabela. Essa ideia se baseia no conceito de coleção de itens, que é um termo para todos os itens em uma tabela do DynamoDB que têm a mesma chave de partição. Para obter mais informações sobre como o DynamoDB aborda a multilocação, consulte Multilocação no DynamoDB.

Primary key		Attributes
Partition key: PK	**Sort key: SK**	
UserOne	PhotoID1	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID2	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserTwo	PhotoID3	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID4	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserThree	PhotoID5	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]

Neste exemplo, estamos administrando um site de hospedagem que provavelmente tem milhares de usuários. Inicialmente, cada usuário só carregará fotos em seu próprio perfil, mas, por padrão, não permitiremos que um veja as fotos do outro. Em condições ideais, seria adicionado um nível a mais de isolamento à autorização da chamada de cada usuário à sua API para garantir que todos solicitem apenas dados da partição deles. Entretanto, no nível do esquema, chaves de partição exclusivas são adequadas.

Principais atributos deste componente básico

- A quantidade de dados lidos por qualquer usuário ou locatário só pode ser igual à quantidade total de itens na partição dele.
- A remoção dos dados de um locatário devido ao encerramento de uma conta ou a uma solicitação de conformidade pode ser feita de forma discreta e econômica. Basta realizar uma consulta em que a chave de partição seja igual ao ID do locatário e, em seguida, executar uma operação `DeleteItem` para cada chave primária retornada.

> **Note**
>
> Como a multilocação faz parte do conceito, é possível usar diferentes provedores de chaves de criptografia em uma única tabela para isolar dados com segurança. AWS O SDK de criptografia de banco de dados para o Amazon DynamoDB possibilita que você inclua criptografia do lado do cliente em workloads do DynamoDB. Você pode realizar a criptografia em cada atributo, o que permite criptografar valores de atributos específicos antes de armazená-los na tabela do DynamoDB e pesquisar atributos criptografados sem descriptografar o banco de dados inteiro de antemão.

Componente básico: índice esparso

Às vezes, um padrão de acesso exige a busca de itens que correspondam a um item raro ou recebam um status (o que exige uma resposta escalada). Em vez de consultar regularmente todo o conjunto de dados em busca desses itens, podemos nos valer do fato de os índices secundários globais (GSI) serem escassamente carregados de dados. Isso significa que apenas os itens na tabela base que tenham os atributos definidos no índice serão replicados no índice.

Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara

Primary key		Attributes	
Partition key: EscalatedTo	Sort key: State#Date		
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator
		d#11223	Sue

Neste exemplo, vemos um caso de uso de IoT em que cada dispositivo no campo relata um status regularmente. Na maioria dos relatórios, esperamos que o dispositivo informe que está tudo bem, mas às vezes pode haver uma falha e ser necessário encaminhá-la a um técnico de reparos. Para relatórios com escalonamento, o atributo `EscalatedTo` é adicionado ao item, mas não está presente de outra forma. Neste exemplo, o GSI é particionado por `EscalatedTo` e, como ele traz as chaves da tabela base, ainda podemos ver qual DeviceID relatou a falha e em que momento.

Embora no DynamoDB as leituras sejam mais baratas do que as gravações, os índices esparsos são uma ferramenta muito eficaz para casos de uso em que as instâncias de um tipo específico de item são raras mas as leituras para encontrá-las são comuns.

Principais atributos deste componente básico

- Os custos de gravação e armazenamento do GSI esparso só se aplicam a itens que correspondam ao padrão de chave; portanto, o custo do GSI pode ser significativamente menor do que o de outros GSIs em que todos os itens são replicados neles.
- É possível também usar uma chave de classificação composta para restringir ainda mais os itens que correspondam à consulta desejada; por exemplo, um carimbo de data e hora pode ser usado para que a chave de classificação veja somente as falhas relatadas nos últimos X minutos (`SK > 5 minutes ago, ScanIndexForward: False`).

Componente básico: vida útil

A maioria dos dados tem um tempo durante o qual se considera útil mantê-los em um datastore primário. Para lidar com o envelhecimento dos dados no DynamoDB, ele tem um atributo chamado vida útil (TTL). O TTL possibilita definir um recurso específico por tabela que precisa ser monitorado em relação a itens com um carimbo de data e hora de época (que esteja no passado). Isso possibilita que você exclua registros expirados da tabela gratuitamente.

> **Note**
>
> Se estiver usando Global Tables versão 2019.11.21 (atual) das tabelas globais e também usar o recurso Vida útil, o DynamoDB replicará as exclusões do TTL em todas as tabelas de réplica. A exclusão inicial do TTL não consome capacidade de gravação na região onde o TTL expira. No entanto, a exclusão do TTL replicada para as tabelas-réplica consome capacidade de gravação replicada em cada uma das regiões de réplica e, nesse caso, se aplicam cobranças.

Primary key		Attributes	
Partition key: PK	**Sort key: MessageTimestamp**		
UserID	2030-06-30T12:12:12	TTL	Message
		1909570332	Hello
	2030-06-30T12:17:22	TTL	Message
		1909570647	DynamoDB
	2030-06-30T12:22:27	TTL	Message
		1909570947	TTL

Neste exemplo, temos uma aplicação projetada para permitir que um usuário crie mensagens de curta duração. Quando se cria uma mensagem no DynamoDB, o atributo TTL é definido como uma data de sete dias no futuro pelo código da aplicação. Em aproximadamente sete dias, o DynamoDB verificará se o carimbo de data e hora de época desses itens está no passado e os excluirá.

Como as exclusões feitas pelo TTL são gratuitas, é altamente recomendável usar esse atributo para remover dados históricos da tabela. Isso reduzirá a conta geral de armazenamento a cada mês e provavelmente diminuirá os custos das leituras do usuário, pois haverá menos dados a serem recuperados pelas consultas. Embora o TTL seja habilitado em nível de tabela, cabe a você decidir

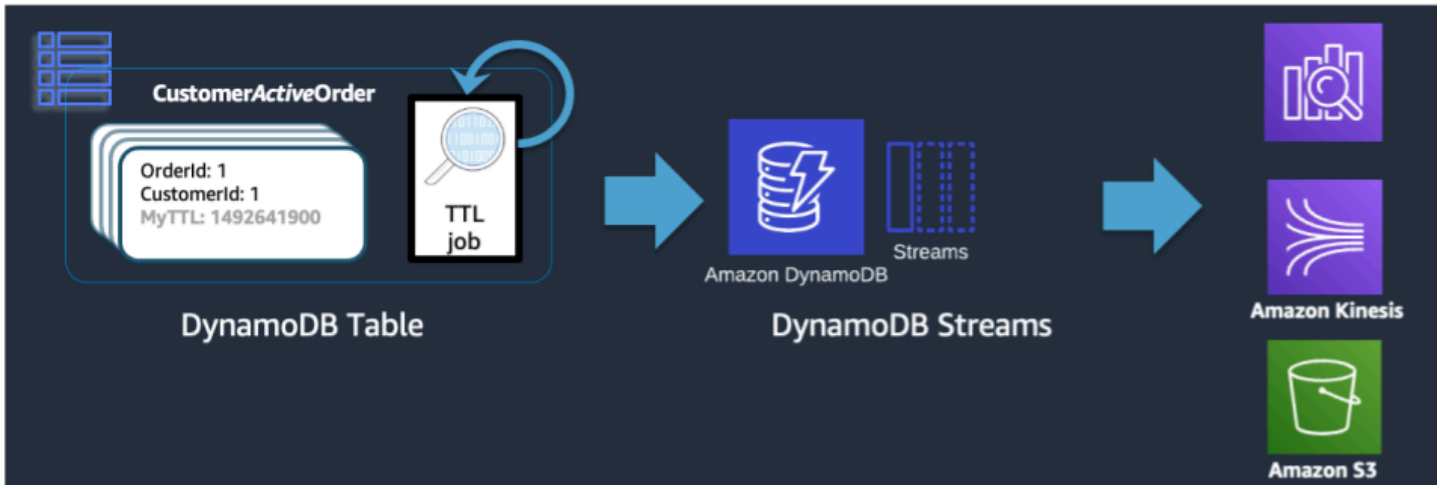
para quais itens ou entidades criará um atributo TTL e definir o carimbo de data e hora de época futuro.

Principais atributos deste componente básico

- As exclusões de TTL são executadas nos bastidores e não afetam a performance da tabela
- Embora o TTL seja um processo assíncrono que é executado a cada 6 horas, aproximadamente, a exclusão de um registro expirado pode levar mais de 48 horas.
  - Não conte com as exclusões de TTL para casos de uso, como registros de bloqueio ou gerenciamento de estado, se houver necessidade de limpar dados obsoletos em menos de 48 horas
- Você pode dar um nome de atributo válido ao TTL, mas o valor deve ser numérico.

Componente básico: vida útil para arquivamento

Embora o TTL seja uma ferramenta eficaz para excluir dados antigos do DynamoDB, muitos casos de uso exigem que os dados sejam mantidos arquivados por um período mais longo do que o do datastore primário. Nesse caso, podemos utilizar a exclusão cronometrada de registros do TTL para enviar registros expirados a um datastore de longo prazo.



Quando uma exclusão de TTL é feita pelo DynamoDB, ela ainda é enviada ao DynamoDB Stream como um evento `Delete`. No entanto, quando a exclusão é realizada pelo TTL do DynamoDB, há um atributo `principal:dynamodb` no registro de fluxo. Usando um assinante Lambda do DynamoDB Stream, podemos aplicar um filtro de eventos somente ao atributo principal do DynamoDB e ter certeza de que todos os registros que correspondam a esse filtro devem ser enviados a um armazenamento de arquivos como o S3 Glacier.

Principais atributos deste componente básico

- Quando as leituras de baixa latência do DynamoDB não são mais necessárias para os itens históricos, migrá-las para um serviço de armazenamento mais frio, como o S3 Glacier, pode reduzir significativamente os custos de armazenamento e, ao mesmo tempo, atender às necessidades de conformidade de dados de seu caso de uso.
- Se os dados persistirem no Amazon S3, poderão ser usadas ferramentas de análise econômicas, como o Amazon Athena ou o Redshift Spectrum, para realizar análises históricas dos dados.

Componente básico: particionamento vertical

Os usuários que já conhecem um banco de dados de modelos de documentos estarão familiarizados com a ideia de armazenar todos os dados relacionados em um único documento JSON. Embora o DynamoDB seja compatível com tipos de dados JSON, ele não comporta a execução de `KeyConditions` em JSON aninhado. Como `KeyConditions` são os fatores que determinam a quantidade de dados lidos do disco e, efetivamente, quantas RCUs uma consulta consome, isso pode gerar ineficiência em grande escala. Para otimizar ainda mais as gravações e leituras do DynamoDB, recomendamos dividir as entidades individuais do documento em itens individuais do DynamoDB, técnica também chamada de particionamento vertical.

```
{
        ”UserProfile” : {
                "FirstName": "Paul",
                "LastName": "Atreides",
                "DateJoined":"1965-08-01"},
        ”Store” : {
                ”store_id”: ”STOREUID”,
                 "city": "Los Angeles",
                 "zip_code":"90029”}
        "ShoppingCart" : [
                {"Spice":
                        { "SKU": "SpiceSKU",
                          "CategoryID":"FictionalSpice",
                          "DateAddded " :"2019-06-11"}},
                {"EspressoBeans":
                        { "SKU": "CaffeineSKU",
                          "CategoryID":"FOODANDDRINK",
                          "DateAddded " :"2019-06-10"}}],
        "ShippingAddress" : {
                 "street_address": "1234 Arrakis Dr",
                 "city": "Los Angeles",
                 "zip_code":"90029",
                 "status": "default"}
        ”OrderHistory#OrderUID” : {
                ”ProductA”: ”SKU_A”,
                ”ProductB”: ”SKU_B”,
                "DateOrdered":”2018-09-28"}
}
```

Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

O particionamento vertical, conforme mostrado acima, é um exemplo básico de design de tabela única em ação, mas também é possível implementá-lo em várias tabelas, se desejado. Como o DynamoDB fatura as gravações em incrementos de 1 KB, o ideal é particionar o documento de uma maneira que resulte em itens com menos de 1 KB.

Principais atributos deste componente básico

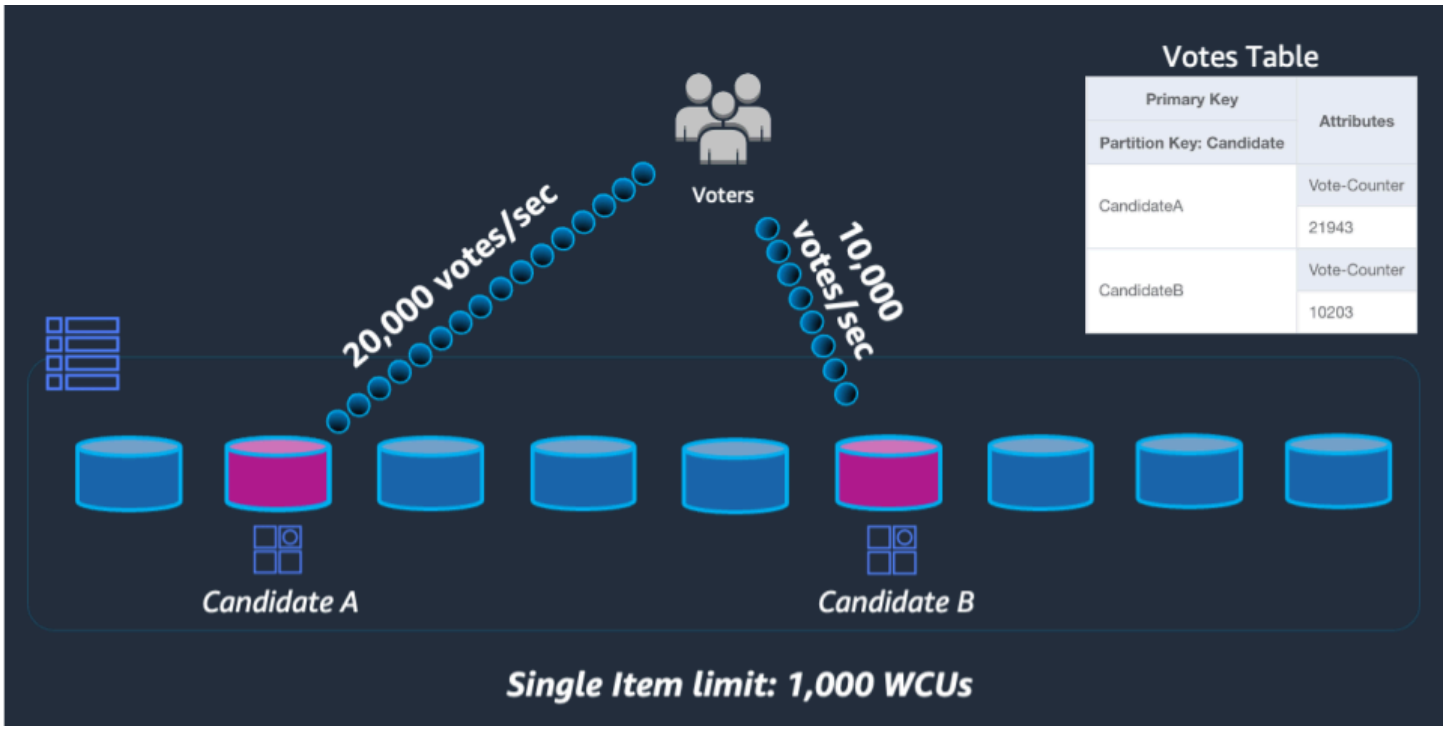
- Uma hierarquia de relacionamentos de dados é mantida por meio de prefixos de chave de classificação para que seja possível reconstruir a estrutura singular do documento no lado do cliente, se necessário.
- É possível atualizar componentes singulares da estrutura de dados de maneira independente, o que resulta em atualizações de itens tão pequenos quanto 1 WCU.
- Usando a chave de classificação `BeginsWith`, a aplicação pode recuperar dados semelhantes em uma única consulta, agregando custos de leitura para reduzir o custo e latência totais.
- Documentos grandes podem ultrapassar facilmente o limite de tamanho de item individual de 400 KB no DynamoDB, e o particionamento vertical ajuda a contornar esse limite

Componente básico: fragmentação de gravação

Um dos poucos limites rígidos do DynamoDB é a restrição ao throughput que uma única partição física pode manter por segundo (não necessariamente uma única chave de partição). No momento, esses limites são:

- 1.000 WCU (ou 1.000 <= 1 KB itens gravados por segundo) e 3.000 RCU (ou 3.000 <= 4 KB de leituras por segundo) altamente consistentes ou
- 6.000 leituras <= 4 KB por segundo finais consistentes

Caso as solicitações da tabela excedam qualquer um desses limites, um erro é enviado de volta ao SDK do cliente `ThroughputExceededException`, mais comumente chamado de controle de utilização. Os casos de uso que exigem operações de leitura além desse limite geralmente são mais bem atendidos por meio da colocação de um cache de leitura na frente do DynamoDB, mas as operações de gravação exigem um design em nível de esquema conhecido como fragmentação de gravação.



Primary Key	Attributes	
Partition Key: Candidate		
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53

Para resolver esse problema, acrescentaremos um número inteiro aleatório ao final da chave de partição para cada candidato no código `UpdateItem` da aplicação. O intervalo do gerador de números inteiros aleatórios precisará ter um limite superior correspondente ou acima da quantidade esperada de gravações por segundo para determinado candidato dividido por 1.000. Para atender a 20.000 votos por segundo, teríamos rand(0,19). Agora que os dados estão armazenados em partições lógicas separadas, eles devem ser combinados novamente no momento da leitura. Como o total de votos não precisa ser em tempo real, uma função do Lambda programada para ler todas as partições de votos a cada X minutos pode realizar uma agregação ocasional para cada candidato e gravá-la de volta em um único registro de total de votos para leituras ao vivo.

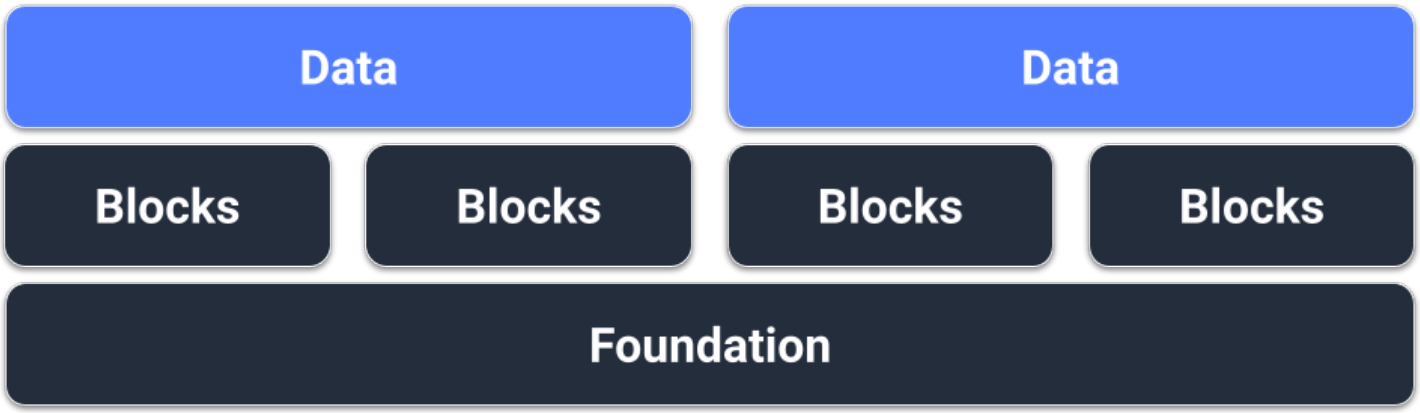
Principais atributos deste componente básico

- Para casos de uso com um throughput extremamente alto para determinada chave de partição que não pode ser evitada, as operações de gravação podem ser distribuídas artificialmente por várias partições do DynamoDB.
- Os GSIs com uma chave de partição com baixa cardinalidade também devem utilizar esse padrão, pois o controle de utilização em um GSI aplicará uma pressão contrária nas operações de gravação na tabela base.

Pacotes de design de esquema de modelagem de dados no DynamoDB

Saiba mais sobre os pacotes de design de esquemas de modelagem de dados para o DynamoDB, incluindo casos de uso, padrões de acesso e designs de esquemas finais para redes sociais, perfis

de jogos, gerenciamento de reclamações, pagamentos recorrentes, status de dispositivos e lojas virtuais.



Pré-requisitos

Antes de tentarmos criar nosso esquema para o DynamoDB, primeiro precisamos reunir alguns dados de pré-requisitos sobre o caso de uso que o esquema precisa apoiar. Diferentemente dos bancos de dados relacionais, o DynamoDB é fragmentado por padrão, o que significa que os dados permanecerão em vários servidores nos bastidores. Portanto, é importante projetar de acordo com a localidade dos dados. Precisaremos montar a seguinte lista para cada design de esquema:

- Lista de entidades (diagrama ER)
- Volumes e throughput previstos para cada entidade
- Padrões de acesso que precisam ser atendidos (consultas e gravações)
- Requisitos de retenção de dados

Tópicos

- Design de esquema de rede social no DynamoDB
- Design de esquema de perfil de jogo no DynamoDB
- Design do esquema do sistema de gerenciamento de reclamações no DynamoDB
- Design de esquema de pagamentos recorrentes no DynamoDB
- Monitoramento das atualizações de status do dispositivo no DynamoDB
- Usar o DynamoDB como armazenamento de dados para uma loja virtual

Design de esquema de rede social no DynamoDB

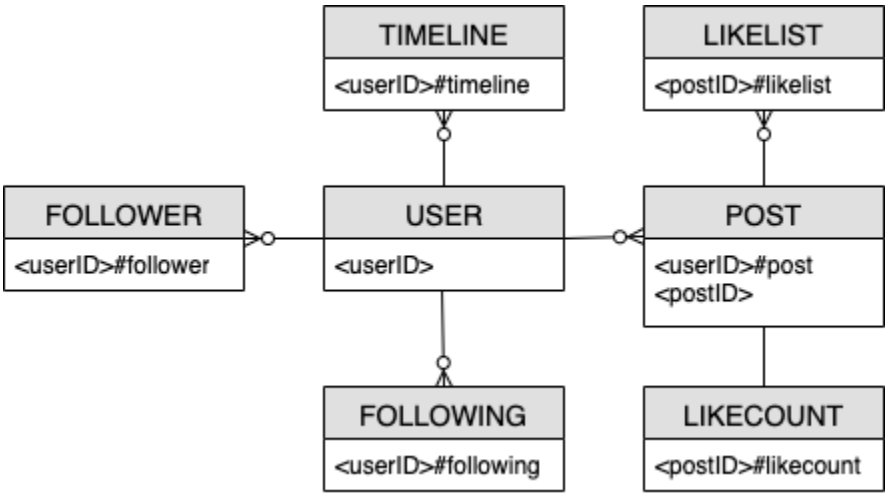
Caso de uso de negócios de redes sociais

Esse caso de uso aborda o uso do DynamoDB como uma rede social. Uma rede social é um serviço on-line que permite que diferentes usuários interajam entre si. A rede social que criaremos permitirá que o usuário veja uma linha do tempo que consiste em suas publicações, seguidores, quem ele está seguindo e publicações escritas por quem ele está seguindo. Os padrões de acesso para este design de esquema são:

- Obter informações do usuário para determinado ID de usuário
- Obter lista de seguidores para determinado ID de usuário
- Obter lista de seguidores para determinado ID de usuário
- Obter lista de seguidores para determinado ID de usuário
- Obter a lista de usuários que curtiram a publicação de determinado PostID
- Obter a contagem de curtidas para determinado PostID
- Obter a linha do tempo de determinado ID de usuário

Diagrama de relacionamento de entidades de redes sociais

Este é o diagrama de relacionamento de entidades (ERD) que usaremos para o design do esquema da rede social.



Padrões de acesso da rede social

Estes são os padrões de acesso que vamos considerar para o design do esquema da rede social.

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

Evolução do design do esquema da rede social

O DynamoDB é um banco de dados NoSQL. Por isso, ele não permite que você realize uma união; isto é, uma operação que combina dados de vários bancos de dados. Os clientes que não estão familiarizados com o DynamoDB podem aplicar filosofias de design do sistema de gerenciamento de banco de dados relacional (RDBMS) (como criar uma tabela para cada entidade) ao DynamoDB quando na verdade não precisam. O objetivo do design de tabela única do DynamoDB é gravar dados em um formato pré-unido de acordo com o padrão de acesso da aplicação e, em seguida, usar imediatamente os dados sem computação adicional. Para obter mais informações, consulte Design de tabela única versus design de várias tabelas no DynamoDB.

Agora vamos ver como desenvolveremos nosso design de esquema para abordar todos os padrões de acesso.

Etapa 1: abordar o padrão de acesso 1 (**`getUserInfoByUserID`**)

Para receber informações de um usuário específico, precisaremos usar `Query` na tabela base com uma condição de chave `PK=<userID>`. A operação de consulta permite paginar os resultados, o que pode ser útil quando um usuário tem muitos seguidores. Para obter mais informações sobre Query, consulte Consultar tabelas no DynamoDB.

Em nosso exemplo, rastreamos dois tipos de dados para nosso usuário: “count” e “info”. A “contagem” de um usuário reflete quantos seguidores ele tem, quantos usuários ele está seguindo e quantas publicações ele criou. As “informações” de um usuário refletem suas informações pessoais, como seu nome.

Vemos esses dois tipos de dados representados pelos dois itens abaixo. O item que tem “count” na chave de classificação (SK) tem maior probabilidade de mudar do que o item com “info”. O DynamoDB considera o tamanho do item conforme ele aparece antes e depois da atualização, e

o throughput provisionado consumido refletirá o item de tamanho maior. Mesmo se você atualizar apenas um subconjunto dos atributos do item, `UpdateItem` ainda consumirá a quantidade total do throughput provisionado (o maior tamanho de item de “antes” e “depois”). Você pode obter os itens por meio de uma única operação `Query` e usar `UpdateItem` para adicionar ou subtrair atributos numéricos existentes.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...

Etapa 2: abordar o padrão de acesso 2 (**`getFollowerListByUserID`**)

Para obter uma lista dos usuários que estão seguindo determinado usuário, precisaremos utilizar `Query` na tabela base com uma condição de chave `PK=<userID>#follower`.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				

Etapa 3: abordar o padrão de acesso 3 (**`getFollowingListByUserID`**)

Para obter uma lista dos usuários que um usuário está seguindo, precisaremos utilizar `Query` na tabela base com uma condição de chave `PK=<userID>#following`. Depois, você pode usar uma operação `TransactWriteItems` para agrupar várias solicitações e fazer o seguinte:

- Adicionar o usuário A à lista de seguidores do usuário B e, depois, incrementar a contagem de seguidores do usuário B em um.
- Adicionar o usuário B à lista de seguidores do usuário A e, depois, incrementar a contagem de seguidores do usuário A em um.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				

Etapa 4: abordar o padrão de acesso 4 (**`getPostListByUserID`**)

Para obter uma lista das publicações de determinado usuário, precisaremos utilizar `Query` na tabela base com uma condição de chave `PK=<userID>#post`. Um fator importante a observar aqui é que os postIDs de um usuário devem ser incrementais: o segundo valor postID deve ser maior que o primeiro valor postID (já que os usuários querem ver suas publicações de forma ordenada). Você pode fazer isso gerando PostIDs com base em um valor de tempo, como um identificador lexicograficamente classificável universalmente exclusivo (ULID).

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	

Etapa 5: abordar o padrão de acesso 5 (**`getUserLikesByPostID`**)

Para obter uma lista dos usuários que curtiram a publicação de determinado usuário, precisaremos utilizar `Query` na tabela base com uma condição de chave `PK=<postID>#likelist`. Essa abordagem é o mesmo padrão que usamos para recuperar as listas de seguidores e seguidos nos padrões de acesso 2 (`getFollowerListByUserID`) e 3 (`getFollowingListByUserID`).

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				

Etapa 6: abordar o padrão de acesso 6 (**`getLikeCountByPostID`**)

Para obter uma contagem de curtidas de determinada publicação, precisaremos realizar uma operação `GetItem` na tabela base com uma condição de chave `PK=<postID>#likecount`. Esse padrão de acesso pode causar problemas de controle de utilização sempre que um usuário com muitos seguidores (como uma celebridade) criar uma publicação, pois a limitação ocorre quando o throughput de uma partição excede 1.000 WCU por segundo. Esse problema não é provocado pelo DynamoDB. Ele só aparece no DynamoDB porque está no final da pilha de software.

Você deve avaliar se é realmente essencial que todos os usuários visualizem a contagem de curtidas simultaneamente ou se isso pode ocorrer gradualmente ao longo do tempo. Em geral, a contagem de curtidas de uma publicação não precisa ser 100% exata imediatamente. Você pode implementar essa estratégia colocando uma fila entre a aplicação e o DynamoDB para que as atualizações ocorram periodicamente.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			

Etapa 7: abordar o padrão de acesso 7 (**`getTimelineByUserID`**)

Para obter a linha do tempo de determinado usuário, precisaremos realizar uma operação `Query` na tabela base com uma condição de chave `PK=<userID>#timeline`. Vamos considerar uma situação em que os seguidores de um usuário precisam ver a publicação dele de forma síncrona. Toda vez que um usuário escreve uma publicação, sua lista de seguidores é lida e seu ID de usuário e postID são inseridos lentamente na chave da linha do tempo de todos os respectivos seguidores. Em seguida, quando a aplicação for iniciada, você poderá ler a chave da linha do tempo com a operação `Query` e preencher a tela da linha do tempo com uma combinação de UserID e PostID usando a operação `BatchGetItem` para qualquer item novo. Não é possível ler a linha do tempo com uma chamada de API, mas essa é uma solução mais econômica se as publicações puderem ser editadas com frequência.

Como a linha do tempo é um local que mostra as publicações recentes, precisaremos de um meio para limpar as antigas. Em vez de usar WCU para excluí-las, você pode usar o atributo TTL do DynamoDB para fazer isso gratuitamente.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

Todos os padrões de acesso e a forma como o design do esquema os aborda estão resumidos na tabela abaixo:

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
getUserInfoByUserID	Tabela base	Consulta	PK=<userID>		

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
getFollowerListByUserID	Tabela base	Consulta	PK=<userID>#follower		
getFollowingListByUserID	Tabela base	Consulta	PK=<userID>#following		
getPostListByUserID	Tabela base	Consulta	PK=<userID>#post		
getUserLikesByPostID	Tabela base	Consulta	PK=<postID>#likelist		
getLikeCountByPostID	Tabela base	GetItem	PK=<postID>#likecount		
getTimelineByUserID	Tabela base	Consulta	PK=<userID>#timeline		

Esquema final da rede social

Veja aqui o design do esquema final. Para baixar esse design de esquema como arquivo JSON, consulte DynamoDB Examples no GitHub.

Tabela base:

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

Como usar o NoSQL Workbench com esse design de esquema

Você pode importar esse esquema final para o NoSQL Workbench, uma ferramenta visual que fornece atributos de modelagem de dados, visualização de dados e desenvolvimento de consultas para o DynamoDB, se quiser explorar e editar ainda mais seu novo projeto. Para começar, siga estas etapas:

1. Baixe o NoSQL Workbench. Para ter mais informações, consulte the section called “Baixar”.
2. Baixe o arquivo do esquema JSON listado acima, que já está no formato do modelo NoSQL Workbench.

3. Importe o arquivo do esquema JSON para o NoSQL Workbench. Para ter mais informações, consulte the section called “Importar um modelo existente”.
4. Depois de importar para o NOSQL Workbench, você pode editar o modelo de dados. Para ter mais informações, consulte the section called “Editar um modelo existente”.
5. Para visualizar o modelo de dados, adicionar dados de exemplo ou importar dados de exemplo de um arquivo CSV, use o atributo Visualizador de dados do NoSQL Workbench.

Design de esquema de perfil de jogo no DynamoDB

Caso de uso de negócios de perfil de jogo

Este caso de uso aborda o uso do DynamoDB para armazenar perfis de jogadores em um sistema de jogos. Os usuários (neste caso, os jogadores) precisam criar perfis para poderem interagir com vários jogos modernos, especialmente os on-line. Geralmente, os perfis de jogos incluem:

- Informações básicas, como nome de usuário
- Dados do jogo, como itens e equipamentos
- Registros do jogo, como tarefas e atividades
- Informações sociais, como listas de amigos

Para atender aos requisitos detalhados de acesso à consulta de dados dessa aplicação, as chaves primárias (chave de partição e chave de classificação) usarão nomes genéricos (PK e SK) para que sejam sobrecarregadas com vários tipos de valor, como veremos abaixo.

Os padrões de acesso para este design de esquema são:

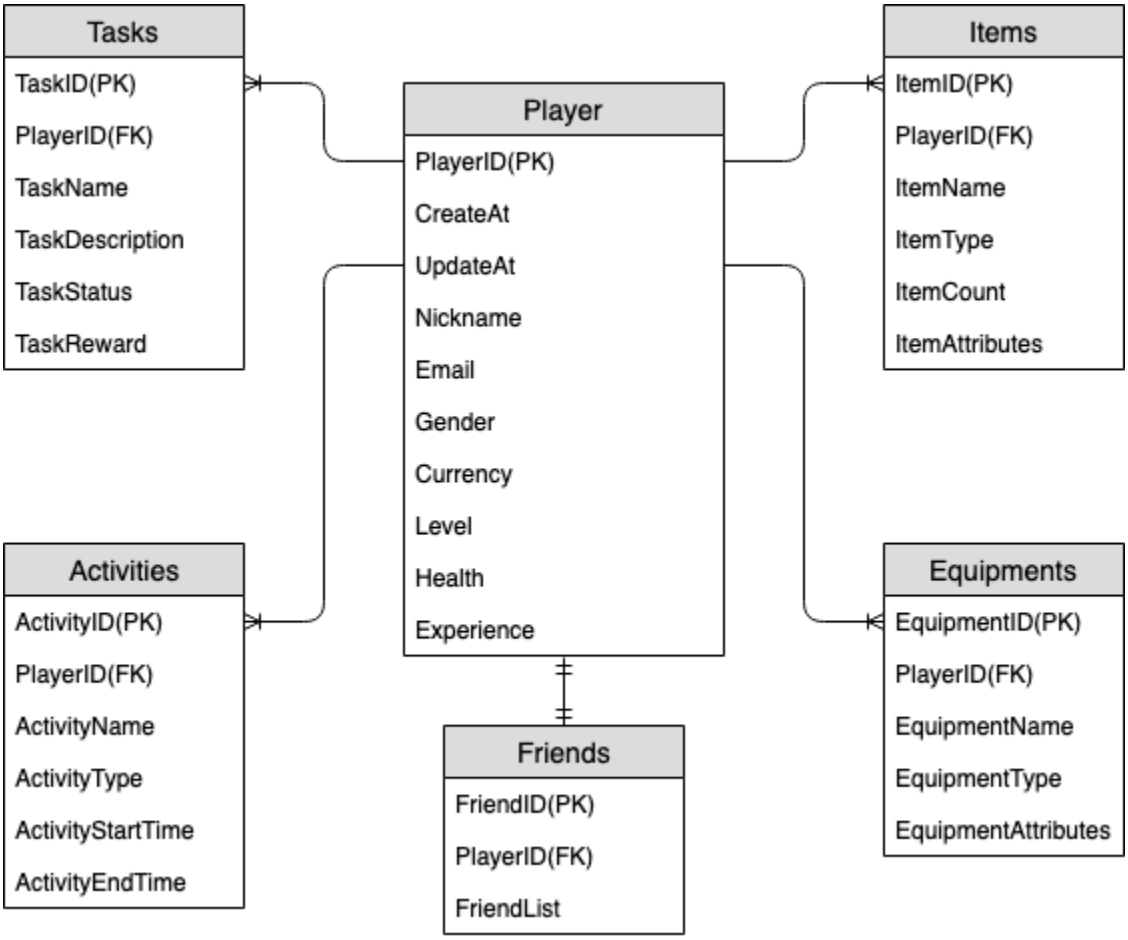
- Obter a lista de amigos de um usuário
- Obter todas as informações de um jogador
- Obter a lista de itens de um usuário
- Obter um item específico da lista de itens do usuário
- Atualizar o personagem de um usuário
- Atualizar a contagem de itens de um usuário

O tamanho do perfil de jogo variará de acordo com o jogo. A compactação de valores de atributos grandes possibilita que eles se ajustem aos limites do item no DynamoDB e reduzam os custos.

Nesse caso, a estratégia de gerenciamento de throughput dependeria de vários fatores, como número de jogadores, número de jogos jogados por segundo e sazonalidade da workload. Normalmente, para um jogo recém-lançado, o número de jogadores e o nível de popularidade são desconhecidos, então começaremos com o modo de throughput sob demanda.

Diagrama de relacionamento de entidades de perfil de jogo

Este é o diagrama de relacionamento de entidades (ERD) que usaremos para o design do esquema do perfil de jogo.



Padrões de acesso do perfil de jogo

Estes são os padrões de acesso que vamos considerar para o design do esquema da rede social.

- `getPlayerFriends`
- `getPlayerAllProfile`
- `getPlayerAllItems`
- `getPlayerSpecificItem`

- `updateCharacterAttributes`
- `updateItemCount`

Evolução do design do esquema do perfil de jogo

No ERD acima, podemos ver que esse é um tipo de modelagem de dados de relação de um para muitos. No DynamoDB, os modelos de dados de um para muitos podem ser organizados em coleções de itens, o que é diferente dos bancos de dados relacionais tradicionais, nos quais várias tabelas são criadas e vinculadas por meio de chaves externas. Coleção de itens é um grupo de itens que compartilham o mesmo valor de chave de partição, mas têm valores de chave de classificação diferentes. Em uma coleção, cada item tem um valor de chave de classificação exclusivo que o distingue dos demais. Com isso em mente, vamos usar o padrão a seguir para os valores `HASH` e `RANGE` referentes a cada tipo de entidade.

Primeiro, usamos nomes genéricos como `PK` e `SK` para armazenar diferentes tipos de entidade na mesma tabela e tornar o modelo preparado para o futuro. A fim de melhorar a legibilidade, podemos incluir prefixos para indicar o tipo de dados ou incluir um atributo arbitrário chamado `Entity_type` ou `Type`. No presente exemplo, usamos uma string que começa com `player` para armazenar `player_ID` como `PK`, usamos `entity name#` como prefixo de `SK` e adicionamos um atributo `Type` para indicar o tipo de entidade desse dado. Isso nos permite comportar o armazenamento de mais tipos de entidade no futuro e usar tecnologias avançadas, como sobrecarga de GSI e GSI esparso, para atender a mais padrões de acesso.

Vamos começar a implementar os padrões de acesso. Padrões de acesso como adicionar jogadores e equipamentos podem ser conseguidos por meio da operação `PutItem`; portanto, podemos ignorá-los. Neste documento, vamos nos concentrar nos padrões de acesso típicos listados acima.

Etapa 1: abordar o padrão de acesso 1 (**`getPlayerFriends`**)

Abordamos o padrão de acesso 1 (`getPlayerFriends`) com esta etapa. No nosso design atual, a relação de amizade é simples e o número de amigos no jogo é pequeno. Para simplificar, usamos um tipo de dados de lista para armazenar listas de amigos (modelagem 1:1). Neste design, usamos `GetItem` para satisfazer esse padrão de acesso. Na operação `GetItem`, fornecemos explicitamente a chave de partição e o valor da chave de classificação para obter um item específico.

No entanto, se um jogo tiver um grande número de amigos e os relacionamentos entre eles forem complexos (como amizades bidirecionais com um componente de convite e aceitação), será necessário usar uma relação de muitos para muitos para armazenar cada amigo

individualmente e escalar para um tamanho ilimitado de lista de amigos. Além disso, se a mudança de amizade envolver a operação de vários itens ao mesmo tempo, as transações do DynamoDB podem ser usadas para agrupar várias ações e enviá-las como uma única operação `TransactWriteItems` ou `TransactGetItems` do tipo “tudo ou nada”.

Primary key		Attributes	
Partition key: PK	**Sort key: SK**		
player001	FRIENDS#player001	Type	FriendList
		Friends	[{"M": {"FriendId": {"S":"player002"},"FriendName": {"S":"Alice"}}}, {"M":{"FriendId": {"S":"player003"},"FriendName": {"S":"Bob"}}}]

Etapa 2: abordar os padrões de acesso 2 (**`getPlayerAllProfile`**), 3 (**`getPlayerAllItems`**) e 4 (**`getPlayerSpecificItem`**)

Abordamos os padrões de acesso 2 (`getPlayerAllProfile`), 3 (`getPlayerAllItems`) e 4 (`getPlayerSpecificItem`) com esta etapa. O que esses três padrões de acesso têm em comum é uma consulta de intervalo, que usa a operação `Query`. Dependendo do escopo da consulta, são usadas expressões de condição de chave e de filtro, que normalmente são utilizadas no desenvolvimento prático.

Na operação de consulta, fornecemos um único valor para a chave de partição e obtemos todos os itens com esse valor de chave de partição. O padrão de acesso 2 (`getPlayerAllProfile`) é implementado dessa forma. Opcionalmente, podemos adicionar uma expressão de condição de chave de classificação, que é uma string que determina os itens a serem lidos da tabela. Para implementar o padrão de acesso 3 (`getPlayerAllItems`), adicionamos a condição de chave de classificação begins_with `ITEMS#`. Além disso, para simplificar o desenvolvimento do lado da aplicação, podemos usar expressões de filtro para implementar o padrão de acesso 4 (`getPlayerSpecificItem`).

Veja aqui um exemplo de pseudocódigo usando uma expressão de filtro que filtra itens da categoria `Weapon`:

```
filterExpression: "ItemType = :itemType"
expressionAttributeValues: {":itemType": "Weapon"}
```

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
player001	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}

> **Note**
>
> A expressão de filtro é aplicada depois que a operação é concluída, porém antes que os resultados sejam retornados. Portanto, a operação consome a mesma quantidade de capacidade de leitura, independentemente de haver uma expressão de filtro.

Se o padrão de acesso for consultar um grande conjunto de dados e filtrar uma grande quantidade de dados para manter apenas um pequeno subconjunto de dados, a abordagem apropriada será criar a chave de partição e a chave de classificação do DynamoDB de uma maneira mais eficiente. Por exemplo, no exemplo acima para obter um determinado `ItemType`, se houvesse muitos itens para cada jogador e se consultar um determinado `ItemType` fosse um padrão de acesso típico, seria mais eficiente inserir `ItemType` em SK como uma chave composta. O modelo de dados ficaria assim: `ITEMS#ItemType#ItemId`.

Etapa 3: abordar os padrões de acesso 5 (**`updateCharacterAttributes`**) e 6 (**`updateItemCount`**)

Abordamos os padrões de acesso 5 (`updateCharacterAttributes`) e 6 (`updateItemCount`) com esta etapa. Quando o jogador precisar modificar o personagem, reduzindo a moeda ou modificando a quantidade de uma determinada arma em seus itens, por exemplo, use `UpdateItem` para implementar esses padrões de acesso. Para atualizar a moeda de um jogador, mas garantir que ela nunca fique abaixo do valor mínimo, podemos adicionar uma the section called "Exemplo na CLI" para reduzir o saldo somente se ele for maior ou igual ao valor mínimo. Veja aqui um exemplo de pseudocódigo:

```
UpdateExpression: "SET currency = currency - :amount"
ConditionExpression: "currency >= :minAmount"
```

Primary key		Attributes									
Partition key: PK	Sort key: SK	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
player001	#METADATA #player001	Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500 Updated to 500-Amount	10	80	1000

Ao desenvolver com o DynamoDB e usar contadores atômicos para diminuir o inventário, podemos garantir a idempotência usando o bloqueio positivo. Veja aqui um exemplo de pseudocódigo para contadores atômicos:

```
UpdateExpression: "SET ItemCount = ItemCount - :incr"
expression-attribute-values: '{":incr":{"N":"1"}}'
```

Primary key		Attributes				
Partition key: PK	Sort key: SK	Type	ItemName	ItemType	ItemCount	ItemAttributes
player001	ITEMS#001	Item	Health Potion	Consumable	5 Updated to 4	{"M":{"HP":{"N":"50"}}}

Além disso, em uma situação em que o jogador compra um item com moeda, todo o processo precisa deduzir a moeda e adicionar um item ao mesmo tempo. Podemos usar as transações do DynamoDB para agrupar várias ações e enviá-las como uma única operação tudo ou nada `TransactWriteItems` ou `TransactGetItems`. `TransactWriteItems` é uma operação de gravação síncrona e idempotente que agrupa até 100 ações de gravação em uma única operação tudo ou nada. As ações são concluídas de forma atômica para que todas sejam bem-sucedidas ou nenhuma tenha êxito. As transações ajudam a eliminar o risco de duplicação ou desaparecimento da moeda. Para obter mais informações sobre transações, consulte Exemplo de transações do DynamoDB.

Todos os padrões de acesso e a forma como o design do esquema os aborda estão resumidos na tabela abaixo:

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
getPlayer Friends	Tabela base	GetItem	PK=PlayerID	SK=“FRIENDS#playerID”	
getPlayer AllProfile	Tabela base	Consulta	PK=PlayerID		
getPlayer AllItems	Tabela base	Consulta	PK=PlayerID	SK begins_with “ITEMS#”	
getPlayer SpecificItem	Tabela base	Consulta	PK=PlayerID	SK begins_with “ITEMS#”	filterExpression: "ItemType = :itemType" expressionAttributeValues: { ":itemType": "Weapon" }
updateCharacterAttributes	Tabela base	UpdateItem	PK=PlayerID	SK=“#METADATA#playerID”	UpdateExpression: "SET currency = currency - :amount" ConditionExpression: "currency >= :minAmount"
updateItemCount	Tabela base	UpdateItem	PK=PlayerID	SK =“ITEMS#ItemID”	update-expression: "SET ItemCount

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
					= ItemCount - :incr" expressio n-attribu te-values : '{":incr": {"N":"1"}}'

Esquema final do perfil de jogo

Veja aqui o design do esquema final. Para baixar esse design de esquema como arquivo JSON, consulte DynamoDB Examples no GitHub.

Tabela base:

Primary key		Attributes										
Partition key: PK	Sort key: SK											
player001	#METADATA #player001	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
		Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500	10	80	1000
	ACTIVITY#001	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647475199	Hunting Trip	{"M":{"Gold":{"N":"50"},"XP":{"N":"200"}}}	1647388800	Hunting					
	ACTIVITY#002	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647647999	Mining Adventure	{"M":{"Gold":{"N":"1000"},"XP":{"N":"500"}}}	1647561600	Mining					
	ACTIVITY#003	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647820799	Arena Challenge	{"M":{"Gold":{"N":"2000"},"XP":{"N":"1000"}}}	1647734400	Arena					
	EQUIPMENTS#001	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Sword of the Dragon	Weapon	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}							
	EQUIPMENTS#001EQUIPMENTS#002	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Armor of the Knight	Armor	{"M":{"DEF":{"N":"100"}}}							
	EQUIPMENTS#003	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Ring of the Mage	Accessory	{"M":{"SP":{"N":"50"}}}							
	FRIENDS#player001	Type	FriendList									
		Friends	[{"M":{"FriendId":{"S":"player002"},"FriendName":{"S":"Alice"}}},{"M":{"FriendId":{"S":"player003"},"FriendName":{"S":"Bob"}}}]									
	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}						
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}						
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}						
	TASK#001	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Find the Lost TreasureFind the Lost Treasure	Get clues from a lost adventurer and find the lost treasure.	InProgress	{"M":{"Gold":{"N":"100"},"XP":{"N":"50"}}}						
	TASK#002	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Defeat Magic Monsters	Go to the Magic Forest and defeat three magic monsters.	Completed	{"M":{"Gold":{"N":"200"},"XP":{"N":"100"}}}						
	TASK#003	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Rescue the Princess	Go to the Demon King's Castle and rescue the princess who is being held captive by the Demon King.	Available	{"M":{"Gold":{"N":"500"},"XP":{"N":"200"}}}						

Como usar o NoSQL Workbench com esse design de esquema

Você pode importar esse esquema final para o NoSQL Workbench, uma ferramenta visual que fornece atributos de modelagem de dados, visualização de dados e desenvolvimento de consultas para o DynamoDB, se quiser explorar e editar ainda mais seu novo projeto. Para começar, siga estas etapas:

1. Baixe o NoSQL Workbench. Para ter mais informações, consulte the section called “Baixar”.
2. Baixe o arquivo do esquema JSON listado acima, que já está no formato do modelo NoSQL Workbench.
3. Importe o arquivo do esquema JSON para o NoSQL Workbench. Para ter mais informações, consulte the section called “Importar um modelo existente”.
4. Depois de importar para o NOSQL Workbench, você pode editar o modelo de dados. Para ter mais informações, consulte the section called “Editar um modelo existente”.
5. Para visualizar o modelo de dados, adicionar dados de exemplo ou importar dados de exemplo de um arquivo CSV, use o atributo Visualizador de dados do NoSQL Workbench.

Design do esquema do sistema de gerenciamento de reclamações no DynamoDB

Caso de uso de negócios do sistema de gerenciamento de reclamações

O DynamoDB é um banco de dados adequado para um caso de uso de sistema de gerenciamento de reclamações (ou central de atendimento), pois a maioria dos padrões de acesso associados a eles corresponde a pesquisas transacionais baseadas em chave-valor. Os padrões de acesso típicos nesse cenário seriam:

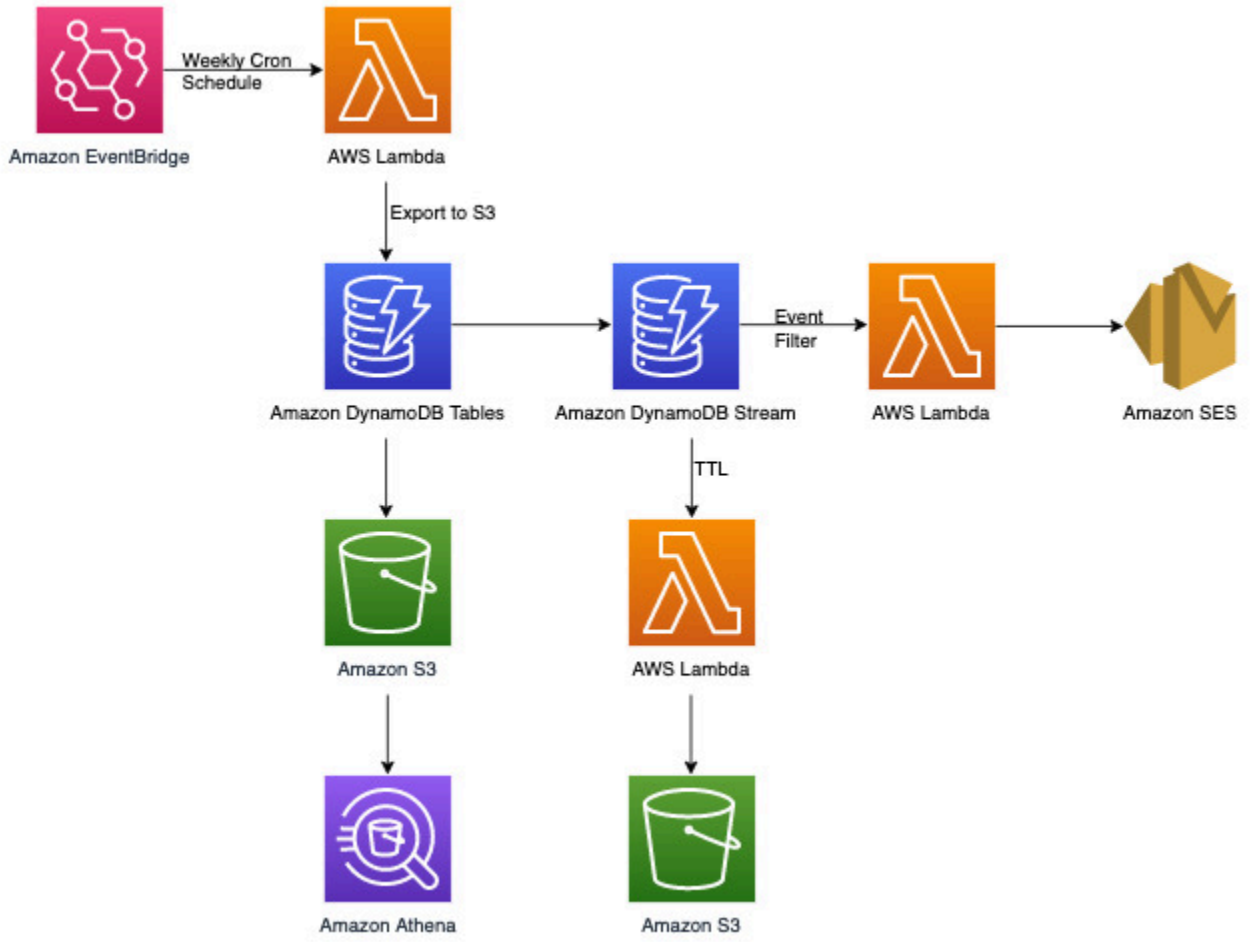
- Criar e atualizar reclamações.
- Escalar uma reclamação.
- Criar e ler comentários sobre uma reclamação.
- Receber todas as reclamações de um cliente.
- Receber todos os comentários de um agente e receber todos os encaminhamentos.

Alguns comentários podem conter anexos que descrevem a reclamação ou a solução. Embora todos esses sejam padrões de acesso de chave-valor, pode haver requisitos adicionais, como

enviar notificações quando um novo comentário é adicionado a uma reclamação ou executar consultas analíticas para descobrir a distribuição da reclamação por severidade (ou performance do agente) por semana. Um requisito adicional relacionado ao gerenciamento do ciclo de vida ou à conformidade seria arquivar os dados da reclamação após três anos de registro em log da reclamação.

Diagrama da arquitetura do sistema de gerenciamento de reclamações

O diagrama a seguir mostra o diagrama de arquitetura do sistema de gerenciamento de reclamações. Este diagrama mostra as diferentes integrações de AWS service (Serviço da AWS) que o sistema de gerenciamento de reclamações usa.



Além dos padrões de acesso transacional de chave-valor que trataremos posteriormente na seção de modelagem de dados do DynamoDB, temos três requisitos não transacionais. O diagrama de arquitetura acima pode ser dividido nestes três fluxos de trabalho:

1. Enviar uma notificação quando um novo comentário for adicionado a uma reclamação.
2. Executar consultas analíticas em dados semanais.
3. Arquivar dados com mais de três anos.

Vamos analisar mais profundamente cada um deles.

Enviar uma notificação quando um novo comentário for adicionado a uma reclamação.

Podemos usar o fluxo de trabalho abaixo para cumprir esse requisito:

DynamoDB Streams é um mecanismo de captura de dados de alterações para registrar todas as atividades de gravação em suas tabelas do DynamoDB. Você pode configurar funções do Lambda para acionar algumas ou todas essas alterações. Um filtro de eventos pode ser configurado em acionadores do Lambda para filtrar eventos que não são relevantes para o caso de uso. Nesse caso, podemos usar um filtro para acionar o Lambda somente quando um novo comentário é adicionado e enviar uma notificação aos IDs de e-mail relevantes, que podem ser obtidos no AWS Secrets Manager ou em qualquer outro repositório de credenciais.

Executar consultas analíticas em dados semanais.

O DynamoDB é adequado para workloads que se concentram principalmente no processamento transacional on-line (OLTP). Quanto aos 10% a 20% restantes dos padrões de acesso com requisitos analíticos, os dados podem ser exportados para o S3 com o atributo Exportar para o Amazon S3 sem impacto no tráfego em tempo real na tabela do DynamoDB. Veja este fluxo de trabalho abaixo:

O Amazon EventBridge pode ser usado para acionar AWS Lambda na programação: permite configurar uma expressão cron para que a invocação do Lambda ocorra periodicamente. O Lambda pode invocar a chamada de API `ExportToS3` e armazenar dados do DynamoDB no S3. Esses dados do S3 podem então ser acessados por um mecanismo SQL como o Amazon Athena para executar consultas analíticas nos dados do DynamoDB sem afetar a workload transacional em tempo real na tabela. Um exemplo de consulta do Athena para encontrar o número de reclamações por nível de gravidade teria a aparência abaixo:

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

Isso gera o seguinte resultado da consulta do Athena:

Results (3)

Search rows

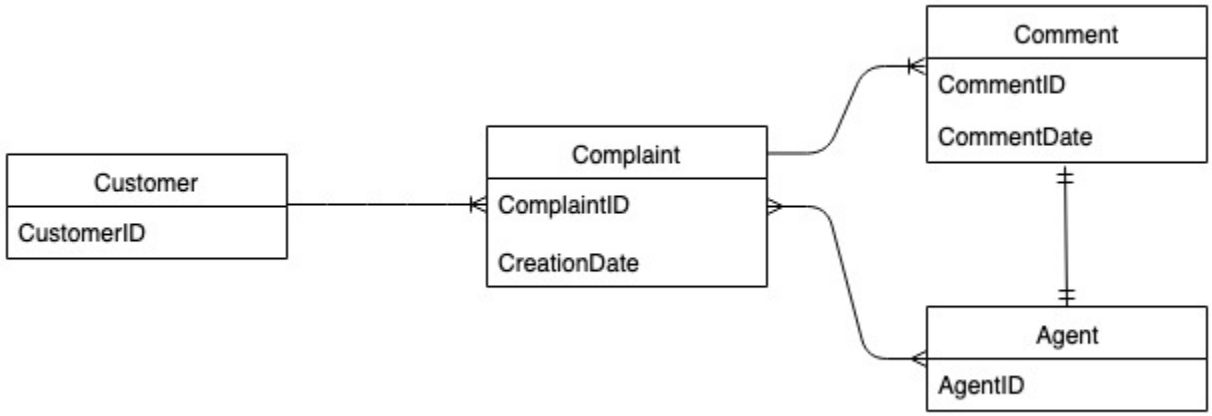
#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

Arquivar dados com mais de três anos.

Você pode utilizar o atributo vida útil (TTL) do DynamoDB para excluir dados obsoletos da tabela do DynamoDB sem custo adicional (exceto no caso de réplicas de tabelas globais para a versão 2019.11.21 (atual), em que as exclusões TTL replicadas em outras regiões consomem capacidade de gravação). Esses dados aparecem e podem ser consumidos no DynamoDB Streams para serem arquivados no Amazon S3. O fluxo de trabalho referente a esse requisito é o seguinte:

Diagrama de relacionamento de entidades do sistema de gerenciamento de reclamações

Este é o diagrama de relacionamento de entidades (ERD) que usaremos para o design do esquema de gerenciamento de reclamações.



Padrões de acesso ao sistema de gerenciamento de reclamações

Estes são os padrões de acesso que vamos considerar para o design do esquema de gerenciamento de reclamações.

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID

4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID (ordem da mais recente para a mais antiga)
13. getCommentsByAgentID (entre duas datas)

Evolução do design do esquema do sistema de gerenciamento de reclamações

Como esse é um sistema de gerenciamento de reclamações, a maioria dos padrões de acesso gira em torno de uma reclamação como entidade primária. O `ComplaintID`, por ser altamente cardinal, garantirá uma distribuição uniforme dos dados nas partições subjacentes e também é o critério de pesquisa mais comum para nossos padrões de acesso identificados. Portanto, `ComplaintID` é um bom candidato à chave de partição nesse conjunto de dados.

Etapa 1: abordar os padrões de acesso 1 (**`createComplaint`**), 2 (**`updateComplaint`**), 3 (**`updateSeveritybyComplaintID`**) e 4 (**`getComplaintByComplaintID`**)

Podemos usar uma chave de classificação genérica chamada “metadados” (ou “AA”) para armazenar informações específicas da reclamação, como `CustomerID`, `State`, `Severity` e `CreationDate`. Usamos operações singleton com `PK=ComplaintID` e `SK=“metadata”` para fazer o seguinte:

1. `PutItem`, para criar uma reclamação.
2. `UpdateItem`, para atualizar a gravidade ou outros campos nos metadados da reclamação.
3. `GetItem`, para buscar metadados para a reclamação.

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
Complaint1321	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	assigned	2023-05-10T15:58:00	P2	<Complaint Description>

Etapa 2: abordar o padrão de acesso 5 (**`addCommentByComplaintID`**)

Esse padrão de acesso exige um modelo de relação de um para muitos entre uma reclamação e comentários sobre a reclamação. Utilizaremos a técnica particionamento vertical aqui para usar uma chave de classificação e criar uma coleção de itens com diferentes tipos de dados. Quando observarmos os padrões de acesso 6 (`getAllCommentsByComplaintID`) e 7 (`getLatestCommentByComplaintID`), sabemos que os comentários precisarão ser classificados por hora. Também podemos receber vários comentários ao mesmo tempo para que possamos usar a técnica chave de classificação composta para acrescentar tempo e `CommentID` ao atributo de chave de classificação.

Uma opção para lidar com essas possíveis colisões de comentários seria aumentar o detalhamento do carimbo de data/hora ou adicionar um número incremental como sufixo em vez de usar `Comment_ID`. Nesse caso, prefixaremos o valor da chave de classificação dos itens correspondentes aos comentários com “comm#” para permitir operações baseadas em intervalos.

Também precisamos garantir que o `currentState` nos metadados da reclamação reflitam o estado quando um novo comentário for adicionado. Adicionar um comentário pode indicar que a reclamação foi atribuída a um agente ou resolvida etc. Para agrupar a adição de comentários e a atualização do estado atual nos metadados da reclamação, de uma maneira “tudo ou nada”, usaremos a API TransactWriteItems. O estado da tabela resultante agora tem a seguinte aparência:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID
		comm3	2023-05-10T16:00:00	investigating	<Comment text>	AgentB
	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	investigating	2023-05-10T15:58:00	P2	<Complaint Description>

Vamos adicionar mais alguns dados na tabela, e também `ComplaintID` como um campo separado de nosso PK, para preparar o modelo para o futuro, caso precisemos de outros índices em `ComplaintID`. Observe também que alguns comentários podem ter anexos, os quais serão armazenados no Amazon Simple Storage Service e só terão as respectivas referências ou URLs mantidas no DynamoDB. É prática recomendada manter o banco de dados transacional o mais enxuto possível para otimizar o custo e a performance. O arquivo de dados agora é semelhante a:

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Etapa 3: abordar os padrões de acesso 6 (**`getAllCommentsByComplaintID`**) e 7 (**`getLatestCommentByComplaintID`**)

Para receber todos os comentários de uma reclamação, podemos usar a operação `query` com a condição `begins_with` na chave de classificação. Em vez de consumir capacidade adicional de leitura para ler a entrada de metadados e, depois, precisar lidar com a sobrecarga de filtrar os resultados relevantes, ter uma condição de chave de classificação como essa nos ajuda a ler apenas o que precisamos. Por exemplo, uma operação de consulta com `PK=Complaint123` e `SK` begins_with `comm#` exibiria o seguinte ao ignorar a entrada de metadados:

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Como precisamos do comentário mais recente para uma reclamação no padrão 7 (`getLatestCommentByComplaintID`), vamos usar dois parâmetros de consulta adicionais:

1. `ScanIndexForward` deve ser definido como False para receber resultados classificados em ordem decrescente.
2. `Limit` deve ser definido como 1 para receber o comentário mais recente (apenas um)..

Tal como no padrão de acesso 6 (`getAllCommentsByComplaintID`), ignoramos a entrada de metadados usando `begins_with comm#` como a condição da chave de classificação. Agora você pode executar o padrão de acesso 7 nesse design usando a operação de consulta com `PK=Complaint123` e `SK=begins_with comm#`, `ScanIndexForward=False` e `Limit` 1. O seguinte item direcionado será retornado como resultado:

Partition key: PK	Sort key: SK	Attributes					
	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
Complaint123	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Vamos adicionar mais dados fictícios à tabela.

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text		
		comm4	2022-12-31T19:32:00	waiting	<comm text>		
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>

Etapa 4: abordar os padrões de acesso 8 (**`getAComplaintbyCustomerIDAndComplaintID`**) e 9 (**`getAllComplaintsByCustomerID`**)

Os padrões de acesso 8 (`getAComplaintbyCustomerIDAndComplaintID`) e 9 (`getAllComplaintsByCustomerID`) introduzem um novo critério de pesquisa: `CustomerID`. Buscá-lo na tabela existente exige uma `Scan` cara, pois é necessário ler todos os dados e, depois, filtrar os itens relevantes para o `CustomerID` em questão. Podemos tornar essa pesquisa mais

eficiente criando um índice secundário global (GSI) com `CustomerID` como a chave de partição. Tendo em mente a relação de um para muitos entre o cliente e as reclamações, bem como o padrão de acesso 9 (`getAllComplaintsByCustomerID`), `ComplaintID` seria o candidato certo para a chave de classificação.

Os dados no GSI ficariam assim:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Um exemplo de consulta nesse GSI para o padrão de acesso 8 (`getAComplaintbyCustomerIDAndComplaintID`) seria: `customer_id=custXYZ`, `sort key=Complaint1321`. O resultado seria:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
custXYZ							
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Para receber todas as reclamações de um cliente sobre o padrão de acesso 9 (`getAllComplaintsByCustomerID`), a consulta no GSI seria: `customer_id=custXYZ` como a condição da chave de partição. O resultado seria:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Etapa 5: abordar o padrão de acesso 10 (**`escalateComplaintByComplaintID`**)

Esse acesso apresenta o aspecto de encaminhamento. Para encaminhar uma reclamação, podemos usar `UpdateItem` para adicionar atributos como `escalated_to` e `escalation_time` ao item de metadados da reclamação existente. O DynamoDB fornece um design de esquema flexível, o que significa que um conjunto de atributos que não são de chave pode ser uniforme ou distinto em diferentes itens. Veja o seguinte exemplo:

`UpdateItem with PK=Complaint1444, SK=metadata`

Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07

Etapa 6: abordar os padrões de acesso 11 (**`getAllEscalatedComplaints`**) e 12 (**`getEscalatedComplaintsByAgentID`**)

Espera-se que apenas algumas reclamações sejam encaminhadas de todo o conjunto de dados. Portanto, criar um índice sobre os atributos relacionados ao encaminhamento resultaria em pesquisas eficientes e em um armazenamento GSI econômico. Podemos fazer isso utilizando a técnica índice esparso. O GSI com uma chave de partição como `escalated_to` e uma chave de classificação como `escalation_time` ficaria desta forma:

Primary key		Attributes							
Partition key: escalated_to	**Sort key: escalation_time**								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Para obter todas as reclamações encaminhadas ao padrão de acesso 11 (`getAllEscalatedComplaints`), nós simplesmente verificamos esse GSI. Observe que essa verificação será eficiente e econômica devido ao tamanho do GSI. Para receber reclamações encaminhadas a um agente específico [padrão de acesso 12 (`getEscalatedComplaintsByAgentID`)], a chave de partição seria `escalated_to=agentID`, e definimos `ScanIndexForward` como `False` para classificação do mais novo para o mais antigo.

Etapa 7: abordar o padrão de acesso 13 (**`getCommentsByAgentID`**)

Para o último padrão de acesso, precisamos pesquisar uma nova dimensão: `AgentID`. Também precisamos de uma classificação baseada em tempo para ler os comentários entre duas datas. Portanto, criamos um GSI com `agent_id` como a chave de partição e `comm_date` como a chave de classificação. Os dados nesse GSI terão a seguinte aparência:

Primary key		Attributes					
Partition key: agentID	**Sort key: comm_date**						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Um exemplo de consulta sobre esse GSI seria `partition key agentID=AgentA` e `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`, cujo resultado é:

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Todos os padrões de acesso e a forma como o design do esquema os aborda estão resumidos na tabela abaixo:

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
createComplaint	Tabela base	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	Tabela base	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	Tabela base	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByComplaintID	Tabela base	GetItem	PK=complaint_id	SK=metadata	

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
addCommentByComplaintID	Tabela base	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#comm_date#comm_id	
getAllCommentsByComplaintID	Tabela base	Consulta	PK=complaint_id	SK begins_with "comm#"	
getLatestCommentByComplaintID	Tabela base	Consulta	PK=complaint_id	SK begins_with "comm#"	scan_index_forward=False, Limit 1
getAComplaintbyCustomerIDAndComplaintID	Customer_complaint_GSI	Consulta	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Consulta	customer_id=customer_id	N/D	
escalateComplaintByComplaintID	Tabela base	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedComplaints	Escalations_GSI	Verificar	N/D	N/D	

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
getEscalatedComplaintsByAgentID (ordem da mais recente para a mais antiga)	Escalations_GSI	Consulta	escalated_to=agent_id	N/D	scan_index_forward=False
getCommentsByAgentID (entre duas datas)	Agents_Comments_GSI	Consulta	agent_id=agent_id	SK entre (data1, data2)	

Esquema final do sistema de gerenciamento de reclamações

Veja aqui os designs do esquema final. Para baixar esse design de esquema como arquivo JSON, consulte DynamoDB Examples no GitHub.

Tabela base

Primary key		Attributes							
Partition key: PK	**Sort key: SK**								
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA			
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>		
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB			
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>		

Customer_Complaint_GSI

Primary key		Attributes							
Partition key: customer_id	**Sort key: complaint_id**								
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>		
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>		
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00

Escalations_GSI

Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Agents_Comments_GSI

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Como usar o NoSQL Workbench com esse design de esquema

Você pode importar esse esquema final para o NoSQL Workbench, uma ferramenta visual que fornece atributos de modelagem de dados, visualização de dados e desenvolvimento de consultas para o DynamoDB, se quiser explorar e editar ainda mais seu novo projeto. Para começar, siga estas etapas:

1. Baixe o NoSQL Workbench. Para ter mais informações, consulte the section called “Baixar”.
2. Baixe o arquivo do esquema JSON listado acima, que já está no formato do modelo NoSQL Workbench.
3. Importe o arquivo do esquema JSON para o NoSQL Workbench. Para ter mais informações, consulte the section called “Importar um modelo existente”.
4. Depois de importar para o NOSQL Workbench, você pode editar o modelo de dados. Para ter mais informações, consulte the section called “Editar um modelo existente”.
5. Para visualizar o modelo de dados, adicionar dados de exemplo ou importar dados de exemplo de um arquivo CSV, use o atributo Visualizador de dados do NoSQL Workbench.

Design de esquema de pagamentos recorrentes no DynamoDB

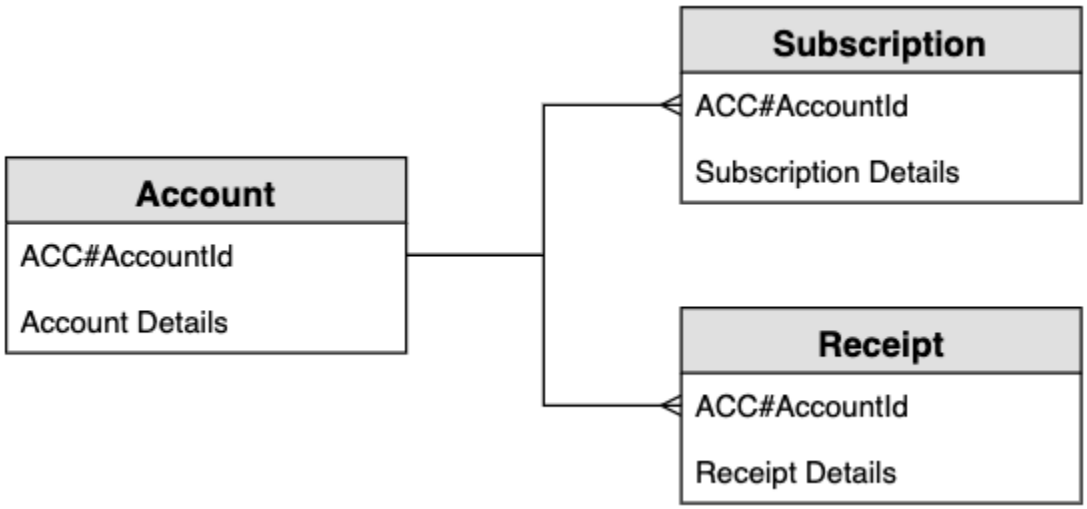
Caso de uso de negócios de pagamentos recorrentes

Esse caso de uso fala sobre o uso do DynamoDB para implementar um sistema de pagamentos recorrentes. O modelo de dados tem as seguintes entidades: contas, assinaturas e recibos. Os detalhes do nosso caso de uso são os seguintes:

- Cada conta pode ter várias assinaturas.
- A assinatura tem uma `NextPaymentDate` quando o próximo pagamento precisar ser processado e uma `NextReminderDate` quando um lembrete por e-mail é enviado ao cliente.
- Há um item para a assinatura que é armazenado e atualizado quando o pagamento é processado (o tamanho médio do item é de cerca de 1 KB e o throughput depende do número de contas e assinaturas).
- O processador de pagamentos também criará um recibo como parte do processo, que é armazenado na tabela e está definido para expirar após um período usando um atributo TTL.

Diagrama de relacionamento de entidades de pagamentos recorrentes

Este é o diagrama de relacionamento de entidades (ERD) que usaremos para o design do esquema de pagamentos recorrentes.



Padrões de acesso recorrentes do sistema de pagamentos recorrentes

Estes são os padrões de acesso que vamos considerar para o design de esquema do sistema de pagamentos recorrentes.

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

Design de esquema de pagamentos recorrentes

Os nomes genéricos PK e SK são usados para atributos de chave com o objetivo de permitir o armazenamento de diferentes tipos de entidade na mesma tabela, como entidades de conta, assinatura e recibo. O usuário primeiro cria uma assinatura, na qual ele concorda em pagar uma quantia no mesmo dia de cada mês em troca de um produto. Ele pode escolher em qual dia do mês processar o pagamento. Também há um lembrete que será enviado antes do processamento do pagamento. A aplicação funciona com dois trabalhos em lote que são executados todos os dias: um trabalho em lote envia lembretes de vencimento naquele dia e o outro processa todos os pagamentos devidos naquele dia.

Etapa 1: abordar o padrão de acesso 1 (**`createSubscription`**)

O padrão de acesso 1 (`createSubscription`) é usado para criar inicialmente a assinatura, e os detalhes, incluindo SKU, `NextPaymentDate`, `NextReminderDate` e `PaymentDetails`, são definidos. Essa etapa mostra o estado da tabela para uma única conta com uma única assinatura. Pode haver várias assinaturas na coleção de itens, então essa é uma relação de um para muitos.

Primary key		Attributes									
Partition key: PK	**Sort key: SK**										
		Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	1970-01-01T00:00:00.000Z	2023-05-28	1970-01-01T00:00:00.000Z	2023-05-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Etapa 2: abordar os padrões de acesso 2 (**`createReceipt`**) e 3 (**`updateSubscription`**)

O padrão de acesso 2 (`createReceipt`) é usado para criar o item de recibo. Depois que o pagamento for processado a cada mês, o processador de pagamentos enviará um recibo de volta à tabela base. Pode haver vários recibos na coleção de itens, então essa é uma relação de um para muitos. O processador de pagamentos também atualizará o item de assinatura [padrão de acesso 3

(`updateSubscription`)] para atualizar para `NextReminderDate` ou `NextPaymentDate` para o próximo mês.

Primary key		Attributes									
Partition key: PK	**Sort key: SK**										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Etapa 3: abordar o padrão de acesso 4 (**`getDueRemindersByDate`**)

A aplicação processa lembretes para o pagamento em lotes para o dia atual. Portanto, a aplicação precisa acessar as assinaturas em uma dimensão diferente: data, em vez de conta. Esse é um bom caso de uso para um índice secundário global (GSI). Nesta etapa, adicionamos o índice `GSI-1`, que usa `NextReminderDate` como a chave de partição do GSI. Não precisamos replicar todos os itens. Esse GSI é um índice esparso, e os itens de recibo não são replicados. Também não precisamos projetar todos os atributos, apenas incluir um subconjunto dos atributos. A imagem abaixo mostra o esquema de `GSI-1` e fornece as informações necessárias para que a aplicação envie o e-mail de lembrete.

Primary key		Attributes				
Partition key: NextReminderDate	**Sort key: LastReminderDate**					
2023-06-21	2023-05-21T14:15:39.247Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

Etapa 4: abordar o padrão de acesso 5 (**`getDuePaymentsByDate`**)

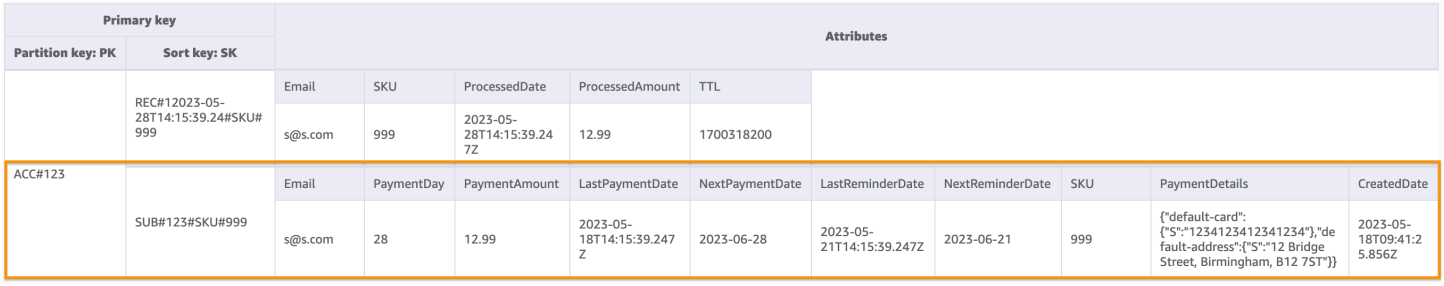
A aplicação processa os pagamentos em lotes para o dia atual da mesma forma que faz para lembretes. Nesta etapa, adicionamos `GSI-2`, e ela usa a `NextPaymentDate` como a chave de partição do GSI. Não precisamos replicar todos os itens. Esse GSI é um índice esparso, e os itens de recibo não são replicados. A imagem abaixo mostra o esquema de `GSI-2`.

Primary key		Attributes						
Partition key: NextPaymentDate	**Sort key: LastPaymentDate**							
2023-06-28	2023-05-18T14:15:39.247Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
		ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Etapa 5: abordar os padrões de acesso 6 (**`getSubscriptionsByAccount`**) e 7 (**`getReceiptsByAccount`**)

A aplicação pode recuperar todas as assinaturas de uma conta usando uma consulta na tabela base que tem como alvo o identificador da conta (o PK) e usa o operador de intervalo para obter

todos os itens em que o SK começa com “SUB#”. A aplicação também pode usar a mesma estrutura de consulta para recuperar todos os recibos usando um operador de intervalo e obter todos os itens em que o SK começa com “REC#”. Isso nos permite cumprir os padrões de acesso 6 (`getSubscriptionsByAccount`) e 7 (`getReceiptsByAccount`). A aplicação usa esses padrões de acesso para que o usuário possa ver suas assinaturas atuais e os recibos anteriores dos últimos seis meses. Não há nenhuma alteração no esquema da tabela nesta etapa e podemos ver abaixo como consideramos apenas os itens de assinatura no padrão de acesso 6 (`getSubscriptionsByAccount`).



Primary key		Attributes									
Partition key: PK	Sort key: SK										
	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
ACC#123		Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
	SUB#123#SKU#999	s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Todos os padrões de acesso e a forma como o design do esquema os aborda estão resumidos na tabela abaixo:

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação
createSubscription	Tabela base	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	Tabela base	PutItem	ACC#account_id	REC#<ReceiptDate>#SKU<SKUID>
updateSubscription	Tabela base	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Consulta	<NextReminderDate>	

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação
getDuePay mentsByDate	GSI-2	Consulta	<NextPaym entDate>	
getSubscr iptionsByAccount	Tabela base	Consulta	ACC#account_id	SK begins_with “SUB#”
getReceip tsByAccount	Tabela base	Consulta	ACC#account_id	SK begins_with “REC#”

Esquema final de pagamentos recorrentes

Veja aqui os designs do esquema final. Para baixar esse design de esquema como arquivo JSON, consulte DynamoDB Examples no GitHub.

Tabela base

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

GSI-1

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate					
2023-06-21	2023-05-21T14:15:39.247Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

GSI-2

Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate							
2023-06-28	2023-05-18T14:15:39.247Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
		ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Como usar o NoSQL Workbench com esse design de esquema

Você pode importar esse esquema final para o NoSQL Workbench, uma ferramenta visual que fornece atributos de modelagem de dados, visualização de dados e desenvolvimento de consultas para o DynamoDB, se quiser explorar e editar ainda mais seu novo projeto. Para começar, siga estas etapas:

1. Baixe o NoSQL Workbench. Para ter mais informações, consulte the section called “Baixar”.
2. Baixe o arquivo do esquema JSON listado acima, que já está no formato do modelo NoSQL Workbench.
3. Importe o arquivo do esquema JSON para o NoSQL Workbench. Para ter mais informações, consulte the section called “Importar um modelo existente”.
4. Depois de importar para o NOSQL Workbench, você pode editar o modelo de dados. Para ter mais informações, consulte the section called “Editar um modelo existente”.
5. Para visualizar o modelo de dados, adicionar dados de exemplo ou importar dados de exemplo de um arquivo CSV, use o atributo Visualizador de dados do NoSQL Workbench.

Monitoramento das atualizações de status do dispositivo no DynamoDB

Esse caso de uso aborda o uso do DynamoDB para monitorar atualizações de status de dispositivos (ou alterações no estado dos dispositivos) no DynamoDB.

Caso de uso

Em casos de uso de IoT (uma fábrica inteligente, por exemplo), muitos dispositivos precisam ser monitorados pelos operadores e eles enviam periodicamente seus status ou logs para um sistema de monitoramento. Quando há um problema com um dispositivo, o status dele muda de normal para aviso. Existem diferentes níveis ou status de log, dependendo da gravidade e do tipo de comportamento anormal no dispositivo. O sistema então designa um operador para verificar o dispositivo e ele pode encaminhar o problema ao supervisor, se necessário.

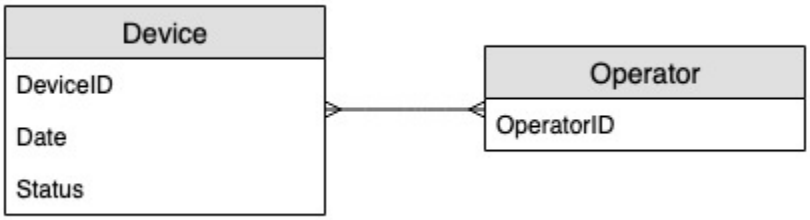
Alguns padrões de acesso típicos desse sistema incluem:

- Criar entrada de log para um dispositivo
- Obter todos os logs de um estado específico do dispositivo, mostrando primeiro os logs mais recentes
- Obter todos os logs de determinado operador entre duas datas

- Obter todos os logs encaminhados a determinado supervisor
- Obter todos os logs encaminhados com um estado de dispositivo específico para determinado supervisor
- Obter todos os logs encaminhados com um estado de dispositivo específico para determinado supervisor em uma data específica

Diagrama de relacionamento de entidades

Este é o diagrama de relacionamento de entidades (ERD) que usaremos para monitorar atualizações de status de dispositivos.



Padrões de acesso

Esses são os padrões de acesso que vamos considerar para monitorar atualizações de status de dispositivos.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

Evolução do design do esquema

Etapa 1: abordar os padrões de acesso 1 (**`createLogEntryForSpecificDevice`**) e 2 (**`getLogsForSpecificDevice`**)

A unidade de escala de um sistema de rastreamento de dispositivos seriam dispositivos individuais. Nesse sistema, um `deviceID` identifica de forma exclusiva um dispositivo. Isso tornar `deviceID` um bom candidato para a chave de partição. Cada dispositivo envia informações ao sistema de rastreamento periodicamente (digamos, a cada cinco minutos, mais ou menos). Essa ordem torna a data um critério lógico de classificação e, portanto, a chave de classificação. Os dados de amostra, nesse caso, ficariam mais ou menos assim:

Primary key		Attributes
Partition key: DeviceID	**Sort key: State#Date**	
d#12345	2020-04-24T14:40:00	State
		WARNING1
	2020-04-24T14:45:00	State
		WARNING1
	2020-04-24T14:50:00	State
		WARNING1
	2020-04-24T14:55:00	State
		NORMAL
d#54321	2020-04-11T05:50:00	State
		WARNING3
	2020-04-11T05:55:00	State
		WARNING3
	2020-04-11T06:00:00	State
		NORMAL
	2020-04-11T09:25:00	State
		WARNING2
	2020-04-11T09:30:00	State
		NORMAL
d#11223	2020-04-27T16:10:00	State
		WARNING4
	2020-04-27T16:15:00	State
		WARNING4

Para buscar entradas de log para um dispositivo específico, podemos realizar uma operação de consulta com a chave de partição `DeviceID="d#12345"`.

Etapa 2: abordar o padrão de acesso 3 (**`getWarningLogsForSpecificDevice`**)

Como `State` é um atributo não chave, abordar o padrão de acesso 3 com o esquema atual exigiria uma expressão de filtro. No DynamoDB, as expressões de filtro são aplicadas depois que os dados são lidos usando expressões de condição de chave. Por exemplo, se fôssemos obter os logs de aviso para `d#12345`, a operação de consulta com a chave de partição `DeviceID="d#12345"` leria quatro itens da tabela acima e, depois, filtraria o único item sem o status aviso. Essa abordagem não é eficiente em grande escala. Uma expressão de filtro pode ser uma boa maneira de excluir itens que são consultados se a proporção de itens excluídos for baixa ou se a consulta for realizada com pouca frequência. No entanto, nos casos em que muitos itens são recuperados de uma tabela e a maioria dos itens é excluída após a filtragem, podemos continuar desenvolvendo nosso design de tabela para que funcione com maior eficiência.

Vamos mudar a forma de lidar com esse padrão de acesso usando chaves de classificação compostas. É possível importar dados de amostra do DeviceStateLog_3.json, onde a chave de classificação é alterada para `State#Date`. Essa chave de classificação é a composição dos atributos `State`, `#` e `Date`. Nesse exemplo, `#` é usado como um delimitador. Os dados agora ficam mais ou menos assim:

Primary key	
Partition key: DeviceID	**Sort key: State#Date**
d#12345	NORMAL#2020-04-24T14:55:00
	WARNING1#2020-04-24T14:40:00
	WARNING1#2020-04-24T14:45:00
	WARNING1#2020-04-24T14:50:00

Para obter somente logs de aviso de um dispositivo, a consulta se torna mais direcionada com esse esquema. A condição de chave para a consulta usa a chave de partição `DeviceID="d#12345"` e a chave de classificação `State#Date begins_with “WARNING”`. Essa consulta lerá somente os três itens relevantes com o estado aviso.

Etapa 3: abordar o padrão de acesso 4 (**`getLogsForOperatorBetweenTwoDates`**)

É possível importar DeviceStateLog_4.json, onde o atributo `Operator` foi adicionado à tabela `DeviceStateLog` com dados de exemplo.

Primary key		Attributes		
Partition key: DeviceID	**Sort key: State#Date**			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State
		Liz	2020-04-24T14:55:00	NORMAL
	WARNING1#2020-04-24T14:40:00	Operator	Date	State
		Liz	2020-04-24T14:40:00	WARNING1
	WARNING1#2020-04-24T14:45:00	Operator	Date	State
		Liz	2020-04-24T14:45:00	WARNING1
	WARNING1#2020-04-24T14:50:00	Operator	Date	State
		Liz	2020-04-24T14:50:00	WARNING1
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State
		Liz	2020-04-11T06:00:00	NORMAL
	NORMAL#2020-04-11T09:30:00	Operator	Date	State
		Sue	2020-04-11T09:30:00	NORMAL
	WARNING2#2020-04-11T09:25:00	Operator	Date	State
		Sue	2020-04-11T09:25:00	WARNING2
	WARNING3#2020-04-11T05:50:00	Operator	Date	State
		Sue	2020-04-11T05:50:00	WARNING3
	WARNING3#2020-04-11T05:55:00	Operator	Date	State
		Liz	2020-04-11T05:55:00	WARNING3
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State
		Sue	2020-04-27T16:10:00	WARNING4
	WARNING4#2020-04-27T16:15:00	Operator	Date	State
		Sue	2020-04-27T16:15:00	WARNING4

Como no momento `Operator` não é uma chave de partição, não há como realizar uma pesquisa direta de chave-valor nessa tabela com base em `OperatorID`. Precisaremos criar uma coleção de

itens com um índice secundário global em `OperatorID`. O padrão de acesso requer uma pesquisa com base em datas, então Data é o atributo da chave de classificação do índice secundário global (GSI). É assim que o GSI se parece agora:

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Para o padrão de acesso 4 (`getLogsForOperatorBetweenTwoDates`), é possível consultar esse GSI com a chave de partição `OperatorID=Liz` e a chave de classificação `Date` entre `2020-04-11T05:58:00` e `2020-04-24T14:50:00`.

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
Liz	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Etapa 4: abordar os padrões de acesso 5 (**`getEscalatedLogsForSupervisor`**), 6 (**`getEscalatedLogsWithSpecificStatusForSupervisor`**) e 7 (**`getEscalatedLogsWithSpecificStatusForSupervisorForDate`**)

Usaremos um índice esparso para abordar esses padrões de acesso.

Os índices secundários globais são esparsos por padrão; portanto, somente os itens na tabela base que contêm atributos de chave primária do índice realmente aparecerão no índice. Essa é outra forma de excluir itens que não são relevantes para o padrão de acesso que está sendo modelado.

É possível importar DeviceStateLog_6.json, onde o atributo `EscalatedTo` foi adicionado à tabela `DeviceStateLog` com dados de exemplo. Conforme mencionado anteriormente, nem todos os logs são encaminhados para um supervisor.

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

Agora você pode criar um novo GSI em que `EscalatedTo` é a chave de partição e `State#Date` é a chave de classificação. Observe que somente itens que têm ambos os atributos `EscalatedTo` e `State#Date` aparecem no índice.

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
		DeviceID	Operator	Date	State
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue	2020-04-27T16:15:00	WARNING4

O restante dos padrões de acesso está resumido da seguinte forma:

Todos os padrões de acesso e a forma como o design do esquema os aborda estão resumidos na tabela abaixo:

Padrão de acesso	Tabela base/ GSI/LSI	Operação	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
createLogEntryForSpecificDevice	Tabela base	PutItem	DeviceID= deviceId	State#Date=state#date	
getLogsForSpecificDevice	Tabela base	Consulta	DeviceID= deviceId	State#Date begins_with "state1#"	ScanIndexForward = False
getWarningLogsForSpecificDevice	Tabela base	Consulta	DeviceID= deviceId	State#Date begins_with "WARNING"	
getLogsForOperatorBetweenTwoDates	GSI-1	Consulta	Operator= operatorName	Data entre date1 e date2	
getEscalatedLogsForSupervisor	GSI-2	Consulta	EscalatedTo=supervisorName		

Padrão de acesso	Tabela base/ GSI/LSI	Operação	Valor da chave de partição	Valores de chave de classificação	Outras condições/ filtros
getEscalatedLogsWithSpecificStatusForSupervisor	GSI-2	Consulta	EscalatedTo=supervisorName	State#Date begins_with "state1#"	
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Consulta	EscalatedTo=supervisorName	State#Date begins_with "state1#date1"	

Esquema final

Veja aqui os designs do esquema final. Para baixar esse design de esquema como arquivo JSON, consulte DynamoDB Examples no GitHub.

Tabela base

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

GSI-1

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

GSI-2

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator	Date	State
		d#11223	Sue	2020-04-27T16:15:00	WARNING4

Como usar o NoSQL Workbench com esse design de esquema

Você pode importar esse esquema final para o NoSQL Workbench, uma ferramenta visual que fornece atributos de modelagem de dados, visualização de dados e desenvolvimento de consultas para o DynamoDB, se quiser explorar e editar ainda mais seu novo projeto. Para começar, siga estas etapas:

1. Baixe o NoSQL Workbench. Para obter mais informações, consulte the section called “Baixar”.
2. Baixe o arquivo do esquema JSON listado acima, que já está no formato do modelo NoSQL Workbench.
3. Importe o arquivo do esquema JSON para o NoSQL Workbench. Para obter mais informações, consulte the section called “Importar um modelo existente”.
4. Depois de importar para o NOSQL Workbench, você pode editar o modelo de dados. Para obter mais informações, consulte the section called “Editar um modelo existente”.
5. Para visualizar o modelo de dados, adicionar dados de exemplo ou importar dados de exemplo de um arquivo CSV, use o atributo Visualizador de dados do NoSQL Workbench.

Usar o DynamoDB como armazenamento de dados para uma loja virtual

Esse caso de uso trata do uso do DynamoDB como armazenamento de dados para uma loja on-line (ou loja eletrônica).

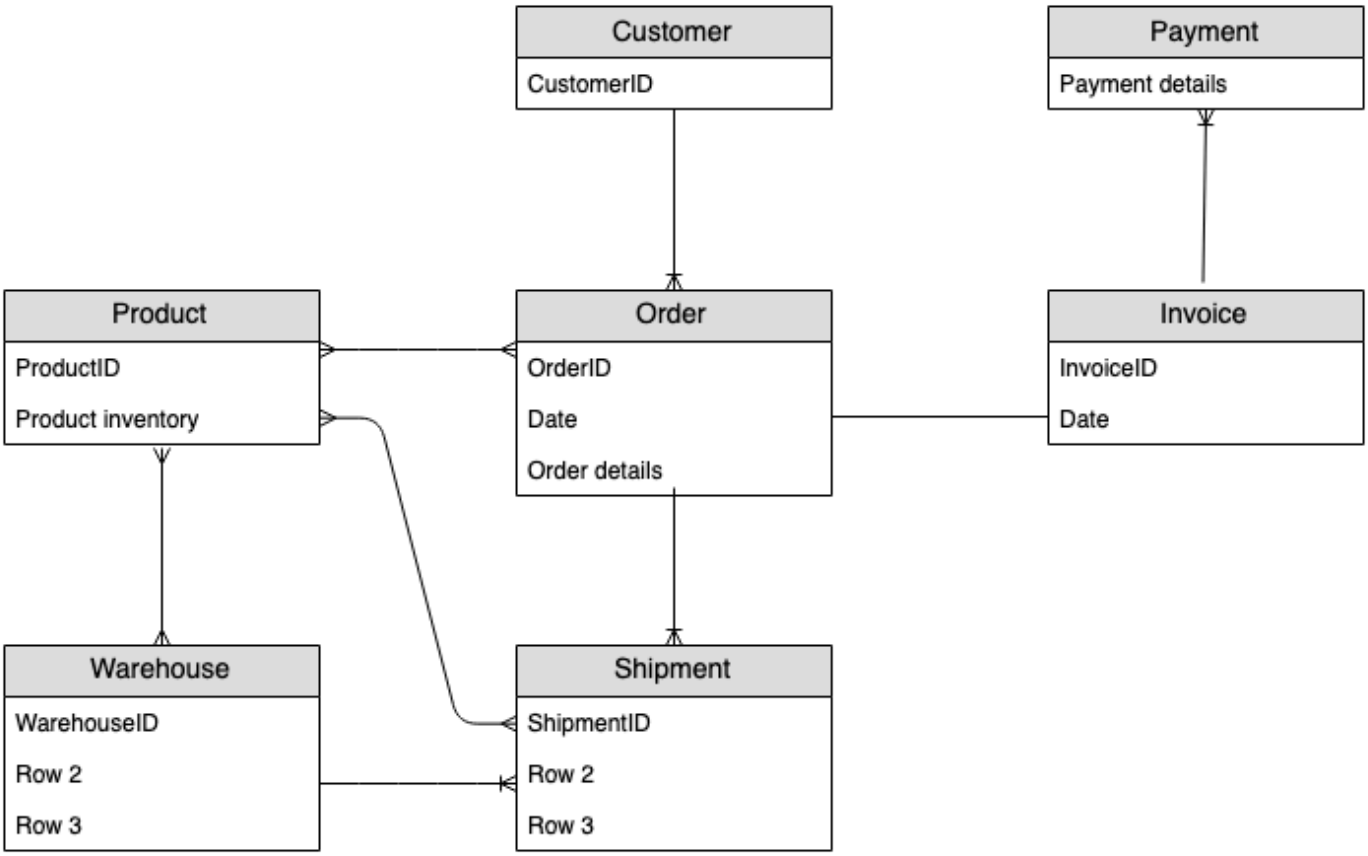
Caso de uso

Uma loja on-line permite que os usuários naveguem por diferentes produtos e, em algum momento, os comprem. Com base na fatura gerada, o cliente pode pagar usando um código de desconto ou vale-presente e depois pagar o valor restante com um cartão de crédito. Os produtos adquiridos serão retirados de um dos vários armazéns e enviados ao endereço fornecido. Os padrões de acesso típicos para uma loja on-line incluem:

- Obter cliente para determinado customerId
- Obter produto para determinado productId
- Obter armazém para determinado warehouseId
- Obter um inventário de produtos para todos os armazéns por meio de um productId
- Obter pedido para determinado orderId
- Obter todos os produtos de determinado orderId
- Obter fatura para determinado orderId
- Obter todos os envios para determinado orderId
- Obter todos os pedidos de determinado productId em determinado intervalo de datas
- Obter fatura para determinado invoiceId
- Obter todos os pagamentos para determinado invoiceId
- Obter detalhes de envio para determinado shipmentId
- Obter todos os envios para determinado warehouseId
- Obter o inventário de todos os produtos para determinado warehouseId
- Obter todas as faturas para determinado customerId em um intervalo de datas específico
- Obter todos os produtos encomendados por determinado customerId em um intervalo de datas específico

Diagrama de relacionamento de entidades

Este é o diagrama de relacionamento de entidades (ERD) que usaremos para modelar o DynamoDB como um armazenamento de dados de uma loja on-line.



Padrões de acesso

Este é o padrão de acesso que consideraremos ao usar o DynamoDB como um armazenamento de dados de uma loja on-line.

1. `getCustomerByCustomerId`
2. `getProductByProductId`
3. `getWarehouseByWarehouseId`
4. `getProductInventoryByProductId`
5. `getOrderDetailsByOrderId`
6. `getProductByOrderId`
7. `getInvoiceByOrderId`
8. `getShipmentByOrderId`
9. `getOrderByProductIdForDateRange`
10. `getInvoiceByInvoiceId`

11getPaymentByInvoiceId

12getShipmentDetailsByShipmentId

13getShipmentByWarehouseId

14getProductInventoryByWarehouseId

15getInvoiceByCustomerIdForDateRange

16getProductsByCustomerIdForDateRange

Evolução do design do esquema

Usando NoSQL Workbench para DynamoDB, importe AnOnlineShop_1.json para criar um modelo de dados chamado AnOnlineShop e uma tabela chamada OnlineShop. Observe que usamos os nomes genéricos PK e SK para a chave de partição e a chave de classificação. Essa é uma prática usada para armazenar diferentes tipos de entidade na mesma tabela.

Etapa 1: abordar o padrão de acesso 1 (**getCustomerByCustomerId**)

Importe AnOnlineShop_2.json para lidar com o padrão de acesso 1 (getCustomerByCustomerId). Algumas entidades não têm relacionamentos com outras, então usaremos o mesmo valor de PK e SK para elas. Nos dados de exemplo, observe que as chaves usam um prefixo c# a fim de distinguir o customerId de outras entidades que serão adicionadas posteriormente. Essa prática também é repetida para outras entidades.

Para abordar esse padrão de acesso, pode ser usada uma operação GetItem com PK=customerId e SK=customerId.

Etapa 2: abordar o padrão de acesso 2 (**getProductByProductId**)

Importe AnOnlineShop_3.json para abordar o padrão de acesso 2 (getProductByProductId) para a entidade product. As entidades do produto têm p# como prefixo e o mesmo atributo de chave de classificação foi usado para armazenar customerID, bem como productID. A nomenclatura genérica e o particionamento vertical nos permite criar essas coleções de itens para um design de tabela única eficaz.

Para abordar esse padrão de acesso, pode ser usada uma operação GetItem com PK=productId e SK=productId.

Etapa 3: abordar o padrão de acesso 3 (**getWarehouseByWarehouseId**)

Importe AnOnlineShop_4.json para abordar o padrão de acesso 3 (`getWarehouseByWarehouseId`) para a entidade `warehouse`. No momento, temos as entidades `customer`, `product` e `warehouse` adicionadas à mesma tabela. Elas são diferenciadas usando prefixos e o atributo `EntityType`. Um atributo de tipo (ou nomenclatura de prefixo) melhora a legibilidade do modelo. A legibilidade seria afetada se simplesmente armazenássemos IDs alfanuméricos para entidades diferentes no mesmo atributo. Seria difícil diferenciar uma entidade da outra na ausência desses identificadores.

Para abordar esse padrão de acesso, pode ser usada uma operação `GetItem` com `PK=warehouseId` e `SK=warehouseId`.

Tabela base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
p#12345	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Etapa 4: abordar o padrão de acesso 4 (**`getProductInventoryByProductId`**)

Importe AnOnlineShop_5.json para abordar o padrão de acesso 4 (`getProductInventoryByProductId`). A entidade `warehouseItem` é usada para acompanhar o número de produtos em cada armazém. Esse item normalmente é atualizado quando um produto é adicionado ou removido de um depósito. Conforme visto no ERD, há uma relação de muitos para muitos entre `product` e `warehouse`. Aqui, a relação de um para muitos de `product` para `warehouse` é modelada como `warehouseItem`. Posteriormente, a relação de um para muitos de `warehouse` para `product` também será modelada.

O padrão de acesso 4 pode ser abordado com uma consulta em `PK=ProductId` e `SK begins_with “w#“`.

Para obter mais informações sobre `begins_with()` e outras expressões que podem ser aplicadas para chaves de classificação, consulte Expressões de condição de chave.

Tabela base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
p#12345	w#12345	EntityType	Quantity	
		warehouseItem	50	
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Etapa 5: abordar os padrões de acesso 5 (**`getOrderDetailsByOrderId`**) e 6 (**`getProductByOrderId`**)

Adicione um mais alguns itens `customer`, `product` e `warehouse` à tabela importando AnOnlineShop_6.json. Depois, importe AnOnlineShop_7.json para criar uma coleção de itens para `order` que possa abordar os padrões de acesso 5 (`getOrderDetailsByOrderId`) e 6 (`getProductByOrderId`). Você pode ver a relação de um para muitos entre `order` e `product` modelada como entidades orderItem.

Para abordar o padrão de acesso 5 (`getOrderDetailsByOrderId`), consulte a tabela com `PK=orderId`. Isso fornecerá todas as informações sobre o pedido, incluindo `customerId` e produtos encomendados.

Tabela base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Para abordar o padrão de acesso 6 (`getProductByOrderId`), precisamos ler os produtos somente em um `order`. Consulte a tabela com `PK=orderId` e `SK begins_with "p#"` para fazer isso.

Tabela base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
o#12345	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Etapa 6: abordar o padrão de acesso 7 (**`getInvoiceByOrderId`**)

Importe AnOnlineShop_8.json para adicionar uma entidade `invoice` à coleção de itens de pedido e lidar com o padrão de acesso 7 (`getInvoiceByOrderId`). Para abordar esse padrão de acesso, é possível usar uma operação de consulta com `PK=orderId` e `SK begins_with "i#"`.

Tabela base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	i#55443	EntityType	Amount	Date
		invoice	400	2020-06-21T19:18:00
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Etapa 7: abordar o padrão de acesso 8 (**`getShipmentByOrderId`**)

Importe AnOnlineShop_9.json para adicionar entidades `shipment` à coleção de itens de pedido e abordar o padrão de acesso 8 (`getShipmentByOrderId`). Estamos estendendo o mesmo modelo particionado verticalmente por meio da adição de mais tipos de entidade no design de tabela única. Observe como a coleção de itens de pedido contém os diferentes relacionamentos que um entidade `order` tem com as entidades `shipment`, `orderItem` e `invoice`.

Para receber envios por `orderId`, você pode realizar uma operação de consulta com `PK=orderId` e `SK begins_with “sh#”`.

Tabela base:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	c#12345	EntityType	Date			
		order	2020-06-21T19:10:00			
	i#55443	EntityType	Amount	Date		
		invoice	400	2020-06-21T19:18:00		
	p#12345	EntityType	Price	Quantity		
		orderItem	100	2		
	p#99887	EntityType	Price	Quantity		
		orderItem	40	5		
	sh#88899	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00	w#12376
	sh#98765	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00	w#12345

Etapa 8: abordar o padrão de acesso 9 (**`getOrderByProductIdForDateRange`**)

Criamos uma coleção de itens de pedido na etapa anterior. Esse padrão de acesso tem novas dimensões de pesquisa (`ProductID` e `Date`), que exige que você escaneie toda a tabela e filtre os registros relevantes para buscar os itens específicos. Para abordar esse padrão de acesso, precisaremos criar um índice secundário global (GSI). Importe AnOnlineShop_10.json para criar uma coleção de itens usando o GSI que possibilita a recuperação de dados de `orderItem` de várias coleções de itens de pedido. Os dados agora têm `GSI1-PK` e `GSI1-SK`, que serão a chave de partição e a chave de classificação de `GSI1`, respectivamente.

O DynamoDB preenche automaticamente itens que contêm os principais atributos de um GSI da tabela para o GSI. Não há necessidade de fazer manualmente nenhuma inserção adicional no GSI.

Para abordar o padrão de acesso 9, execute uma consulta em `GSI1` com `GSI1-PK=productId` e `GSI1SK between (date1, date2)`.

Tabela base:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	p#12345	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#12345	2020-06-21T19:18:00	100	2
	p#99887	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#99887	2020-06-21T19:20:00	40	5

GSI1:

Primary key		Attributes				
Partition key: GSI1-PK	**Sort key: GSI1-SK**					
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#12345	orderItem	2	100
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#99887	orderItem	5	40

Etapa 9: abordar os padrões de acesso 10 (**`getInvoiceByInvoiceId`**) e 11 (**`getPaymentByInvoiceId`**)

Importe AnOnlineShop_11.json para abordar os padrões de acesso 10 (`getInvoiceByInvoiceId`) e 11 (`getPaymentByInvoiceId`), ambos relacionados a `invoice`. Embora esses sejam dois padrões de acesso diferentes, eles são realizados usando a mesma condição de chave. `Payments` são definidos como um atributo com o tipo de dados do mapa na entidade `invoice`.

> **Note**
>
> `GSI1-PK` e `GSI1-SK` estão sobrecarregados para armazenar informações sobre entidades diferentes para que vários padrões de acesso possam ser atendidos por meio do mesmo GSI. Para obter mais informações quanto à sobrecarga do GSI, consulte Sobrecarga de índices secundários globais no DynamoDB.

Para abordar os padrões de acesso 10 e 11, consulte `GSI1` com `GSI1-PK=invoiceId` e `GSI1-SK=invoiceId`.

GSI1:

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
i#55443	i#55443	o#12345	i#55443	invoice	c#12345	i#2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00

Etapa 10: abordar os padrões de acesso 12 (**`getShipmentDetailsByShipmentId`**) e 13 (**`getShipmentByWarehouseId`**)

Importe AnOnlineShop_12.json para abordar os padrões de acesso 12 (`getShipmentDetailsByShipmentId`) e 13 (`getShipmentByWarehouseId`).

Observe que as entidades `shipmentItem` são adicionadas à coleção de itens de pedido na tabela base para poder recuperar todos os detalhes sobre um pedido em uma única operação de consulta.

Tabela base:

Primary key		Attributes							
Partition key: PK	Sort key: SK								
o#12345	sh#88899	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#88899	sh#88899	w#12376	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
	sh#98765	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#98765	sh#98765	w#12345	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00
	shp#12345	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#99887	3				
	shp#54321	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#88899	p#99887	2				
	shp#55555	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#12345	2				

As chaves de partição e de classificação do `GSI1` já foram usadas para modelar uma relação de um para muitos entre `shipment` e `shipmentItem`. Para abordar o padrão de acesso 12 (`getShipmentDetailsByShipmentId`), consulte `GSI1` com `GSI1-PK=shipmentId` e `GSI1-SK=shipmentId`.

GSI1:

Primary key		Attributes							
Partition key: GSI1-PK	**Sort key: GSI1-SK**								
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
sh#88899	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
sh#98765	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

Precisaremos criar outro GSI (`GSI2`) para modelar a nova relação de um para muitos entre `warehouse` e `shipment` para o padrão de acesso 13 (`getShipmentByWarehouseId`). Para

abordar esse padrão de acesso, consulte `GSI2` com `GSI2-PK=warehouseId` e `GSI2-SK begins_with “sh#”`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12376	sh#88899	o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12345	sh#98765	o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00

Etapa 11: abordar os padrões de acesso 14 (**`getProductInventoryByWarehouseId`**), 15 (**`getInvoiceByCustomerIdForDateRange`**) e 16 (**`getProductsByCustomerIdForDateRange`**)

Importe AnOnlineShop_13.json para adicionar dados relacionados ao próximo conjunto de padrões de acesso. Para abordar o padrão de acesso 14 (`getProductInventoryByWarehouseId`), consulte `GSI2` com `GSI2-PK=warehouseId` e `GSI2-SK begins_with “p#”`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Para abordar o padrão de acesso 15 (`getInvoiceByCustomerIdForDateRange`), consulte `GSI2` com `GSI2-PK=customerId` e `GSI2-SK between (i#date1, i#date2)`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Para abordar o padrão de acesso 16 (`getProductsByCustomerIdForDateRange`), consulte `GSI2` com `GSI2-PK=customerId` e `GSI2-SK between (p#date1, p#date2)`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Note

No NoSQL Workbench, as facetas representam os diferentes padrões de acesso a dados de uma aplicação para o DynamoDB. As facetas oferecem uma maneira de visualizar um subconjunto dos dados em uma tabela, sem precisar ver os registros que não atendam às restrições da faceta. As facetas são consideradas uma ferramenta visual de modelagem de dados e não existem como uma estrutura utilizável no DynamoDB, pois são puramente um apoio para a modelagem de padrões de acesso.

Importe AnOnlineShop_facets.json para ver as facetas desse caso de uso.

Todos os padrões de acesso e a forma como o design do esquema os aborda estão resumidos na tabela abaixo:

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação
getCustomerByCustomerId	Tabela base	GetItem	PK=customerId	SK=customerId
getProductByProductId	Tabela base	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	Tabela base	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	Tabela base	Consulta	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	Tabela base	Consulta	PK=orderId	
getProductByOrderId	Tabela base	Consulta	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	Tabela base	Consulta	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	Tabela base	Consulta	PK=orderId	SK begins_with "sh#"
getOrderByProductIdForDateRange	GSI1	Consulta	PK=productId	SK entre date1 e date2
getInvoiceByInvoiceId	GSI1	Consulta	PK=invoiceId	SK=invoiceId

Padrão de acesso	Tabela base/ GSI/LSI	Operation	Valor da chave de partição	Valores de chave de classificação
getPaymentByInvoiceId	GSI1	Consulta	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Consulta	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Consulta	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Consulta	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Consulta	PK=customerId	SK entre i#date1 and i#date2
getProductsByCustomerIdForDateRange	GSI2	Consulta	PK=customerId	SK entre p#date1 and p#date2

Esquema final da loja on-line

Veja aqui os designs do esquema final. Para baixar esse design de esquema como arquivo JSON, consulte Padrões de design do DynamoDB no GitHub.

Tabela base

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
c#12345	c#12345	EntityType	Email	Name	
		customer	samaneh@example.com	Samaneh	
c#23456	c#23456	EntityType	Email	Name	
		customer	kathleen@example.com	Kathleen	
c#54321	c#54321	EntityType	Email	Name	
		customer	henrik@example.com	Henrik	
p#12345	p#12345	EntityType	Detail	Price	
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#12345	50
p#99887	p#99887	EntityType	Detail	Price	
		product	{"Name":{"S":"The Book"},"Description":{"S":"The best book ever"}}	40	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#99887	4
	w#12376	EntityType	Quantity		
		warehouseItem	4		
w#12345	w#12345	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}		
w#12376	w#12376	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":		

GSI1

Primary key		Attributes							
Partition key: GSI1-PK	**Sort key: GSI1-SK**								
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#12345	orderItem	c#12345	2020-06-21T19:18:00	100	2	
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#99887	orderItem	c#12345	2020-06-21T19:20:00	40	5	
i#55443	i#55443	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	c#12345	2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
sh#88899	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
sh#98765	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

GSI2

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
	sh#98765	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00
c#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	
w#12376	sh#88899	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00

Como usar o NoSQL Workbench com esse design de esquema

Você pode importar esse esquema final para o NoSQL Workbench, uma ferramenta visual que fornece atributos de modelagem de dados, visualização de dados e desenvolvimento de consultas para o DynamoDB, se quiser explorar e editar ainda mais seu novo projeto. Para começar, siga estas etapas:

1. Baixe o NoSQL Workbench. Para ter mais informações, consulte the section called “Baixar”.
2. Baixe o arquivo do esquema JSON listado acima, que já está no formato do modelo NoSQL Workbench.
3. Importe o arquivo do esquema JSON para o NoSQL Workbench. Para ter mais informações, consulte the section called “Importar um modelo existente”.
4. Depois de importar para o NOSQL Workbench, você pode editar o modelo de dados. Para ter mais informações, consulte the section called “Editar um modelo existente”.
5. Para visualizar o modelo de dados, adicionar dados de exemplo ou importar dados de exemplo de um arquivo CSV, use o atributo Visualizador de dados do NoSQL Workbench.

Práticas recomendadas para modelagem de dados relacionais no DynamoDB

Esta seção fornece práticas recomendadas para modelagem de dados relacionais no Amazon DynamoDB. Primeiro, apresentamos os conceitos tradicionais de modelagem de dados. Em seguida, descrevemos as vantagens de usar o DynamoDB em relação aos sistemas tradicionais de gerenciamento de banco de dados relacional, mostrando como ele elimina a necessidade de operações de JOIN e reduz as despesas operacionais indiretas.

Depois, explicamos como criar uma tabela do DynamoDB que escale com eficiência. Por fim, fornecemos um exemplo de como modelar dados relacionais no DynamoDB.

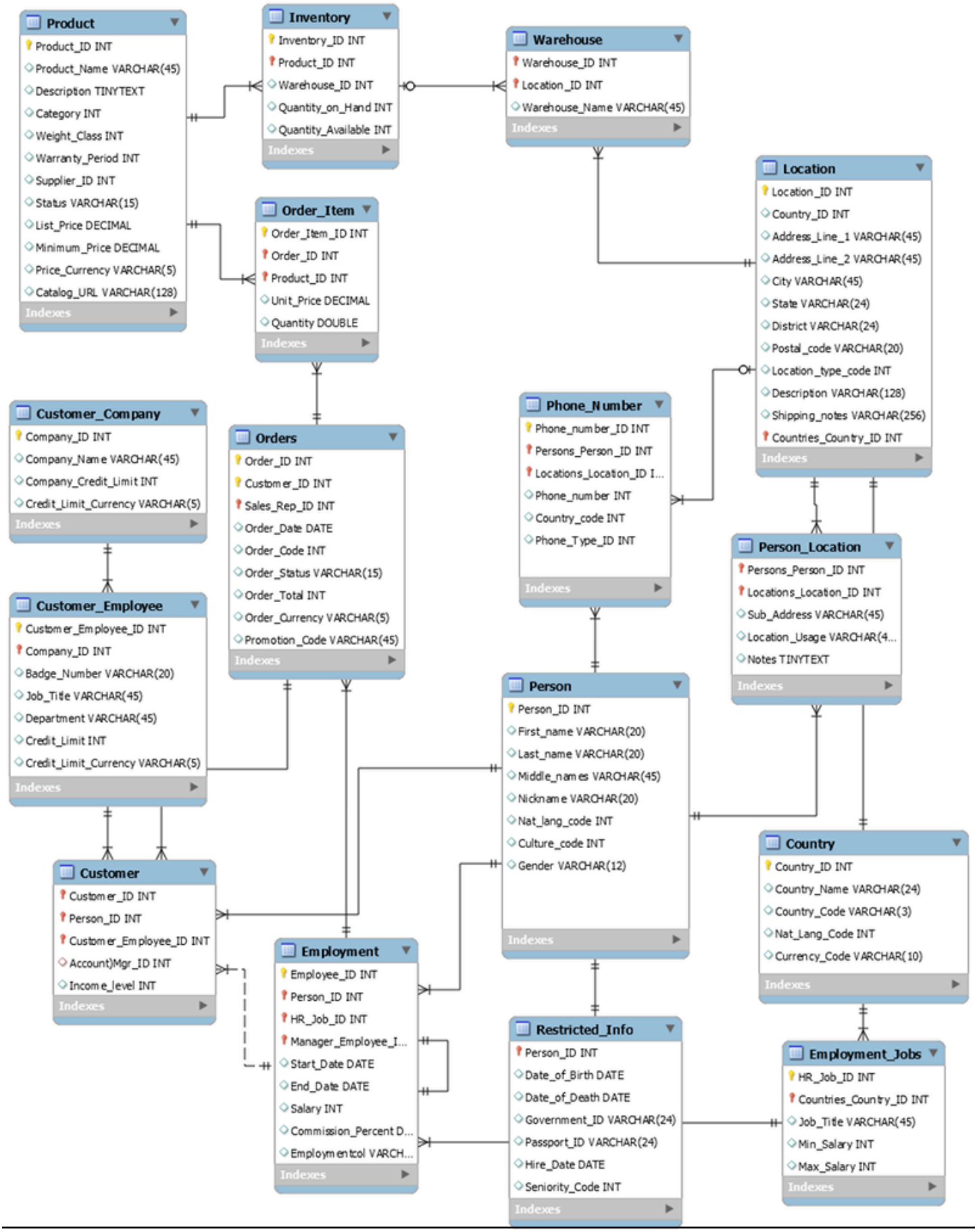
Tópicos

- Modelos tradicionais de banco de dados relacional
- Como o DynamoDB elimina a necessidade de operações JOIN
- Como as transações do DynamoDB eliminam a sobrecarga no processo de gravação
- Primeiras passos para a modelagem de dados relacionais no DynamoDB
- Exemplo de modelagem de dados relacionais no DynamoDB

Modelos tradicionais de banco de dados relacional

Os sistemas de gerenciamento de banco de dados relacional (RDBMS) tradicionais armazenam dados em uma estrutura relacional normalizada. O objetivo do modelo de dados relacional é reduzir a duplicação de dados (por meio da normalização) para apoiar a integridade referencial e reduzir as anomalias de dados.

O esquema a seguir é um exemplo de um modelo de dados relacional para uma aplicação genérica de entrada de pedidos. A aplicação atende a um esquema de recursos humanos que apoia os sistemas de suporte operacional e comercial de um fabricante fictício.


Product
Product_ID INT
Product_Name VARCHAR(45)
Description TINYTEXT
Category INT
Weight_Class INT
Warranty_Period INT
Supplier_ID INT
Status VARCHAR(15)
List_Price DECIMAL
Minimum_Price DECIMAL
Price_Currency VARCHAR(5)
Catalog_URL VARCHAR(128)
Indexes
Inventory
Inventory_ID INT
Product_ID INT
Warehouse_ID INT
Quantity_on_Hand INT
Quantity_Available INT
Indexes
Warehouse
Warehouse_ID INT
Location_ID INT
Warehouse_Name VARCHAR(45)
Indexes
Location
Location_ID INT
Country_ID INT
Address_Line_1 VARCHAR(45)
Address_Line_2 VARCHAR(45)
City VARCHAR(45)
State VARCHAR(24)
District VARCHAR(24)
Postal_code VARCHAR(20)
Location_type_code INT
Description VARCHAR(128)
Shipping_notes VARCHAR(256)
Countries_Country_ID INT
Indexes
Order_Item
Order_Item_ID INT
Order_ID INT
Product_ID INT
Unit_Price DECIMAL
Quantity DOUBLE
Indexes
Customer_Company
Company_ID INT
Company_Name VARCHAR(45)
Company_Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Orders
Order_ID INT
Customer_ID INT
Sales_Rep_ID INT
Order_Date DATE
Order_Code INT
Order_Status VARCHAR(15)
Order_Total INT
Order_Currency VARCHAR(5)
Promotion_Code VARCHAR(45)
Indexes
Phone_Number
Phone_number_ID INT
Persons_Person_ID INT
Locations_Location_ID I...
Phone_number INT
Country_code INT
Phone_Type_ID INT
Indexes
Person_Location
Persons_Person_ID INT
Locations_Location_ID INT
Sub_Address VARCHAR(45)
Location_Usage VARCHAR(4...
Notes TINYTEXT
Indexes
Customer_Employee
Customer_Employee_ID INT
Company_ID INT
Badge_Number VARCHAR(20)
Job_Title VARCHAR(45)
Department VARCHAR(45)
Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Person
Person_ID INT
First_name VARCHAR(20)
Last_name VARCHAR(20)
Middle_names VARCHAR(45)
Nickname VARCHAR(20)
Nat_lang_code INT
Culture_code INT
Gender VARCHAR(12)
Indexes
Country
Country_ID INT
Country_Name VARCHAR(24)
Country_Code VARCHAR(3)
Nat_Lang_Code INT
Currency_Code VARCHAR(10)
Indexes
Customer
Customer_ID INT
Person_ID INT
Customer_Employee_ID INT
Account)Mgr_ID INT
Income_level INT
Indexes
Employment
Employee_ID INT
Person_ID INT
HR_Job_ID INT
Manager_Employee_I...
Start_Date DATE
End_Date DATE
Salary INT
Commission_Percent D...
Employmentcol VARCH...
Indexes
Restricted_Info
Person_ID INT
Date_of_Birth DATE
Date_of_Death DATE
Government_ID VARCHAR(24)
Passport_ID VARCHAR(24)
Hire_Date DATE
Seniority_Code INT
Indexes
Employment_Jobs
HR_Job_ID INT
Countries_Country_ID INT
Job_Title VARCHAR(45)
Min_Salary INT
Max_Salary INT
Indexes

Como um serviço de banco de dados não relacional, o DynamoDB oferece muitas vantagens em relação aos sistemas tradicionais de gerenciamento de banco de dados relacional.

Como o DynamoDB elimina a necessidade de operações JOIN

Um RDBMS usa uma linguagem de consulta estruturada (SQL) para retornar dados à aplicação. Em decorrência da normalização do modelo de dados, essas consultas normalmente exigem o uso do operador `JOIN` para combinar dados de uma ou mais tabelas.

Por exemplo, para gerar uma lista de itens de ordens de compra classificadas pela quantidade em estoque em todos os depósitos que podem enviar cada item, você pode emitir a seguinte consulta de SQL no esquema anterior:

```
SELECT * FROM Orders
  INNER JOIN Order_Items ON Orders.Order_ID = Order_Items.Order_ID
  INNER JOIN Products ON Products.Product_ID = Order_Items.Product_ID
  INNER JOIN Inventories ON Products.Product_ID = Inventories.Product_ID
  ORDER BY Quantity_on_Hand DESC
```

Consultas SQL desse tipo podem fornecer uma API flexível para acesso aos dados, mas exigem um volume significativo de processamento. Cada junção na consulta aumenta a complexidade do tempo de execução da consulta, pois os dados de cada tabela devem ser armazenados e depois montados para retornar o conjunto de resultados.

Outros fatores que podem afetar o tempo de execução das consultas são o tamanho das tabelas e se as colunas que estão sendo unidas têm índices. A consulta anterior inicia consultas complexas em várias tabelas e, depois, classifica o conjunto de resultados.

Eliminar a necessidade de `JOINs` é o principal objetivo da modelagem de dados NoSQL. É por isso que criamos o DynamoDB para oferecer suporte à Amazon.com e que o DynamoDB pode oferecer performance consistente em qualquer escala. Dada a complexidade do tempo de execução das consultas SQL e `JOINs`, a performance do RBDMS não é constante em grande escala. Isso causa problemas de performance à medida que as aplicações do cliente se ampliam.

Embora a normalização dos dados reduza a quantidade de dados armazenados em disco, geralmente os recursos mais restritos que afetam a performance são o tempo de CPU e a latência da rede.

O DynamoDB foi criado para minimizar as duas restrições, eliminando `JOINs` (e incentivando a desnormalização dos dados) e otimizando a arquitetura do banco de dados para responder

totalmente a uma consulta de aplicação com uma única solicitação para um item. Essas qualidades permitem que o DynamoDB forneça performance de milissegundos de um dígito em qualquer escala. Isso ocorre porque a complexidade do tempo de execução das operações do DynamoDB é constante, independentemente do tamanho dos dados, para padrões comuns de acesso.

Como as transações do DynamoDB eliminam a sobrecarga no processo de gravação

Outro fator que pode desacelerar um RDBMS é o uso de transações para gravação em um esquema normalizado. Conforme mostrado no exemplo, as estruturas de dados relacionais usadas pela maioria das aplicações de processamento de transações on-line (OLTP) devem ser divididas e distribuídas entre várias tabelas lógicas quando são armazenadas em um RDBMS.

Portanto, um framework de transações compatível com ACID é necessário para evitar problemas de integridade de dados e condições de disputa que poderão ocorrer se uma aplicação tentar ler um objeto que esteja em processo de gravação. Esse framework de transações, quanto acoplado a um esquema relacional, pode adicionar uma sobrecarga significativa ao processo de gravação.

A implementação de transações no DynamoDB impede problemas comuns de escalabilidade encontrados em um RDBMS. O DynamoDB faz isso ao emitir uma transação como uma única chamada de API e limitar o número de itens que podem ser acessados nessa única transação. Transações de longa duração podem causar problemas operacionais ao manter os dados bloqueados por muito tempo ou perpetuamente, pois a transação nunca é encerrada.

Para evitar esses problemas no DynamoDB, as transações foram implementadas com duas operações de API distintas: `TransactWriteItems` e `TransactGetItems`. Essas operações de API não têm semânticas de início e fim, que são comuns em um RDBMS. Além disso, o DynamoDB tem um limite de acesso de 100 itens em uma transação para igualmente evitar transações de longa duração. Para saber mais sobre transações do DynamoDB, consulte Trabalhar com transações.

Por esses motivos, quando sua empresa precisar de resposta de baixa latência a consultas de alto tráfego, aproveitar as vantagens de um sistema NoSQL normalmente faz sentido do ponto de vista técnico e econômico. O Amazon DynamoDB ajuda a resolver os problemas que limitam a escalabilidade do sistema relacional simplesmente evitando-os.

Normalmente, a performance de um RDBMS não escala bem pelos seguintes motivos:

- Ele usa junções caras para remontar as visualizações exigidas dos resultados de consulta.

- Ele normaliza os dados e armazena-os em várias tabelas que exigem várias consultas para gravar em disco.
- Isso geralmente implica em custos de desempenho de um sistema de transações compatível com ACID.

O DynamoDB tem uma boa escalabilidade por estes motivos:

- A flexibilidade do esquema permite o DynamoDB armazene dados hierárquicos e complexos em um único item.
- O design da chave composta permite o armazenamento de itens relacionados na mesma tabela.
- As transações são realizadas em uma única operação. O limite ao número de itens que podem ser acessados é 100, para evitar operações de longa duração.

As consultas no armazenamento de dados tornam-se muito mais simples, normalmente da seguinte forma:

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

O DynamoDB executa muito menos trabalho para retornar os dados solicitados em comparação ao RDBMS no exemplo anterior.

Primeiras passos para a modelagem de dados relacionais no DynamoDB

> **Important**
>
> O design do NoSQL exige uma visão diferente daquela no design do RDBMS. Para um RDBMS, você pode criar um modelo de dados normalizado sem pensar nos padrões de acesso. Você poderá estendê-lo posteriormente quando surgirem novas perguntas e requisitos de consulta. Por outro lado, no Amazon DynamoDB, você não deve iniciar o design do seu esquema até saber quais perguntas ele precisará responder. Compreender os problemas de negócios e os casos de uso de aplicativo antecipadamente é absolutamente essencial.

Para iniciar o design de uma tabela do DynamoDB que poderá ser escalada com eficiência, é necessário realizar várias etapas primeiro para identificar os padrões de acesso exigidos pelos sistemas de suporte operacional e administrativo (OSS/BSS) que o design precisa comportar:

- Para novos aplicativos, analise as histórias dos usuários referentes a atividades e objetivos. Documente os vários casos de uso identificados e analise os padrões de acesso que eles exigem.
- Para aplicativos existentes, analise os logs de consulta para saber como as pessoas estão usando o sistema atualmente e quais são os principais padrões de acesso.

Após concluir esse processo, você deve encerrar com uma lista que pode ser semelhante à seguinte:

	Most Common/Import Access Patterns in Our Organization
1	Look up employee details by employee ID
2	Query employee details by employee name
3	Find an employee's phone number(s)
4	Find a customer's phone number(s)
5	Get orders for a given customer within a given date range
6	Show all open orders within a given date range across all customers
7	See all employees hired recently
8	Find all employees working in a given warehouse
9	Get all items on order for a given product
10	Get current inventories for a given product at all warehouses
11	Get customers by account representative
12	Get orders by account representative and date
13	Get all employees with a given job title
14	Get inventory by product and warehouse
15	Get total product inventory
16	Get account representatives ranked by order total and sales period

Em um aplicativo real, sua lista pode ser muito mais longa. Mas essa coleção representa a faixa de complexidade dos padrões de consulta que você pode encontrar em um ambiente de produção.

Uma abordagem comum ao design de esquemas do DynamoDB é identificar entidades da camada de aplicação e usar a desnormalização e a agregação de chaves compostas para reduzir a complexidade da consulta.

No DynamoDB, isso significa usar as chaves de classificação compostas, os índices secundários globais sobrecarregados, as tabelas/índices particionados e outros padrões de design. Você pode usar esses elementos para estruturar os dados, para que um aplicativo possa recuperar o que for necessário para um determinado padrão de acesso, usando uma única consulta em uma tabela ou um índice. O padrão principal que você pode usar para modelar o esquema normalizado, mostrado em Modelagem relacional, é o padrão da lista de adjacências. Outros padrões usados nesse design podem incluir a fragmentação de gravação do índice secundário global, a sobrecarga do índice secundário global, as chaves compostas e as agregações materializadas.

> **Important**
>
> Em geral, você deve manter o mínimo de tabelas possível em uma aplicação do DynamoDB. As exceções incluem os casos que envolvem dados de séries temporais de alto volume

ou conjuntos de dados que têm padrões muito diferentes de acesso. Uma única tabela com índices invertidos pode normalmente habilitar consultas simples para criar e recuperar estruturas de dados hierárquicas e complexas, exigidas pelo aplicativo.

Para usar o NoSQL Workbench para DynamoDB a fim de ajudar a visualizar o design da chave de partição, consulte Criar modelos de dados com o NoSQL Workbench.

Exemplo de modelagem de dados relacionais no DynamoDB

Esse exemplo descreve como modelar dados relacionais no Amazon DynamoDB. Um design de tabela do DynamoDB corresponde ao esquema relacional de entrada de pedidos que é mostrado em Modelagem relacional. Ele acompanha a Padrão de design da lista de adjacências, que é um modo comum de representar as estruturas de dados relacionais no DynamoDB.

O padrão de design requer que você defina um conjunto de tipos de entidades que se correlacionam geralmente com várias tabelas no esquema relacional. Os itens da entidade são adicionados à tabela usando uma chave primária composta (partição e classificação). A chave de partição desses itens de entidade é o atributo que identifica exclusivamente o item e que é mencionado genericamente em todos os itens como PK. O atributo de chave de classificação contém um valor do atributo que pode ser usado para um índice invertido ou um índice secundário global. Ele é referido genericamente como SK.

Você define as seguintes entidades que são compatíveis com o esquema relacional de entrada de pedidos.

1. HR-Employee - PK: EmployeeID, SK: Employee Name
2. HR-Region - PK: RegionID, SK: Region Name
3. HR-Country - PK: CountryId, SK: Country Name
4. HR-Location - PK: LocationID, SK: Country Name
5. HR-Job - PK: JobID, SK: Job Title
6. HR-Department - PK: DepartmentID, SK: DepartmentName
7. OE-Customer - PK: CustomerID, SK: AccountRepID
8. OE-Order - PK OrderID, SK: CustomerID
9. OE-Product - PK: ProductID, SK: Product Name
10. OE-Warehouse - PK: WarehouseID, SK: Region Name

Após ter adicionado esses itens de entidade à tabela, você pode definir as relações entre eles adicionando itens de borda às partições de item de entidade. A tabela a seguir demonstra essa etapa.

Nesse exemplo, as partições `Employee`, `Order` e `Product Entity` na tabela têm itens de borda adicionais que contêm indicadores para outros itens de entidade na tabela. Em seguida, defina alguns índices secundários globais (GSIs – Global secondary indexes) para compatibilidade com todos os padrões de acesso definidos anteriormente. Os itens de entidade não usam o mesmo tipo de valor para a chave primária nem o atributo de chave de classificação. Tudo isso é necessário para que os atributos de chave primária e de chave de classificação presentes sejam inseridos na tabela.

O fato de algumas dessas entidades usarem nomes próprios e outras usarem outros IDs de entidade como os valores da chave de classificação permite que o mesmo índice secundário global seja compatível com vários tipos de consultas. Essa técnica é chamada de sobrecarga de GSI. Ela elimina efetivamente o limite padrão de 20 índices secundários globais para as tabelas que contêm vários tipos de itens. Isso é mostrado no diagrama a seguir como GSI 1.

O GSI 2 foi projetado para compatibilidade com um padrão bastante comum de acesso a aplicativos, que é obter todos os itens na tabela que têm um determinado estado. Para uma tabela grande com uma distribuição desigual de itens entre os estados disponíveis, esse padrão de acesso pode resultar em uma chave dinâmica, a menos que os itens sejam distribuídos em mais de uma partição lógica que pode ser consultada simultaneamente. Esse padrão de design é chamado `write sharding`.

Para realizar isso para o GSI 2, o aplicativo adiciona o atributo de chave primária do GSI 2 a cada item de pedidos. Ele preenche o campo com um número aleatório em um intervalo de 0 – N, em que N pode ser genericamente calculado usando a fórmula a seguir, a menos que haja um motivo específico para se fazer de outra maneira.

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

Por exemplo, suponha que você espere o seguinte:

- Até 2 milhões de pedidos estarão no sistema, aumentando para 3 milhões em 5 anos.
- Até 20% desses pedidos estarão em um estado ABERTO por um determinar tempo.

- O registro médio de pedidos é cerca de 100 bytes, com três registros de `OrderItem` na partição de pedidos com cerca de 50 bytes cada, oferecendo um tamanho médio de entidade de pedidos de 250 bytes.

Para a tabela, o cálculo do fator N seria semelhante ao seguinte.

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K

N = (0.2 * 3M) / 48K = 13
```

Nesse caso, você precisa distribuir todos os pedidos em pelo menos 13 partições lógicas em GSI 2 para garantir que uma leitura de todos os itens de `Order` com um status `OPEN` não cause uma partição dinâmica na camada de armazenamento físico. É uma boa prática preencher esse número para permitir anomalias no conjunto de dados. Então, um modelo que use `N = 15` é provavelmente bom. Como mencionado anteriormente, você faz isso adicionando o valor 0 – N aleatório ao atributo PK GSI 2 de cada registro `Order` e`OrderItem` que é inserido na tabela.

Esse detalhamento supõe que o padrão de acesso que requer que a coleta de todas as faturas `OPEN` ocorra relativamente sem frequência, de modo que você possa usar a capacidade de intermitência para preencher a solicitação. Você pode consultar o seguinte índice secundário global usando uma condição de chave de classificação de `State` e de `Date Range` para produzir um subconjunto ou todos os `Orders` em um determinado estado, conforme necessário.

Neste exemplo, os itens são distribuídos de modo aleatório entre 15 partições lógicas. Essa estrutura funciona, porque o padrão acesso exige a recuperação de um grande número de itens. Portanto, é improvável que alguns dos 15 threads retornarão conjuntos vazios de resultados que poderiam potencialmente representar a capacidade desperdiçada. Uma consulta sempre usa 1 unidade de capacidade de leitura (RCU) ou 1 unidade de capacidade de gravação (WCU), mesmo se nada for retornado ou nenhum dado for gravado.

Se o padrão acesso exigir uma consulta de alta velocidade nesse índice secundário global que retorna um conjunto de resultados esparsos, é provavelmente melhor usar um algoritmo hash para distribuir os itens em vez de um padrão aleatório. Nesse caso, você pode selecionar um atributo que seja conhecido quando a consulta for executada em tempo de execução e aplicar hash a esse atributo em um espaço de chaves de 0 – 14 quando os itens são inseridos. Então, eles podem ser lidos de modo eficiente no índice secundário global.

Por fim, você pode reanalisar os padrões de acesso que foram definidos anteriormente. A seguir está a lista de padrões de acesso e as condições de consulta que serão usadas com a nova versão do DynamoDB da aplicação para acomodá-los.

S. Não.	Padrões de acesso	Condições de consulta
1	Procurar os detalhes do funcionário por ID do funcionário	Chave primária na tabela, ID=“HR-EMPLOYEE”
2	Consultar detalhes do funcionário por nome do funcionário	Use GSI-1, PK=“Nome do funcionário”
3	Obter apenas os detalhes do trabalho atual de um funcionário	Chave primária na tabela, PK=HR-EMPLOYEE-1, SK começa com “JH”
4	Obter pedidos de um cliente em um intervalo de datas	Use GSI-1, PK=CUSTOMER1, SK=“STATUS-DATE”, para cada StatusCode
5	Mostrar todos os pedidos no status OPEN para um intervalo de datas de todos os clientes	Use GSI-2, PK=query em paralelo para o intervalo [0..N], SK entre Open-Date1 e Open-Date2
6	Todos os funcionários contratados recentemente	Use GSI-1, PK=“HR-CONFIDENTIAL”, SK > date1
7	Encontrar todos os funcionários em um armazém específico	Usar GSI-1, PK=WAREHOUSE1
8	Obter todos os Orderitems de um produto, incluindo inventários do local do depósito	Use GSI-1, PK=PRODUCT1

S. Não.	Padrões de acesso	Condições de consulta
9	Obter clientes por represent ante da conta	Use GSI-1, PK=ACCOUNT-REP
10	Obter pedidos por represent ante de conta e data	Use GSI-1, PK=ACCOUNT-REP, SK="STATUS-DATE", para cada StatusCode
11	Obter todos os funcionários com um cargo específico	Use GSI-1, PK=JOBTITLE
12	Obter inventário por produto e armazém	Chave primária na tabela, PK=OE-PRODUCT1,SK=PRODUCT1
13	Obter o inventário total de produtos	Chave primária na tabela, PK=OE-PRODUCT1,SK=PRODUCT1
14	Obter representantes de conta classificados por total do pedido e período de vendas	Use GSI-1, PK=YYYY-Q1, scanIndexForward=False

Migrar um banco de dados relacional para o DynamoDB

A migração de um banco de dados relacional para o DynamoDB exige um planejamento cuidadoso para garantir um resultado bem-sucedido. Este guia ajudará você a entender como esse processo funciona, quais ferramentas estão disponíveis e, depois, como avaliar possíveis estratégias de migração e selecionar uma que atenda às suas necessidades.

Tópicos

- Motivos para migrar para o DynamoDB
- Considerações ao migrar um banco de dados relacional para o DynamoDB
- Noções básicas de como funciona uma migração para o DynamoDB
- Ferramentas úteis na migração para o DynamoDB
- Selecionar a estratégia apropriada para migrar para o DynamoDB
- Realizar uma migração off-line para o DynamoDB
- Realizar uma migração híbrida para o DynamoDB
- Realizar uma migração on-line para o DynamoDB migrando cada tabela 1:1
- Realizar uma migração on-line para o DynamoDB usando tabela de preparação personalizada.

Motivos para migrar para o DynamoDB

A migração para o Amazon DynamoDB apresenta uma série de benefícios interessantes para empresas e organizações. Veja a seguir algumas das principais vantagens que tornam o DynamoDB uma opção atraente para migração de banco de dados:

- Escalabilidade: o DynamoDB foi concebido para lidar com grandes workloads e escalar perfeitamente para acomodar volumes e tráfego de dados crescentes. Com o DynamoDB, é fácil aumentar ou reduzir a escala do banco de dados verticalmente com base na demanda, garantindo que as aplicações possam lidar com picos repentinos de tráfego sem comprometer a performance.
- Performance: o DynamoDB oferece acesso a dados de baixa latência, permitindo que as aplicações recuperem e processem dados com velocidade excepcional. Sua arquitetura distribuída garante que as operações de leitura e gravação sejam distribuídas em vários nós, oferecendo tempos de resposta consistentes de um dígito de milissegundo, mesmo com altas taxas de solicitação.

- Totalmente gerenciado: o DynamoDB é um serviço totalmente gerenciado oferecido pela AWS. Isso significa que a AWS lida com os aspectos operacionais do gerenciamento de banco de dados, incluindo provisionamento, configuração, aplicação de patches, backups e escalabilidade. Isso permite que você se concentre mais no desenvolvimento de aplicações e menos nas tarefas de administração do banco de dados.
- Arquitetura sem servidor: o DynamoDB comporta um modelo sem servidor, conhecido como DynamoDB sob demanda, em que você paga somente pelas solicitações reais de leitura e gravação feitas pela aplicação, sem a necessidade de provisionamento antecipado de capacidade. Esse modelo de pagamento por solicitação oferece economia e despesas operacionais mínimas, pois você paga apenas pelos recursos que consome, sem a necessidade de provisionar e monitorar a capacidade.
- Flexibilidade do NoSQL: ao contrário dos bancos de dados relacionais tradicionais, o DynamoDB segue um modelo de dados NoSQL, oferecendo flexibilidade no design do esquema. Com o DynamoDB, é possível armazenar dados estruturados, semiestruturados e não estruturados, o que o torna adequado para lidar com tipos de dados diversos e dinâmicos. Essa flexibilidade permite ciclos de desenvolvimento mais rápidos e uma adaptação mais fácil às mudanças nos requisitos de negócios.
- Alta disponibilidade e durabilidade: o DynamoDB replica dados em várias zonas de disponibilidade em uma região, garantindo alta disponibilidade e durabilidade dos dados. Ele gerencia automaticamente a replicação, o failover e a recuperação, minimizando o risco de perda de dados ou interrupções no serviço. O DynamoDB oferece um SLA de disponibilidade de até 99,999%.
- Segurança e conformidade: o DynamoDB integra-se ao AWS Identity and Access Management para oferecer um controle de acesso minucioso. Ele fornece criptografia em repouso e em trânsito, garantindo a segurança dos dados. O DynamoDB também segue vários padrões de conformidade, incluindo HIPAA, PCI DSS e RGPD, possibilitando que você atenda aos requisitos regulatórios.
- Integração com o ecossistema da AWS: como parte do ecossistema da AWS, o DynamoDB integra-se perfeitamente com outros serviços da AWS, como AWS Lambda, AWS CloudFormation e AWS AppSync. Essa integração permite criar arquiteturas sem servidor, aproveitar a infraestrutura como código e criar aplicações orientadas por dados em tempo real.

Considerações ao migrar um banco de dados relacional para o DynamoDB

Os sistemas de bancos de dados relacionais e os bancos de dados NoSQL têm diferentes pontos fortes e fracos. Essas diferenças tornam o design de banco de dados diferente entre os dois sistemas.

Tipo de tarefa	Banco de dados relacional	Banco de dados NoSQL
Consultar o banco de dados	Em bancos de dados relacionais, as consultas de dados são flexíveis, mas têm um custo relativamente alto e não escalam com facilidade em situações de grande volume de tráfego (consulte Primeiras passos para a modelagem de dados relacionais no DynamoDB). Uma aplicação de banco de dados relacional pode implementar lógica de negócios em procedimentos armazenados, subconsultas SQL, consultas de atualização em massa e consultas de agregação.	Em um banco de dados NoSQL como o DynamoDB, há formas limitadas de consultar dados com eficiência. As demais formas de consulta podem apresentar alto custo e baixa performance. As gravações no DynamoDB são singletons. A lógica de negócios da aplicação que antes era executada em procedimentos armazenados deve ser refatorada para ser executada fora do DynamoDB em código personalizado executado em um host, como Amazon EC2 ou AWS Lambda.
Projetar o banco de dados	Você deve criar o design para ter flexibilidade sem se preocupar com detalhes de implementação ou performance. A otimização de consultas geralmente não afeta o design do esquema, mas a normalização é importante.	Você deve criar o esquema especificamente para fazer as consultas mais comuns e importantes do modo mais rápido e econômico possível. Suas estruturas de dados são adaptadas aos requisitos

Tipo de tarefa	Banco de dados relacional	Banco de dados NoSQL
		específicos de seus casos de uso de negócios.

A criação de um projeto para banco de dados NoSQL requer uma mentalidade diferente da criação de um projeto para um sistema de gerenciamento de banco de dados relacional (RDBMS). Para um RDBMS, você pode criar um modelo de dados normalizado sem pensar nos padrões de acesso. Você poderá estendê-lo posteriormente quando surgirem novas perguntas e requisitos de consulta. Você pode organizar cada tipo de dados em sua própria tabela.

Com o projeto NoSQL, você pode projetar o esquema do DynamoDB quando não souber a quais perguntas ele precisará responder. É essencial compreender os problemas de negócios e os padrões de leitura e gravação da aplicação. Também é necessário manter o mínimo de tabelas possível em uma aplicação do DynamoDB. Com menos tabelas, há mais escalabilidade, menos gerenciamento de permissões e menor sobrecarga para o DynamoDB. Isso também pode ajudar a manter os custos de backup mais baixos em geral.

A tarefa de modelar dados relacionais para o DynamoDB e criar uma versão da aplicação front-end é um tópico separado. Este guia pressupõe que você tenha uma nova versão da aplicação criada para usar o DynamoDB, mas mesmo assim é necessário determinar a melhor forma de migrar e sincronizar dados históricos durante a substituição.

Considerações sobre tamanho

O tamanho máximo de cada item (linha) armazenado em uma tabela do DynamoDB é 400 KB. Para ter mais informações, consulte the section called “Cotas”. O tamanho do item é determinado pelo tamanho total de todos os nomes e valores de atributos em um item. Para ter mais informações, consulte the section called “Tamanhos e formatos de item”.

Se a aplicação precisa armazenar mais dados em um item do que o limite de tamanho do DynamoDB permite, divida o item em um conjunto de itens, compacte os dados do item ou armazene-o como um objeto no Amazon Simple Storage Service (Amazon S3) ao armazenar o identificador do objeto do Amazon S3 no item do DynamoDB. Consulte the section called “Itens grandes”. O custo de atualização de um item baseia-se no tamanho total do item. Para workloads que exigem atualizações frequentes de itens existentes, ter itens pequenos de 1 ou 2 KB custará menos para atualizar do que itens maiores. Consulte the section called “Trabalhar com coleções de itens” para ter mais informações sobre conjuntos de itens.

Ao selecionar a partição e os atributos da chave de classificação, outras configurações da tabela, o tamanho e a estrutura do item e se deseja criar índices secundários, não deixe de analisar a documentação de modelagem do DynamoDB, bem como the section called “Otimização de custo” no guia. Teste seu plano de migração para que a solução do DynamoDB seja econômica e ajuste-se aos recursos e às limitações do DynamoDB.

Noções básicas de como funciona uma migração para o DynamoDB

Antes de analisar as ferramentas de migração disponíveis para nós, pense em como as gravações são processadas pelo DynamoDB.

A operação de gravação padrão e mais comum é uma única operação de API `PutItem`. É possível realizar uma operação `PutItem` em um loop para processar conjuntos de dados. O DynamoDB comporta conexões simultâneas praticamente ilimitadas; portanto, supondo que você possa configurar e executar uma rotina de carregamento extremamente multiencadeada, como MapReduce ou Spark, a velocidade das gravações é limitada apenas pela capacidade da tabela de destino (que geralmente também é ilimitada).

Ao carregar dados no DynamoDB, é importante entender a velocidade de gravação do carregador. Se os itens (linhas) que você está carregando tiverem 1 KB de tamanho ou menos, essa velocidade será simplesmente o número de itens por segundo. A tabela de destino pode então ser provisionada com WCU (unidades de capacidade de gravação) suficientes para lidar com essa taxa. Se seu carregador exceder a capacidade provisionada em qualquer segundo que seja, as solicitações adicionais poderão ter controle de utilização ou ser totalmente rejeitadas. É possível conferir se há controles de utilização nos gráficos do CloudWatch encontrados na guia de monitoramento do console do DynamoDB.

A segunda operação que pode ser executada é com uma API relacionada, chamada `BatchWriteItem`. `BatchWriteItem` permite combinar até 25 solicitações de gravação em uma chamada de API. Elas são recebidas pelo serviço e processados como solicitações `PutItem` separadas para a tabela. No momento, ao escolher `BatchWriteItem`, você não terá a vantagem das novas tentativas automáticas incluídas no SDK da AWS ao fazer chamadas únicas com `PutItem`. Portanto, se houver algum erro (como exceções de controle de utilização), será necessário procurar a lista de todas as gravações com falha na chamada de resposta para `BatchWriteItem`. Para ter mais informações sobre como lidar com avisos de controle de utilização,

caso sejam detectados nos gráficos de controle de utilização do CloudWatch, consulte the section called “Controle de utilização”.

O terceiro tipo de importação de dados é possível com o recurso de importação do DynamoDB do S3. Esse recurso permite que você prepare um grande conjunto de dados no Amazon S3 e peça ao DynamoDB que importe automaticamente os dados para uma nova tabela. A importação não é instantânea e levará um tempo proporcional ao tamanho do conjunto de dados. No entanto, isso é conveniente, pois não exige nenhuma plataforma ETL nem o código personalizado do DynamoDB. O DynamoDB carrega os dados em uma nova tabela criada pela importação. No momento, não é possível carregar dados em uma tabela existente. O DynamoDB importa os dados no estado em que se encontram, sem transformações. Semelhante ao `PutItem`, ele requer um processo upstream e grava os dados no formato escolhido em um bucket do Amazon S3.

Ferramentas úteis na migração para o DynamoDB

Há várias ferramentas comuns de migração e ETL que você pode usar para migrar dados para o DynamoDB.

A Amazon oferece uma série de ferramentas de dados que podem ser utilizadas na migração, incluindo AWS Database Migration Service (DMS), AWS Glue, Amazon EMR e Amazon Managed Streaming for Apache Kafka. Todas essas ferramentas podem ser usadas para realizar uma migração em tempo de inatividade e elas podem utilizar os atributos de captura de dados de alteração (CDC) do banco de dados relacional para comportar migrações on-line. Ao escolher uma ferramenta, será útil pensar no conjunto de habilidades e na experiência que sua organização tem com cada ferramenta, além dos atributos, do desempenho e do custo de cada uma.

Muitos clientes optam por elaborar seus próprios scripts e trabalhos de migração para criar transformações de dados personalizadas para o processo de migração. Se você planeja operar uma tabela de alto volume do DynamoDB com tráfego intenso de gravação ou trabalhos grandes e regulares de carga em massa, talvez você mesmo queira gravar código de migração para conhecer melhor o comportamento do DynamoDB em tráfego intenso de gravação. Controle de utilização e provisionamento eficiente de tabelas são situações que podem ocorrer no início do projeto ao realizar uma migração prática.

Selecionar a estratégia apropriada para migrar para o DynamoDB

Uma grande aplicação de banco de dados relacional pode abranger uma centena ou mais de tabelas e comportar várias funções diferentes da aplicação. Ao abordar uma grande migração, pense em

dividir sua aplicação em componentes ou microsserviços menores e migrar um pequeno conjunto de tabelas por vez. Depois, é possível migrar componentes adicionais para o DynamoDB em ondas.

Ao selecionar uma estratégia de migração, vários fatores podem direcionar você para uma solução ou outra. Podemos apresentar essas opções em uma árvore de decisão a fim de simplificar as opções disponíveis para nós, de acordo com nossos requisitos e recursos disponíveis. Os conceitos são brevemente mencionados aqui (mas serão abordados com mais detalhes posteriormente no guia):

- Migração off-line: se a aplicação puder tolerar algum tempo de inatividade durante a migração, isso simplificará o processo.
- Migração híbrida: essa abordagem possibilita um tempo de atividade parcial durante a migração, como permitir leituras, mas não gravações, ou permitir leituras e inserções, mas não atualizações e exclusões.
- Migração on-line: aplicações que não exigem tempo de inatividade durante a migração são mais difíceis de migrar e podem exigir planejamento significativo e desenvolvimento personalizado. Uma decisão importante é calcular e ponderar os custos da criação de um processo de migração personalizado em comparação ao custo para a empresa de ter uma janela de tempo de inatividade durante a substituição.

If (Se)	E	Então
For possível desativar a aplicação por algum tempo durante uma janela de manutenção para realizar a migração de dados. Essa é uma migração off-line.		Use o AWS DMS e realize uma migração off-line usando uma tarefa de carga completa. Pré-modele os dados de origem com um SQL `VIEW`, se desejar.
For possível executar a aplicação no modo somente leitura durante a migração. Essa é uma migração híbrida.		Desabilite as gravações na aplicação ou no banco de dados de origem. Use o AWS DMS e realize uma migração off-line usando uma tarefa de carga completa.

If (Se)	E	Então
For possível executar a aplicação com leituras e inserções de novos registros, mas sem atualizações ou exclusões, durante a migração. Essa é uma migração híbrida.	Você tiver habilidades em desenvolvimento de aplicações e puder atualizar a aplicação relacional existente para realizar gravações duplas, inclusive no DynamoDB, para todos os novos registros.	Use o AWS DMS e realize uma migração off-line usando uma tarefa de carga completa. Simultaneamente, implante uma versão da aplicação existente que permita leituras e execute gravações duplas.
Você precisar de uma migração com o mínimo de tempo de inatividade. Essa é uma migração on-line.	• Você estiver migrando tabelas de origem de um para um para o DynamoDB sem grandes alterações no esquema.	Use o AWS DMS para realizar uma migração de dados on-line. Executar uma tarefa de carregamento em massa seguida pela tarefa de sincronização do CDC.
Você precisar de uma migração com o mínimo de tempo de inatividade. Essa é uma migração on-line.	• Você estiver combinando tabelas de origem em menos tabelas do DynamoDB seguindo o esquema de empilhamento ou a filosofia de tabela única. • Você tiver habilidades em desenvolvimento de banco de dados de backend e capacidade disponível no host SQL.	Crie a tabela pronta para NoSQL no banco de dados SQL. Preencha-a e sincronize-a usando JOINs, UNIONs, VIEWs, gatilhos e procedimentos armazenados.

If (Se)	E	Então
Você precisar de uma migração com o mínimo de tempo de inatividade. Essa é uma migração on-line.	• Você estiver combinando tabelas de origem em menos tabelas do DynamoDB seguindo a filosofia de tabela única. Por exemplo: • Você não tiver habilidades de desenvolvimento de banco de dados de backend e capacidade disponível no host SQL.	Pense nas abordagens de migração híbrida ou off-line.
Você precisar de uma migração com o mínimo de tempo de inatividade. Essa é uma migração on-line.	For possível ignorar a migração de dados históricos de transações ou arquivá-los no Amazon S3 em vez de migrá-los. Você só precisa migrar algumas pequenas tabelas estáticas.	Escreva um script ou use qualquer ferramenta ETL para migrar as tabelas. Pré-modele os dados de origem com um SQL `VIEW`, se desejar.

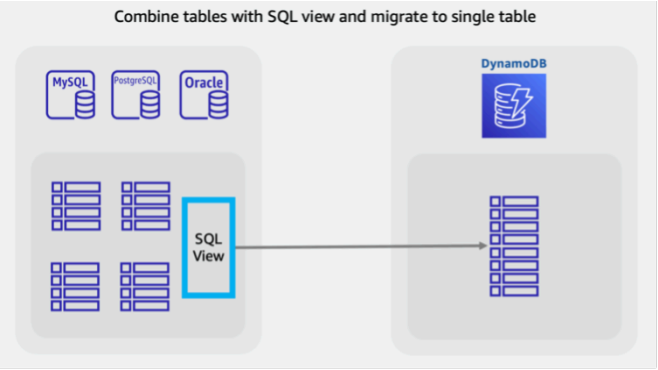
Realizar uma migração off-line para o DynamoDB

As migrações off-line são adequadas para quando é possível permitir uma janela de tempo de inatividade para realizar a migração. Os bancos de dados relacionais geralmente têm pelo menos algum tempo de inatividade a cada mês para manutenção e correção, ou podem levar mais tempo de inatividade para atualizações de hardware ou atualizações de versões principais.

O Amazon S3 pode ser usado como uma área de preparação durante uma migração. Os dados armazenados no formato CSV (valores separados por vírgula) ou no formato JSON do DynamoDB podem ser importados automaticamente para uma nova tabela do DynamoDB usando o recurso de importação do DynamoDB do S3.

Convém combinar tabelas para utilizar padrões exclusivos de acesso ao NoSQL (por exemplo, transformar quatro tabelas herdadas em uma única tabela do DynamoDB). Uma única solicitação de documento de chave-valor ou uma consulta para uma coleção de itens pré-agrupados geralmente

exibe uma melhor latência do que um banco de dados SQL que realize uma junção de várias tabelas. No entanto, isso dificulta a tarefa de migração. Uma visualização SQL pode fazer o trabalho no banco de dados de origem para preparar um único conjunto de dados representando todas as quatro tabelas em um conjunto.



Essa visualização pode `JOIN` tabelas em um formato desnormalizado ou manter as entidades normalizadas e empilhar tabelas usando um SQL `UNION`. As principais decisões sobre a reformulação de dados relacionais são abordadas neste vídeo. Para migrações off-line, usar uma visualização para combinar tabelas é uma ótima maneira de moldar dados para um esquema de tabela única do DynamoDB.

Planejamento

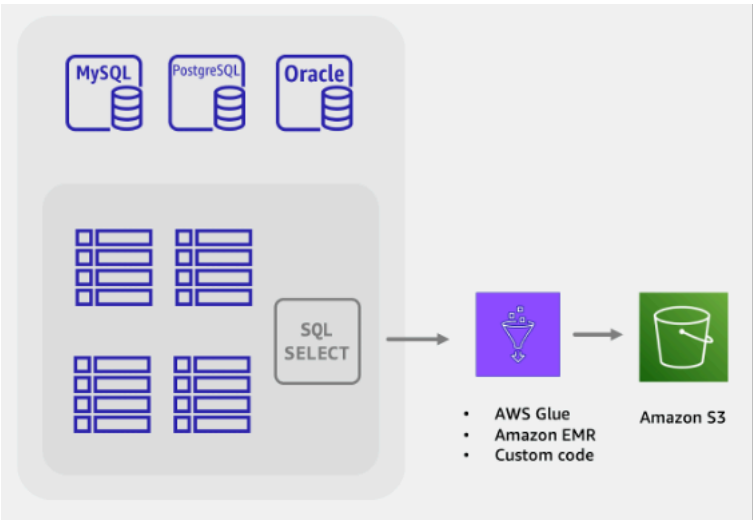
Executar uma migração off-line usando o Amazon S3

Ferramentas

- Um trabalho de ETL para extrair e transformar dados SQL e armazená-los em um bucket do S3, como:
  - AWS Database Migration Service, um serviço que pode carregar históricos de dados em massa e também processar registros de captura de dados de alteração (CDC) para sincronizar tabelas de origem e destino.
  - AWS Glue
  - Amazon EMR
  - Seu próprio código personalizado.
- O recurso de importação do DynamoDB do S3.

Etapas de migração off-line:

1. Crie um trabalho de ETL que possa consultar o banco de dados SQL, transformar os dados da tabela no formato JSON ou CSV do DynamoDB e salvá-los em um bucket do S3.



2. O recurso de importação do DynamoDB do S3 é invocado para criar uma tabela e carregar automaticamente os dados do bucket do S3.

A migração totalmente off-line é simples e direta, mas pode não ser popular entre proprietários e usuários de aplicações. Os usuários se beneficiariam se a aplicação pudesse fornecer níveis reduzidos de serviço durante a migração, em vez de nenhum serviço.

É possível adicionar uma funcionalidade para desabilitar as gravações durante a migração off-line, permitindo que as leituras continuem normalmente. Os usuários da aplicação ainda podem navegar e consultar com segurança os dados existentes enquanto os dados relacionais estão sendo migrados. Se é isso que você está procurando, continue lendo para saber mais sobre migrações híbridas.

Realizar uma migração híbrida para o DynamoDB

Embora todas as aplicações de banco de dados realizem operações de leitura e gravação, os tipos de operações de gravação que estão sendo realizadas devem ser considerados ao planejar uma migração híbrida ou on-line. As gravações de banco de dados podem ser classificadas em três buckets: inserções, atualizações e exclusões. Algumas aplicações podem não exigir o processamento imediato das exclusões. Por exemplo, essas aplicações podem adiar as exclusões para um processo de limpeza em massa no final do mês. Esses tipos de aplicações podem ser mais simples de migrar e, ao mesmo tempo, possibilitam tempo de atividade parcial.

Planejamento

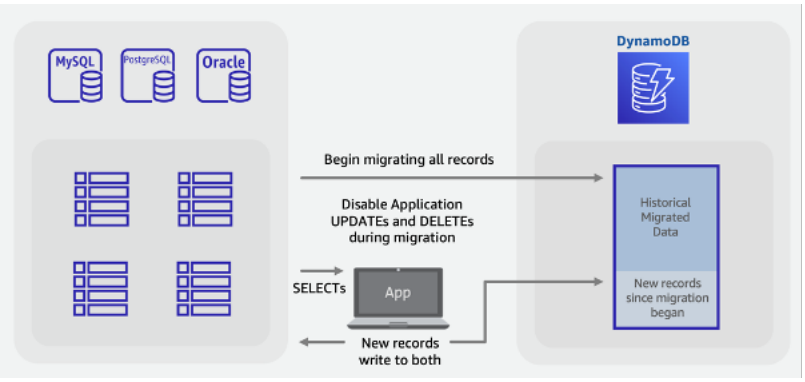
Realizar uma migração híbrida on-line/off-line com gravações duplas de aplicações

Ferramentas

- Um trabalho de ETL para extrair e transformar dados SQL e armazená-los em um bucket do S3, como:
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - Seu próprio código personalizado.

Etapas de migração híbridas:

1. Criar uma tabela do DynamoDB de destino. Essa tabela receberá dados históricos em massa e novos dados ativos.
2. Crie uma versão da aplicação herdada que tenha as exclusões e as atualizações desabilitadas enquanto executa todas as inserções como gravações duplas no banco de dados SQL e no DynamoDB
3. Iniciar o trabalho de ETL ou a tarefa AWS DMS para alocar os dados existentes e implantar a nova versão da aplicação ao mesmo tempo
4. Quando a alocação for concluída, o DynamoDB terá todos os registros novos e existentes e estará pronto para a substituição da aplicação.



> **Note**
>
> A tarefa de alocação grava diretamente do SQL no DynamoDB. Não é possível usar o recurso de importação do S3 como no exemplo de migração off-line, pois esse recurso cria uma nova tabela que não estará ativa até que o DynamoDB carregue os dados.

Realizar uma migração on-line para o DynamoDB migrando cada tabela 1:1

Muitos bancos de dados relacionais têm um recurso chamado captura de dados de alterações (CDC), em que o banco de dados permite que os usuários solicitem uma lista das alterações em uma tabela que ocorreram antes ou depois de um momento específico. A CDC usa logs internos para habilitar esse recurso e não exige nenhuma coluna de carimbo de data e hora para a tabela funcionar.

Ao migrar um esquema de tabelas SQL para um banco de dados NoSQL, convém combinar e remodelar os dados em menos tabelas. Isso permitirá que você colete dados em um único local e evite a necessidade de unir manualmente os dados relacionados em operações de leitura em várias etapas. No entanto, a modelagem de dados de tabela única nem sempre é necessária e, às vezes, você migrará tabelas 1 para 1 para o DynamoDB. Essas migrações de tabelas 1 para 1 são menos complicadas, pois é possível utilizar o recurso CDC do banco de dados de origem, usando ferramentas ETL comuns que comportam esse tipo de migração. Os dados de cada linha ainda podem ser transformados em novos formatos, mas o escopo de cada tabela permanece o mesmo.

Pense em migrar tabelas SQL de 1 para 1 para o DynamoDB, com a ressalva de que o DynamoDB não comporta junções do lado do servidor. Você precisará adicionar lógica à aplicação para combinar dados de várias tabelas.

Planejamento

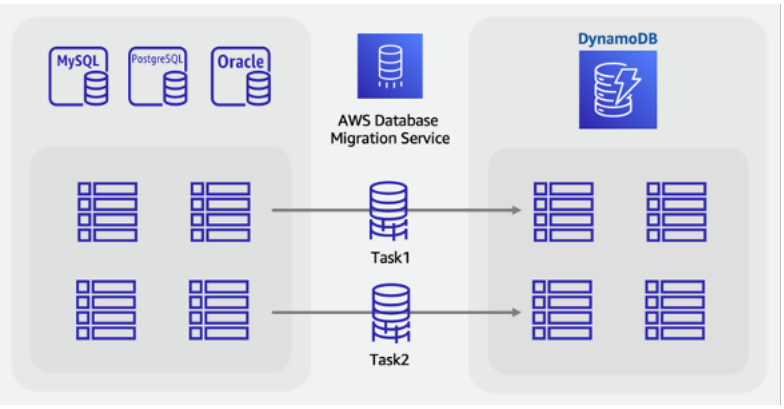
Realizar uma migração on-line de cada tabela para o DynamoDB usando o AWS DMS.

Ferramentas

- AWS DMS (DMS)

Etapas de migração on-line:

1. Identificar as tabelas no esquema de origem que serão migradas.
2. Criar o mesmo número de tabelas no DynamoDB com a estrutura de chaves da origem
3. Criar um servidor de replicação em AWS DMS e configure os endpoints de origem e destino.
4. Definir todas as transformações necessárias por linha (como colunas concatenadas ou conversão de datas no formato de string ISO-8601).
5. Criar uma tarefa de migração para cada tabela para Carregamento total e captura de dados de alterações.
6. Monitorar essas tarefas até que a fase de replicação contínua tenha começado.
7. Nesse ponto, é possível realizar qualquer auditoria de validação e, depois, transferir os usuários para a aplicação que lê e grava no DynamoDB.



Realizar uma migração on-line para o DynamoDB usando tabela de preparação personalizada.

De forma semelhante ao cenário de migração off-line, você pode combinar tabelas para utilizar padrões exclusivos de acesso ao NoSQL (por exemplo, transformar quatro tabelas herdadas em uma única tabela do DynamoDB). Um SQL `VIEW` pode fazer o trabalho no banco de dados de origem para preparar um único conjunto de dados representando todas as quatro tabelas em um conjunto.

No entanto, para migrações on-line com dados dinâmicos e variáveis, não é possível utilizar os recursos da CDC, pois eles não são aceitos pelas `VIEW`s. Se suas tabelas incluírem uma coluna de carimbo de data e hora da última atualização e elas forem incorporadas à `VIEW`, você poderá criar um trabalho de ETL personalizado que as use para obter uma carga em massa com a sincronização.

Uma nova abordagem para esse desafio é usar recursos SQL padrão, como visualizações, procedimentos armazenados e gatilhos, para criar uma tabela SQL que esteja no formato final desejado do NoSQL do DynamoDB.

Se o servidor de banco de dados tiver capacidade sobressalente, será possível criar essa única tabela de preparação antes do início da migração. Isso seria feito escrevendo um procedimento armazenado que leria as tabelas existentes, transformaria os dados conforme necessário e gravaria na nova tabela de preparação. É possível adicionar um conjunto de gatilhos para replicar as alterações nas tabelas na tabela de preparação em tempo real. Se os gatilhos não forem permitidos de acordo com a política da empresa, as alterações nos procedimentos armazenados poderão ter o mesmo resultado. Você adicionaria algumas linhas de código a qualquer procedimento que grave dados para, além disso, gravar as mesmas alterações na tabela de preparação.

Ter essa tabela de preparação pronta e totalmente sincronizada com as tabelas de aplicações herdadas proporcionará um excelente ponto de partida para uma migração dinâmica. Ferramentas que usam a CDC do banco de dados para realizar migrações em tempo real, como o AWS DMS, agora podem ser usadas nessa tabela. Uma vantagem dessa abordagem é que ela usa habilidades e recursos SQL conhecidos disponíveis no mecanismo de banco de dados relacional.

Planejamento

Executar uma migração on-line com uma tabela de preparação SQL usando o AWS DMS.

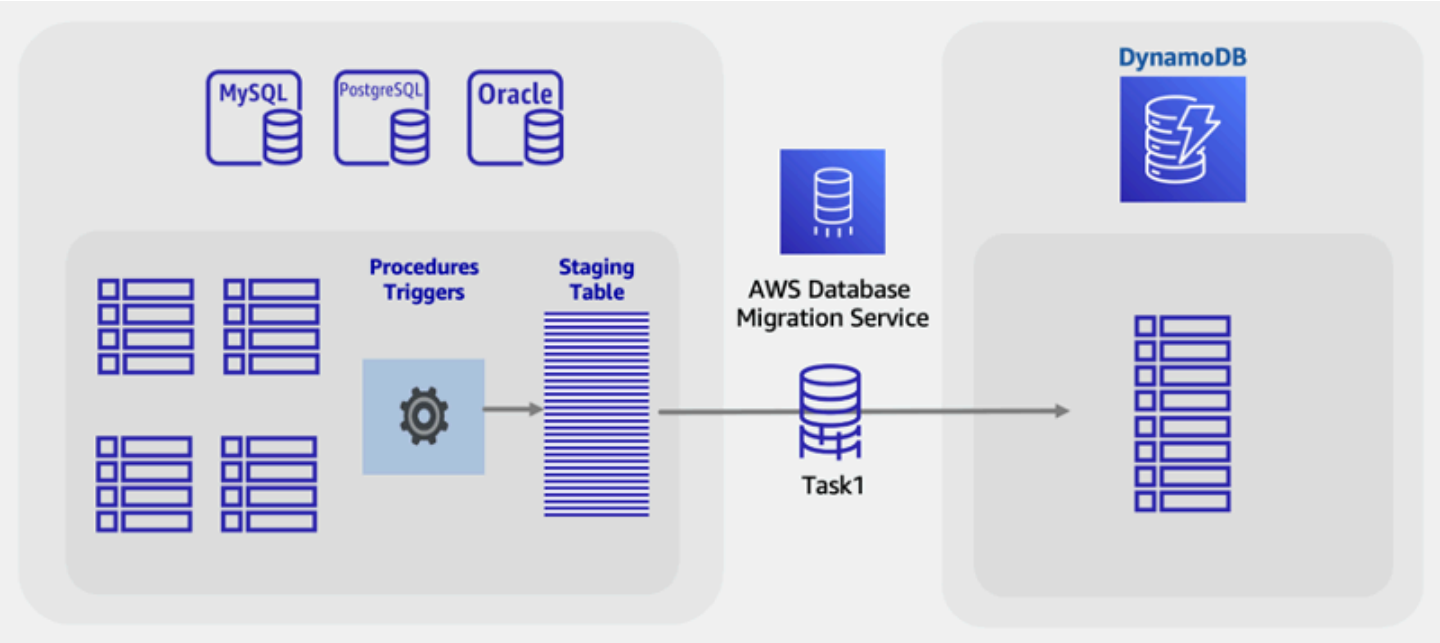
Ferramentas

- Procedimentos ou gatilhos armazenados em SQL personalizados
- AWS DMS

Etapas de migração on-line:

1. No mecanismo de banco de dados relacional de origem, verifique se há espaço em disco e capacidade de processamento extras.
2. Criar uma tabela de preparação no banco de dados SQL, com carimbos de data e hora ou recursos de CDC habilitados.
3. Gravar e realizar um procedimento armazenado para copiar os dados da tabela relacional existente na tabela de preparação.
4. Implantar gatilhos ou modificar os procedimentos existentes para gravação dupla na nova tabela de preparação enquanto executa gravações normais nas tabelas existentes.

5. Execute AWS DMS para migrar e sincronizar essa tabela de origem com uma tabela de destino do DynamoDB.



Este guia apresentou várias considerações e abordagens para migrar dados de bancos de dados relacionais para o DynamoDB, com foco em minimizar o tempo de inatividade e usar ferramentas e técnicas comuns de banco de dados. Para obter mais informações, consulte as informações a seguir.

- AWS DMS Manual do usuário
- Guia do usuário do AWS Glue
- Práticas recomendadas para migrar do RDBMS para o DynamoDB

NoSQL Workbench para DynamoDB

O NoSQL Workbench para Amazon DynamoDB é uma aplicação GUI multiplataforma do lado do cliente para operações e desenvolvimento de bancos de dados modernos. Ele está disponível para Windows, macOS e Linux. O NoSQL Workbench é uma ferramenta de desenvolvimento visual que fornece recursos de modelagem de dados, visualização de dados e desenvolvimento de consultas para ajudar você a projetar, criar, consultar e gerenciar tabelas do DynamoDB. O NoSQL Workbench agora inclui o DynamoDB local como parte opcional do processo de instalação, o que facilita a modelagem de dados no DynamoDB local. Para saber mais sobre o DynamoDB local e seus requisitos, consulte Configurar o DynamoDB local (versão para download) .

Modelagem de dados

Com o NoSQL Workbench para DynamoDB, você pode criar novos modelos de dados a partir de ou projetar modelos com base em modelos de dados existentes que satisfaçam os padrões de acesso a dados das suas aplicações. Pode também importar e exportar o modelo de dados designado no final do processo. Para ter mais informações, consulte Criar modelos de dados com o NoSQL Workbench.

Visualização de dados

O visualizador de modelo de dados oferece uma tela na qual você pode mapear consultas e visualizar os padrões de acesso (facetas) do aplicativo sem precisar escrever código. Cada faceta corresponde a um padrão de acesso diferente no DynamoDB. Você pode gerar dados de amostra automaticamente para uso no modelo de dados. Para ter mais informações, consulte Visualizar padrões de acesso a dados.

Criação de operação

O NoSQL Workbench oferece uma avançada interface gráfica do usuário para você desenvolver e testar consultas. Você pode usar o criador de operações para visualizar, explorar e consultar conjuntos de dados dinâmicos. O criador de operações estruturadas é compatível com expressão de projeção e expressão de condição, além de gerar código de amostra em vários idiomas. Você pode clonar tabelas diretamente de uma conta do Amazon DynamoDB para outra em diferentes regiões. Também é possível clonar tabelas diretamente entre o DynamoDB local e uma conta do Amazon DynamoDB para uma cópia mais rápida do esquema de chaves da tabela (e, opcionalmente, do esquema e dos itens do GSI) entre seus ambientes de desenvolvimento. Para ter mais informações, consulte Explorar conjuntos de dados e criar operações com o NoSQL Workbench.

O vídeo a seguir detalha conceitos de modelagem de dados com o NoSQL Workbench.

Tópicos

- Baixar o NoSQL Workbench para DynamoDB
- Instalar NoSQL Workbench para DynamoDB
- Criar modelos de dados com o NoSQL Workbench
- Visualizar padrões de acesso a dados
- Explorar conjuntos de dados e criar operações com o NoSQL Workbench
- Modelos de dados de amostra para o NoSQL Workbench
- Histórico de versões do NoSQL Workbench

Baixar o NoSQL Workbench para DynamoDB

Siga estas instruções para baixar o NoSQL Workbench e o DynamoDB local* para Amazon DynamoDB.

Pré-requisitos

Há dois softwares obrigatórios para as instalações do Ubuntu: libfuse2 e curl.

libfuse2

A partir do Ubuntu 22.04, o libfuse2 não é mais instalado por padrão. Para resolver isso, execute `sudo add-apt-repository universe && sudo apt install libfuse2` para instalar a versão mais recente do Ubuntu.

curl

Atualize o Ubuntu e execute `sudo apt update && sudo apt upgrade`.

Depois, instale cURL e execute: `sudo apt install curl`.

Como baixar o NoSQL Workbench e o DynamoDB local

1. Baixe a versão apropriada do NoSQL Workbench para seu sistema operacional.

Sistema operacional	Link para fazer download
macOS (Intel)**	Baixar para macOS (Intel)
macOS (chip da Apple)	Baixar para macOS (chip da Apple)
Windows	Download para Windows
Linux***	Download para Linux

* O NoSQL Workbench inclui o DynamoDB local como parte opcional do processo de instalação.

** Se aparecer uma mensagem de aviso ao tentar abrir o NoSQL Workbench informando que o aplicativo não está registrado na Apple por um desenvolvedor identificado, faça o seguinte:

1. Localize o aplicativo e abra-o.
2. Clique com a tecla Control no ícone do aplicativo e selecione Abrir no menu de atalho.

   O aplicativo será salvo como uma exceção nas configurações de segurança. Abra o aplicativo clicando duas vezes nele, assim como com qualquer aplicativo registrado.

*** O NoSQL Workbench é compatível com Ubuntu 12.04, Fedora 21 e Debian 8 ou quaisquer versões mais recentes dessas distribuições do Linux.

2. Inicie a aplicação que baixou e siga as etapas em Instalar o NoSQL Workbench.

> **Note**
>
> O Ambiente de Execução Java (JRE) versão 11.x ou mais recente é necessário para executar o DynamoDB local.

Instalar NoSQL Workbench para DynamoDB

Siga estas etapas para instalar o NoSQL Workbench e o DynamoDB local em uma plataforma compatível.

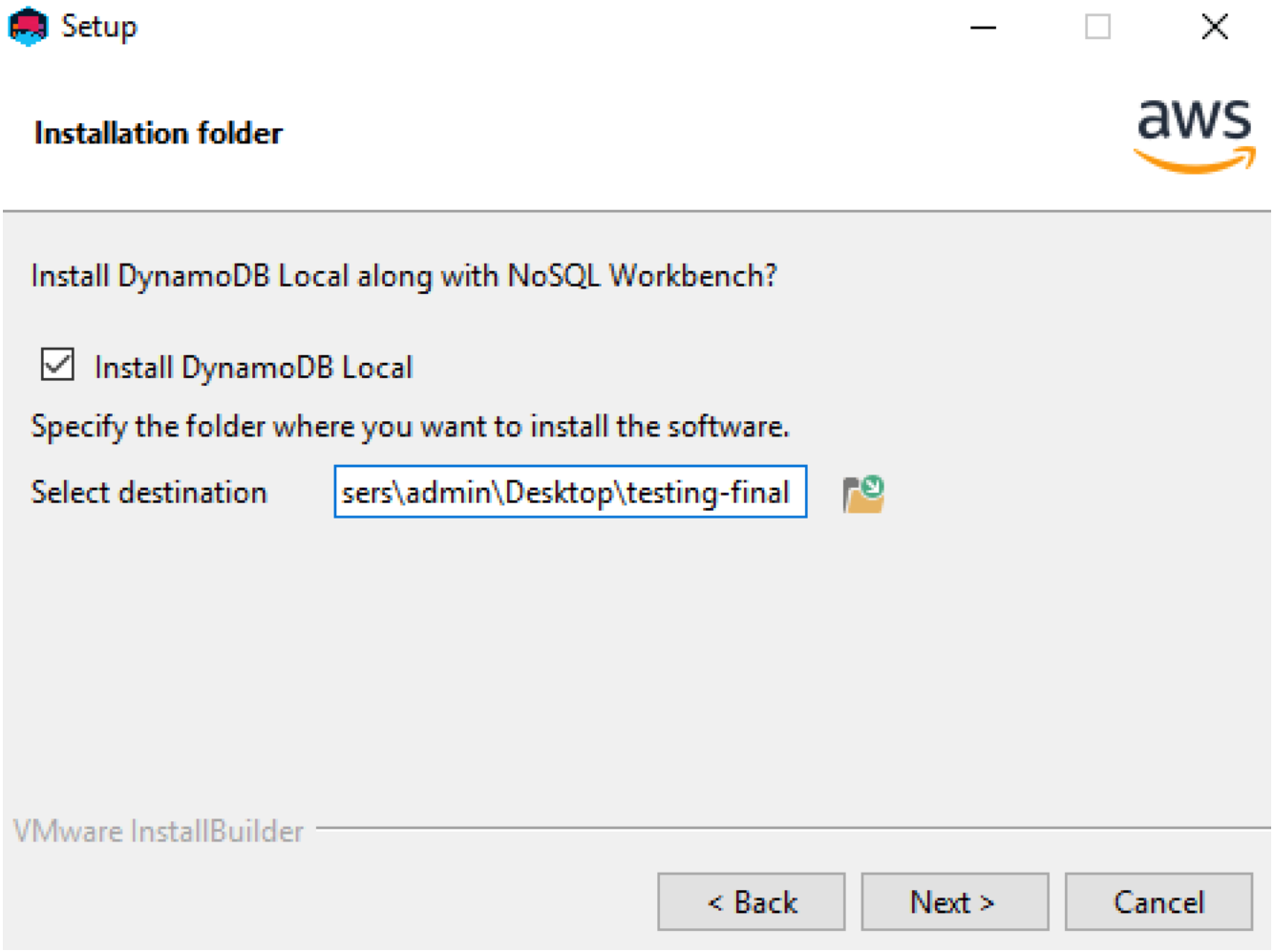
Windows

Para instalar o NoSQL Workbench no Windows

1. Execute o aplicativo instalador do NoSQL Workbench e escolha o idioma de configuração. Depois, escolha OK para iniciar a configuração. Para obter mais informações sobre como baixar o NoSQL Workbench, consulte Baixar o NoSQL Workbench para DynamoDB.
2. Escolha Next (Próximo) para continuar a configuração e escolha Next (Próximo) na tela a seguir.
3. Por padrão, a caixa de seleção Instalar DynamoDB local está marcada para incluir o DynamoDB local como parte da instalação. Manter essa opção selecionada garante que o DynamoDB local seja instalado e que o caminho de destino seja igual ao caminho de instalação do NoSQL Workbench. Se a caixa de seleção dessa opção for desmarcada, a instalação do DynamoDB local será ignorada e o caminho de instalação será somente para o NoSQL Workbench.

   Escolha o destino em que você deseja instalar o software e escolha Next (Próximo).

   > **Note**
   >
   > Se você optou por não incluir o DynamoDB local como parte da configuração, desmarque a caixa de seleção Instalar DynamoDB local, escolha Próximo e prossiga para a etapa 6. Em outro momento, você poderá baixar o DynamoDB local separadamente, como uma instalação independente. Para ter mais informações, consulte Configurar o DynamoDB local (versão para download) .



4. Escolha o número da porta a ser usada pelo DynamoDB local. A porta padrão é 8000. Depois de inserir o número da porta, escolha Next (Próximo).
5. Escolha Next (Próximo) para iniciar a configuração.
6. Após o término da configuração, escolha Finish (Concluir) para fechar a tela de configuração.
7. Abra a aplicação no caminho de instalação, como /Programs/DynamoDBWorkbench/.

macOS

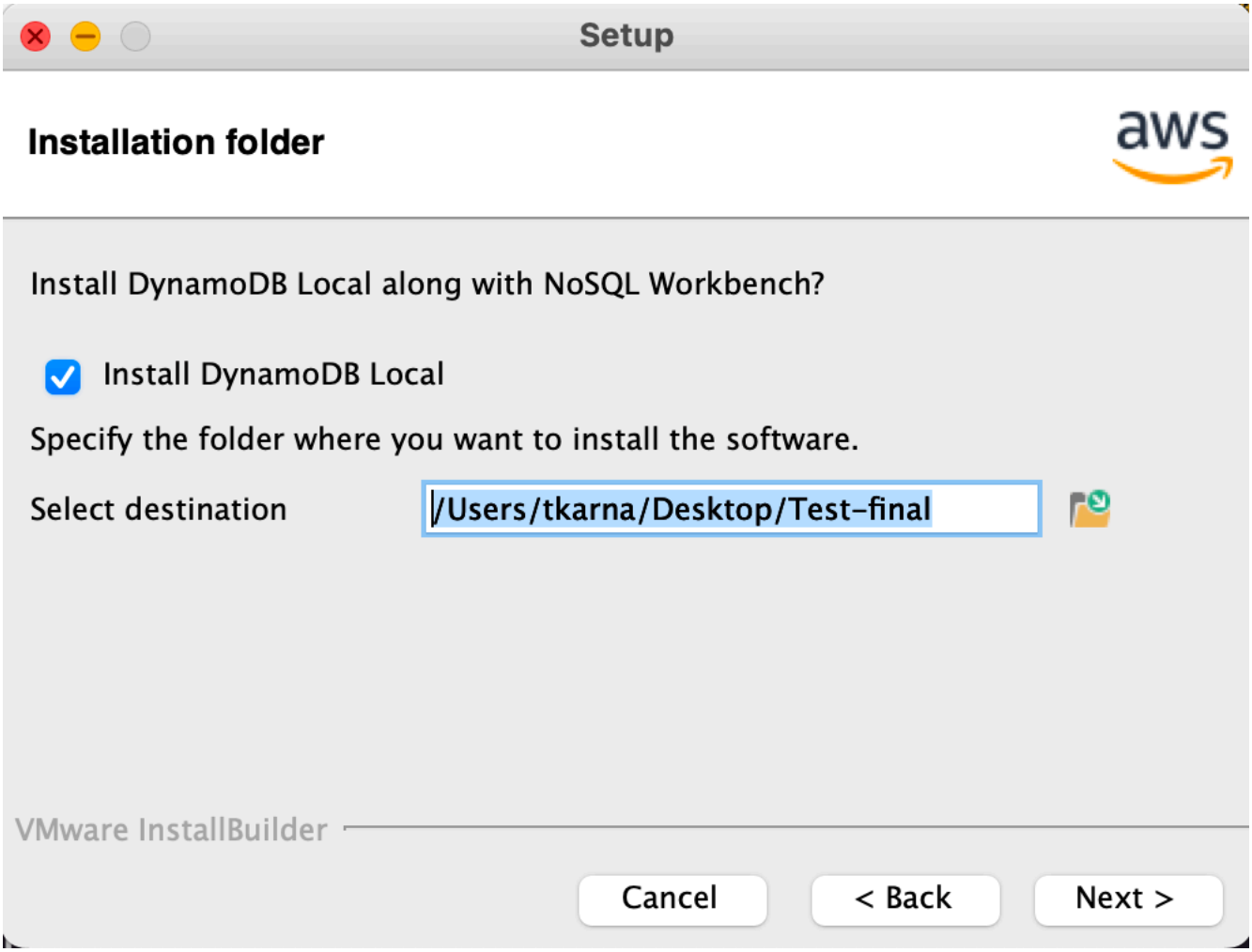
Para instalar o NoSQL Workbench no macOS

1. Execute o aplicativo instalador do NoSQL Workbench e escolha o idioma de configuração. Depois, escolha OK para iniciar a configuração. Para obter mais informações sobre como baixar o NoSQL Workbench, consulte Baixar o NoSQL Workbench para DynamoDB.

2. Escolha Next (Próximo) para continuar a configuração e escolha Next (Próximo) na tela a seguir.
3. Por padrão, a caixa de seleção Instalar DynamoDB local está marcada para incluir o DynamoDB local como parte da instalação. Manter essa opção selecionada garante que o DynamoDB local seja instalado e que o caminho de destino seja igual ao caminho de instalação do NoSQL Workbench. Se essa opção for desmarcada, a instalação do DynamoDB local será ignorada e o caminho de instalação será somente para o NoSQL Workbench.

   Escolha o destino em que você deseja instalar o software e escolha Next (Próximo).

   > **Note**
   >
   > Se você optou por não incluir o DynamoDB local como parte da configuração, desmarque a caixa de seleção Instalar DynamoDB local, escolha Próximo e prossiga para a etapa 6. Em outro momento, você poderá baixar o DynamoDB local separadamente, como uma instalação independente. Para ter mais informações, consulte Configurar o DynamoDB local (versão para download) .



4. Escolha o número da porta a ser usada pelo DynamoDB local. A porta padrão é 8000. Depois de inserir o número da porta, escolha Next (Próximo).
5. Escolha Next (Próximo) para iniciar a configuração.
6. Após o término da configuração, escolha Finish (Concluir) para fechar a tela de configuração.
7. Abra a aplicação no caminho de instalação, como /Applications/DynamoDBWorkbench/.

> **Note**
>
> O NoSQL Workbench para macOS realiza atualizações automáticas. Para receber notificações sobre as atualizações, ative o acesso à notificação para o NoSQL Workbench em System Preferences > Notifications (Preferências do Sistema > Notificações).

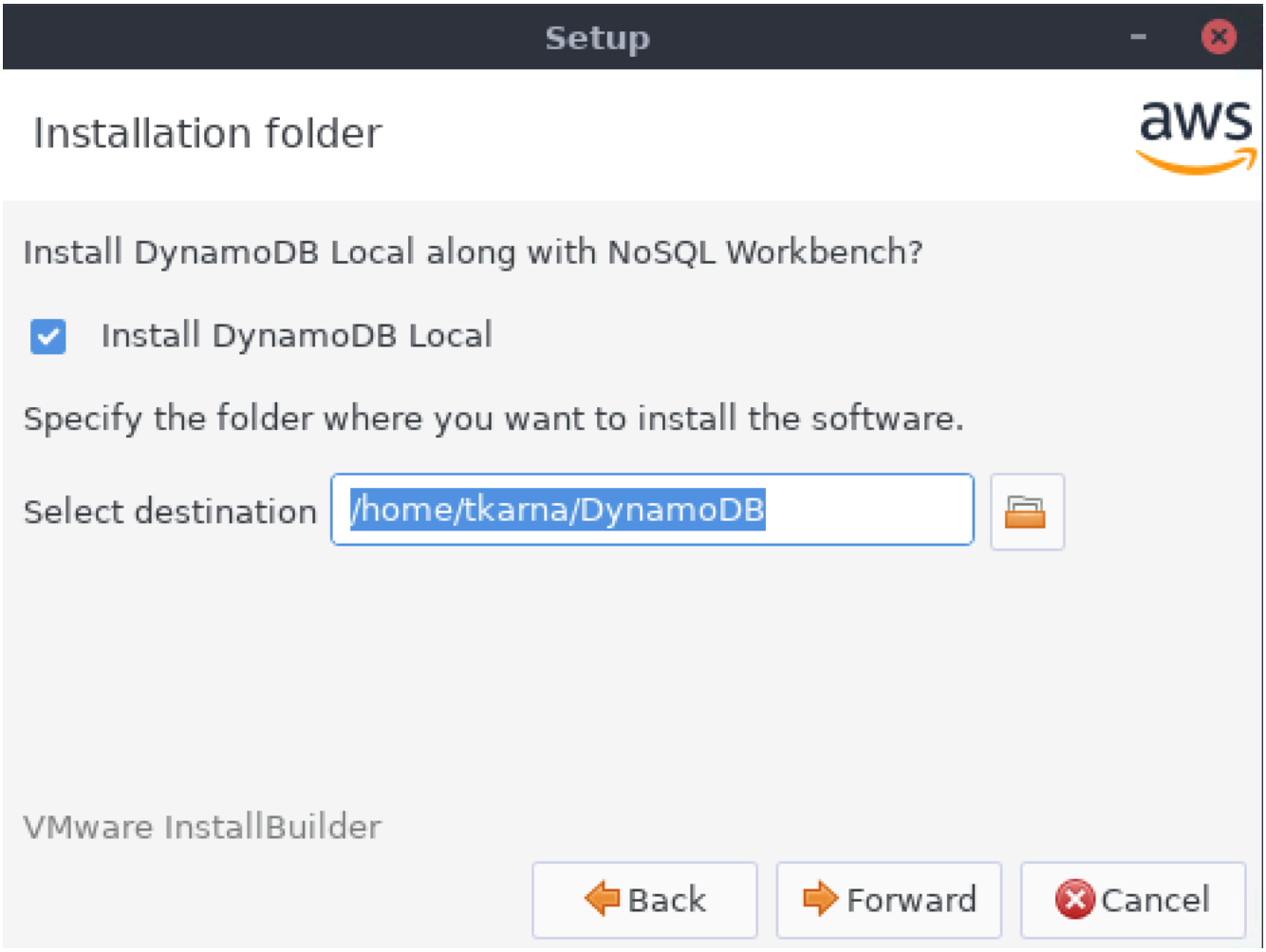
Linux

Para instalar o NoSQL Workbench no Linux

1. Execute o aplicativo instalador do NoSQL Workbench e escolha o idioma de configuração. Depois, escolha OK para iniciar a configuração. Para obter mais informações sobre como baixar o NoSQL Workbench, consulte Baixar o NoSQL Workbench para DynamoDB.
2. Escolha Forward (Encaminhar) para continuar a configuração e escolha Forward (Encaminhar) na tela a seguir.
3. Por padrão, a caixa de seleção Instalar DynamoDB local está marcada para incluir o DynamoDB local como parte da instalação. Manter essa opção selecionada garante que o DynamoDB local seja instalado e que o caminho de destino seja igual ao caminho de instalação do NoSQL Workbench. Se essa opção for desmarcada, a instalação do DynamoDB local será ignorada e o caminho de instalação será somente para o NoSQL Workbench.

   Escolha o destino em que você deseja instalar o software e escolha Forward (Encaminhar).

   > **Note**
   >
   > Se você optou por não incluir o DynamoDB local como parte da configuração, desmarque a caixa de seleção Instalar DynamoDB local, escolha Avançar e prossiga para a etapa 6. Em outro momento, você poderá baixar o DynamoDB local separadamente, como uma instalação independente. Para ter mais informações, consulte Configurar o DynamoDB local (versão para download) .



4. Escolha o número da porta a ser usada pelo DynamoDB local. A porta padrão é 8000. Depois de inserir o número da porta, escolha Forward (Encaminhar).
5. Escolha Forward (Encaminhar) para iniciar a configuração.
6. Após o término da configuração, escolha Finish (Concluir) para fechar a tela de configuração.
7. Abra a aplicação no caminho de instalação, como /usr/local/programs/DynamoDBWorkbench/.

Como iniciar um AppImage no Linux

1. Torne o arquivo AppImage executável:

   chmod +x noSQL-workbench-linux.AppImage

Substitua noSQL-workbench-linux.AppImage pelo nome real do arquivo do AppImage que você baixou.

2. Execute o AppImage:

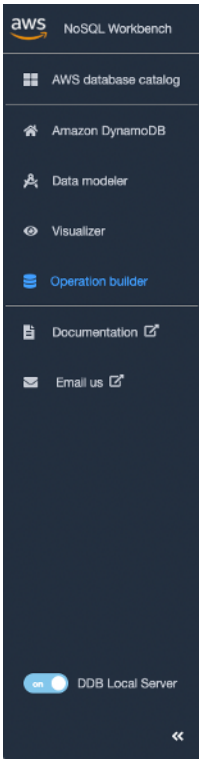
   ./noSQL-workbench-linux.AppImage

   Isso iniciará a aplicação NoSQL Workbench.

> **Note**
>
> Dependendo da distribuição Linux, talvez seja necessário instalar dependências adicionais para que o AppImage funcione corretamente. Se você encontrar algum problema, consulte a documentação fornecida pelos desenvolvedores do AppImage ou busque suporte da comunidade.

> **Note**
>
> Se você optar por instalar o DynamoDB local como parte da instalação do NoSQL Workbench, o DynamoDB local será pré-configurado com as opções padrão. Para editar as opções padrão, modifique o script DDBLocalStart localizado no diretório /resources/DDBLocal_Scripts/. Você pode encontrá-lo no caminho fornecido durante a instalação. Para saber mais sobre as opções do DynamoDB local, consulte Observações sobre o uso do DynamoDB local.

Se optar por instalar o DynamoDB local como parte da instalação do NoSQL Workbench, você terá acesso a um botão para habilitar e desabilitar o DynamoDB local, conforme mostrado na imagem a seguir.



Criar modelos de dados com o NoSQL Workbench

Você pode usar a ferramenta de modelador de dados no NoSQL Workbench para Amazon DynamoDB para criar novos modelos de dados ou para criar modelos com base em modelos de dados existentes que satisfaçam os padrões de acesso a dados das suas aplicações. O modelador de dados inclui alguns modelos de dados de exemplo para ajudar você a começar.

Tópicos

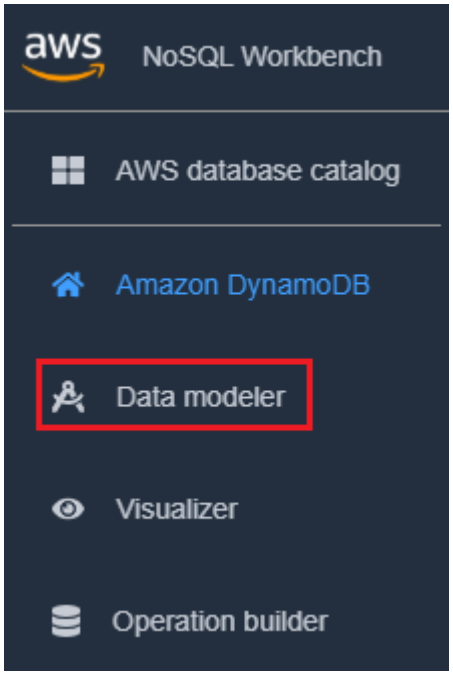
- Criar um novo modelo de dados
- Importar um modelo de dados existente
- Exportar um modelo de dados
- Editar um modelo de dados existente

Criar um novo modelo de dados

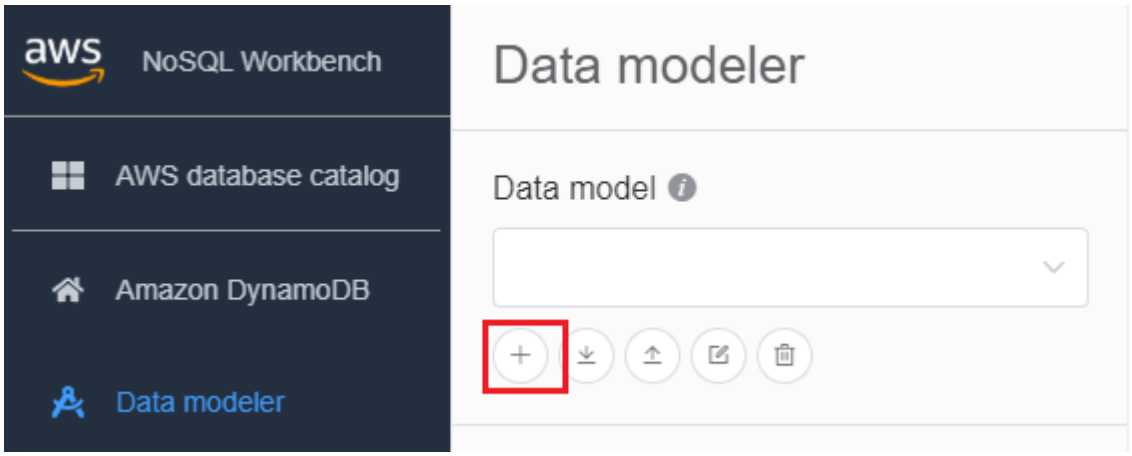
Siga estas etapas para criar um novo modelo de dados no Amazon DynamoDB usando o NoSQL Workbench.

Para criar um novo modelo de dados

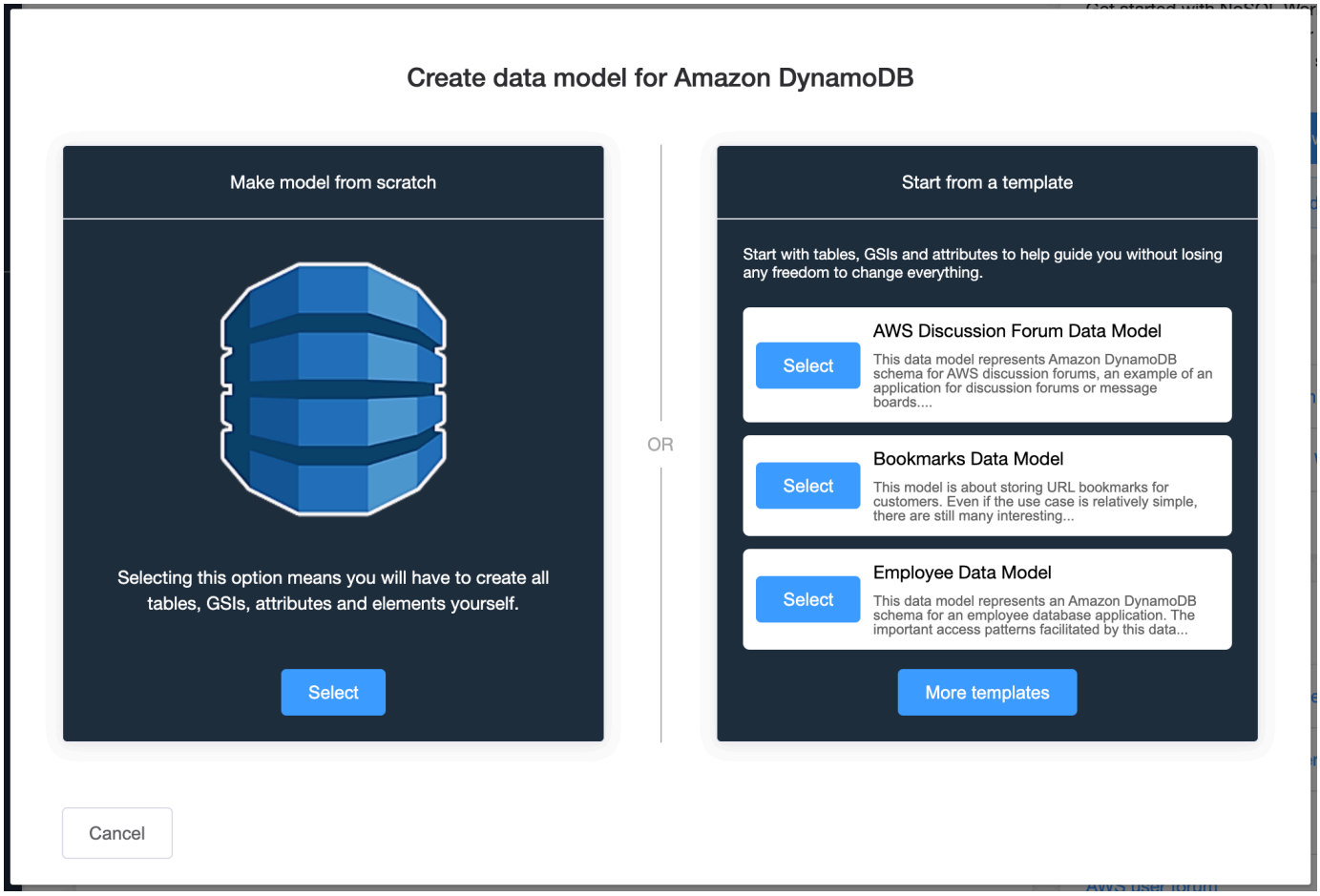
1. Abra o NoSQL Workbench e, no painel de navegação no lado esquerdo, escolha o ícone do Data modeler (Modelador de dados).



2. Escolha Create data model (Criar modelo de dados).

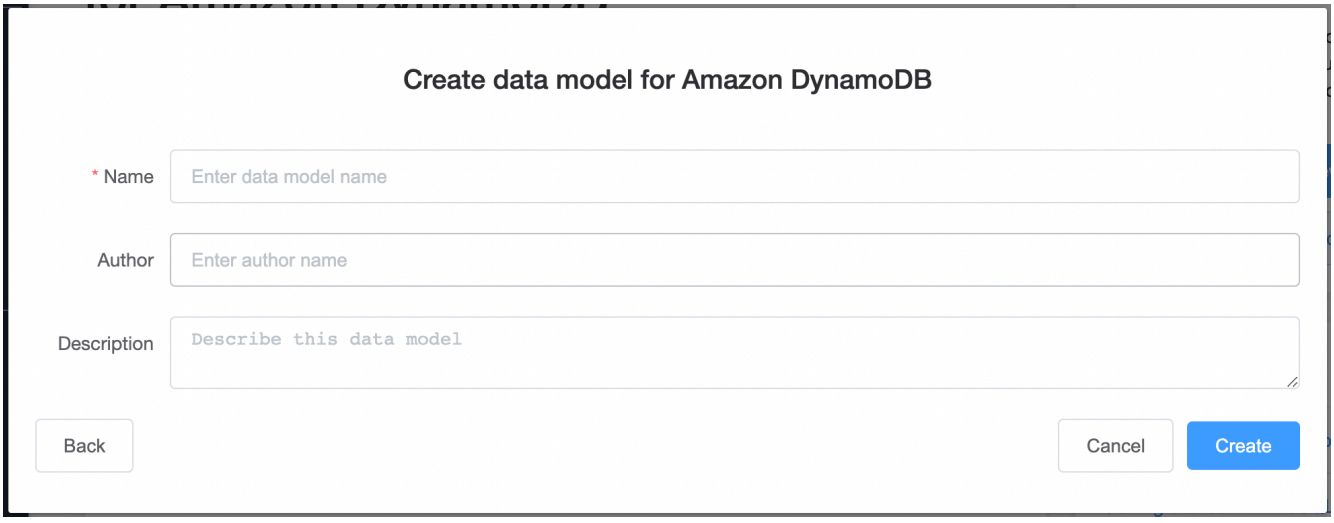


Criar modelo de dados tem duas opções: criar modelo do zero e começar a partir de um modelo.



Make model from scratch

Para criar um modelo a partir do zero, digite um nome, um autor e uma descrição para o modelo de dados. Quando terminar, escolha Create (Criar).



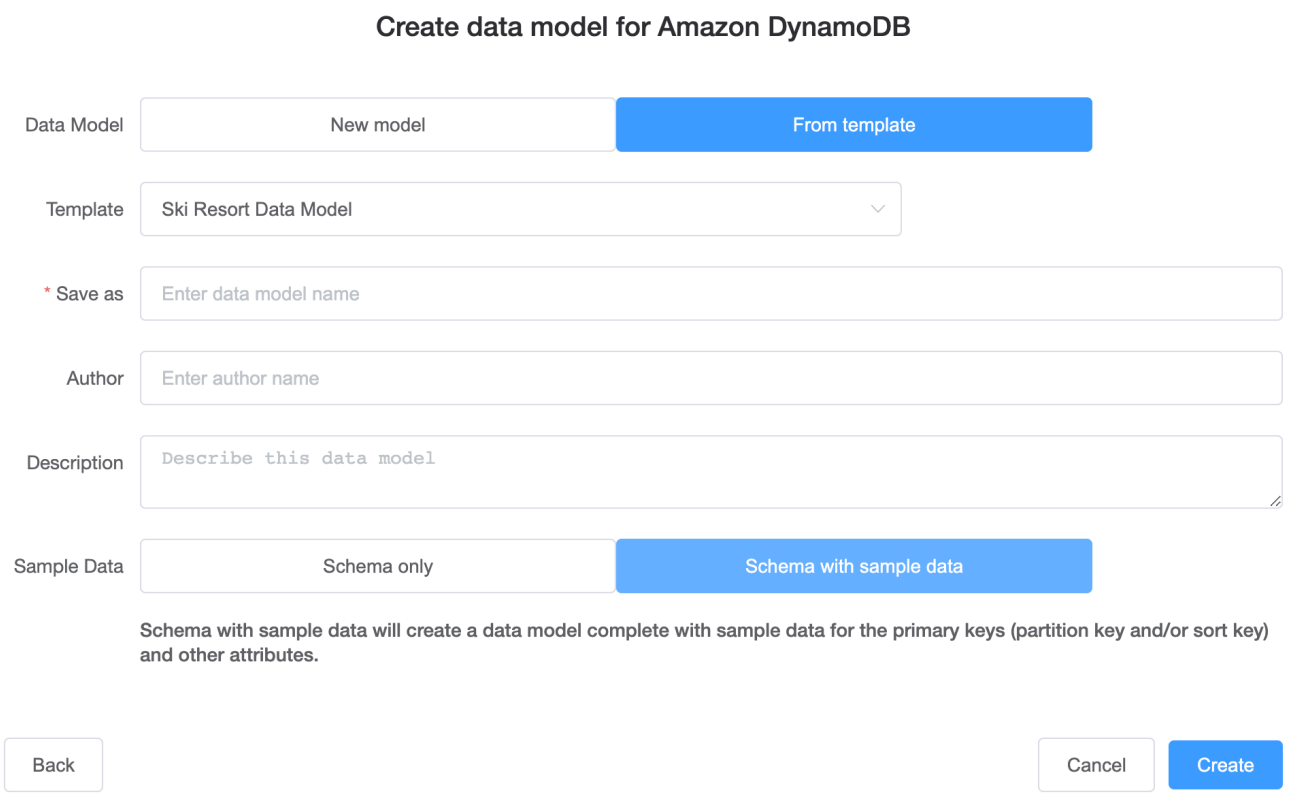
Start from a template

Começar com um modelo permite que você escolha um modelo de exemplo para começar. Escolha More templates (Mais modelos) para ver mais opções de modelos. Escolha Select (Selecionar) para o modelo que deseja usar.

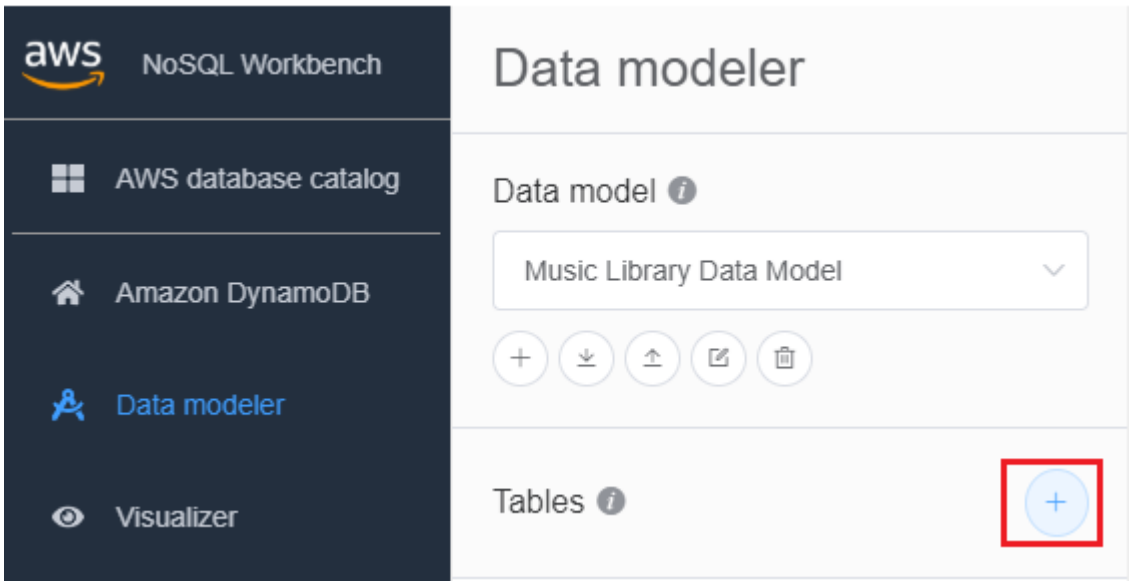
Insira um nome, autor e descrição do modelo de dados para o modelo selecionado. Você pode escolher entre Schema only (Somente esquema) e Schema with sample data (Esquema com dados de exemplo).

- O Schema only (Somente esquema) cria um modelo de dados vazio com a chave primária (chave de partição e classificação) e outros atributos.
- O Schema with sample data Esquema com dados de exemplo) criará um modelo de dados completo com dados de exemplo para a chave primária (chave de partição e classificação) e outros atributos.

Quando essas informações estiverem completas, escolha Create (Criar) para criar o modelo.



3. Com o modelo criado, escolha Add table (Adicionar tabela).



Para obter mais informações sobre tabelas, consulte Trabalhar com tabelas no DynamoDB.

4. Especifique o seguinte:

- Table name (Nome da tabela): insira um nome exclusivo para a tabela.
- Chave de partição: insira um nome de chave de partição e especifique o respectivo tipo. Opcionalmente, você também pode selecionar um formato de tipo de dados mais granular para geração de dados de amostra.
- Se desejar adicionar uma chave de classificação:
  1. Selecione Add sort key (Adicionar chave de classificação).
  2. Especifique o nome da chave de classificação e seu tipo. Opcionalmente, você também pode selecionar um formato de dados mais granular para geração de dados de amostra.

> **Note**
>
> Para saber mais sobre design de chave primária, como projetar e usar chaves de partição de forma eficaz e usar chaves de classificação, consulte o seguinte:
>
> - Chave primária
> - Práticas recomendadas para projetar e usar chaves de partição de maneira eficaz no DynamoDB
> - Práticas recomendadas para usar chaves de classificação para organizar dados no DynamoDB

5. Para adicionar outros atributos, faça o seguinte para cada atributo:

   1. Escolha Adicionar um atributo.
   2. Especifique o nome e o tipo do atributo. Opcionalmente, você também pode selecionar um formato de dados mais granular para geração de dados de amostra.

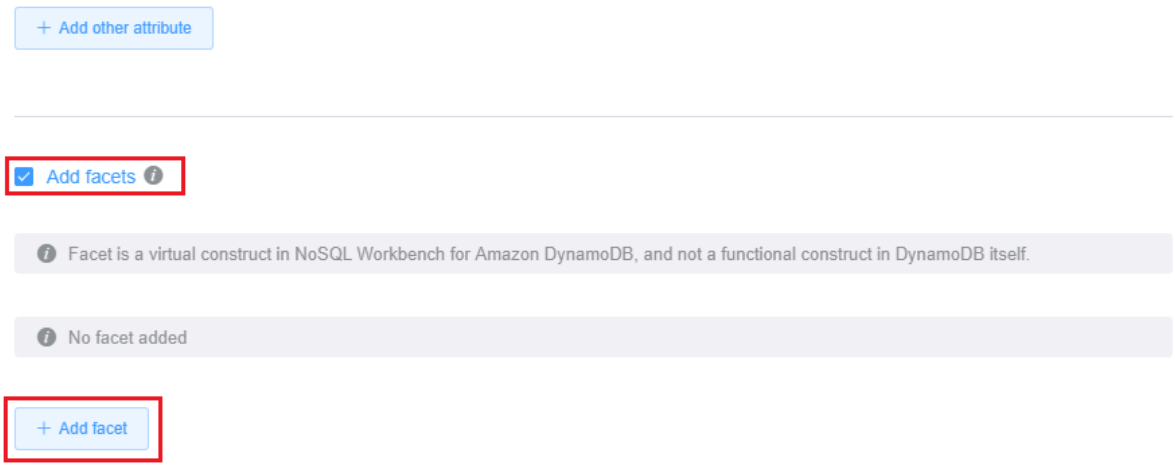
6. Adicionar uma faceta:

   Opcionalmente, é possível adicionar uma faceta. Faceta é uma estrutura virtual no NoSQL Workbench. Não é uma estrutura funcional no próprio DynamoDB.

   > **Note**
   >
   > No NoSQL Workbench, as facetas ajudam a visualizar os diferentes padrões de acesso a dados de uma aplicação para o Amazon DynamoDB com apenas um subconjunto dos dados em uma tabela. Para saber mais sobre facetas, consulte Exibir padrões de acesso a dados.

   Para adicionar uma faceta,

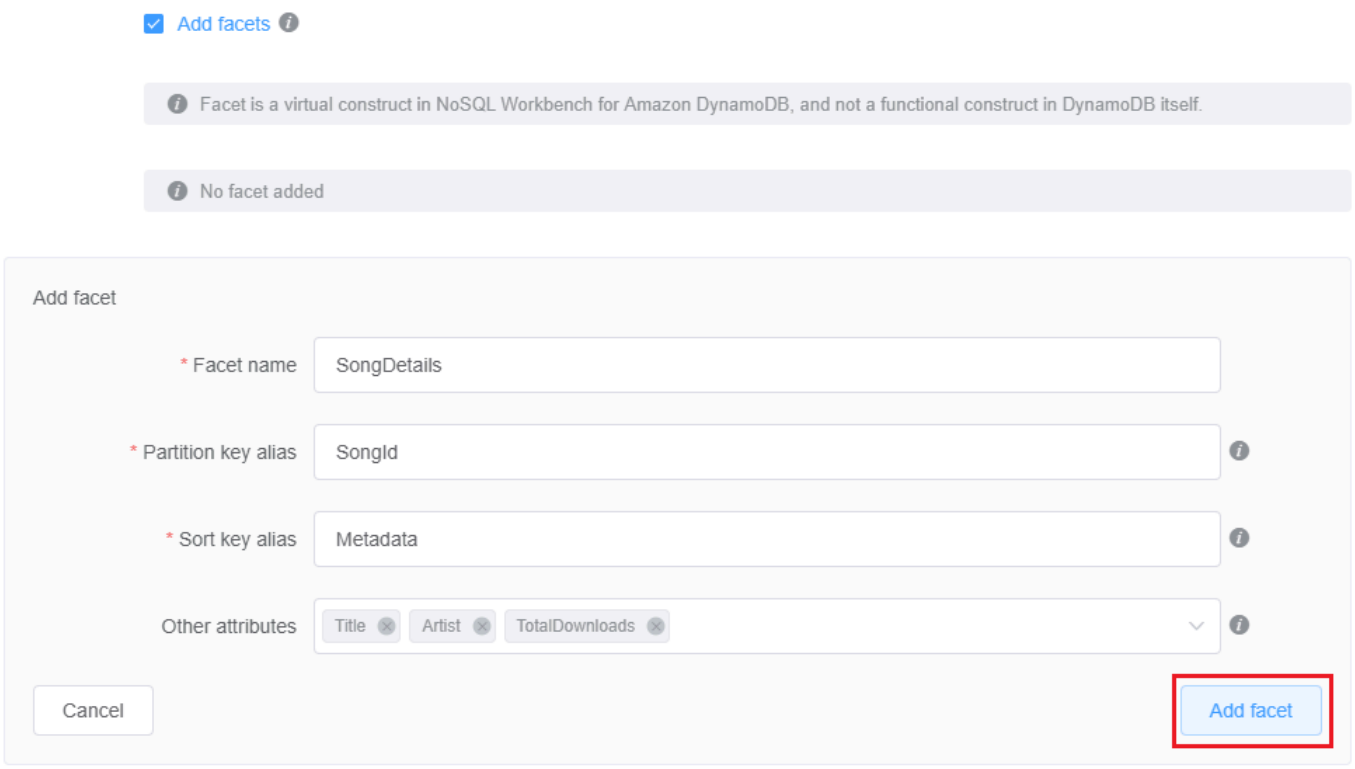
   - Selecione Add facets (Adicionar facetas).
   - Escolha Add facet (Adicionar faceta).

   - Especifique o seguinte:
     - O Facet name (Nome da faceta).
     - Um alias de chave de partição, para ajudar a distinguir essa visualização de facetas.

- Um Sort key alias (Alias da chave de classificação).
- Escolha os Other attributes (Outros atributos) que fazem parte desta faceta.

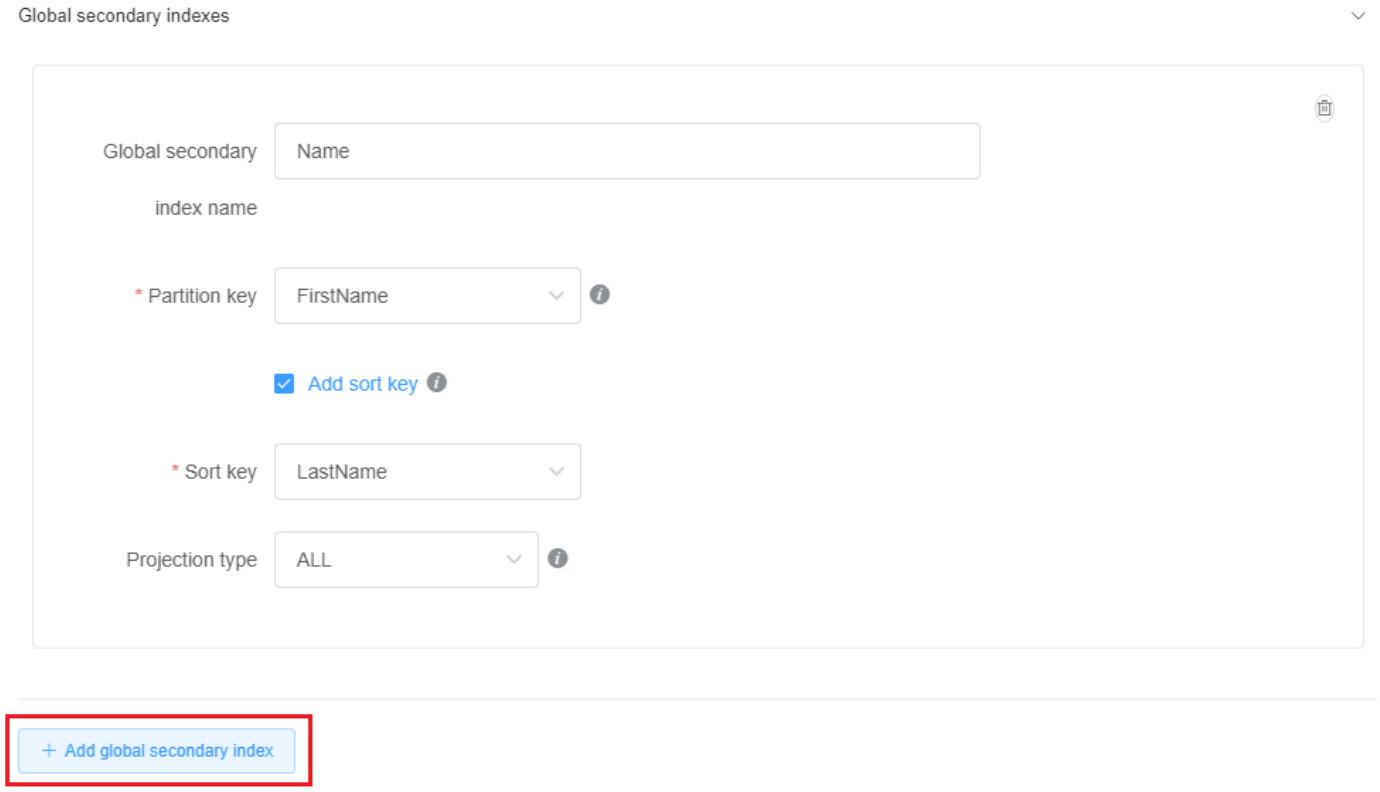
Escolha Add facet (Adicionar faceta).



Repita esta etapa se quiser adicionar mais facetas.

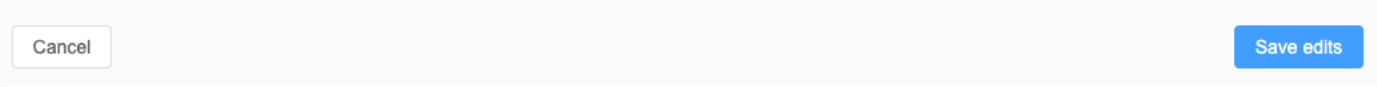
7. Se desejar adicionar um índice secundário global, escolha Add global secondary index (Adicionar índice secundário global).

   Especifique o Global secondary index name (Nome do índice secundário global), o atributo Partition key (Chave de partição) e o Projection type (Tipo de projeção).



Para obter mais informações sobre como trabalhar com índices secundários globais no DynamoDB, consulte Índices secundários globais.

8. Salve as edições nas configurações da sua tabela.



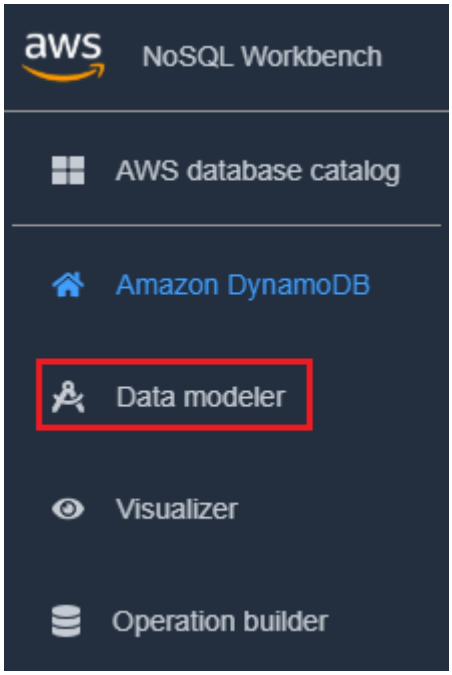
Para obter mais informações sobre a operação da API `CreateTable`, consulte CreateTable na Referência da API do Amazon DynamoDB.

Importar um modelo de dados existente

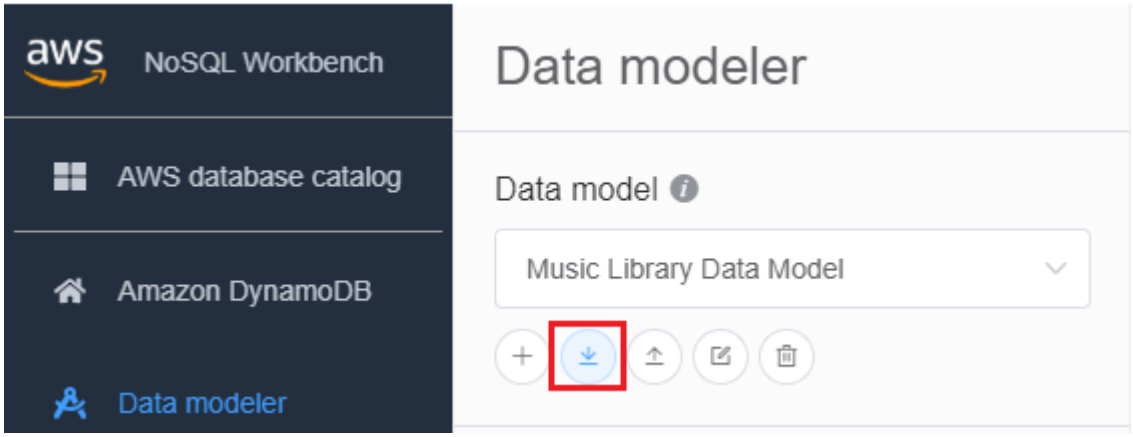
Use o NoSQL Workbench para Amazon DynamoDB para criar um modelo de dados importando e modificando um modelo existente. Você pode importar modelos de dados no formato de modelo NoSQL Workbench ou no Formato de modelo AWS CloudFormation JSON.

Para importar um modelo de dados

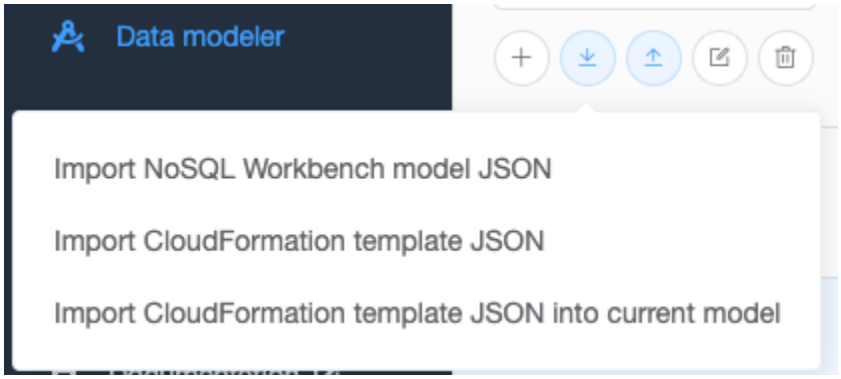
1. No NoSQL Workbench, no painel de navegação no lado esquerdo, escolha o ícone Data modeler (Modelador de dados).



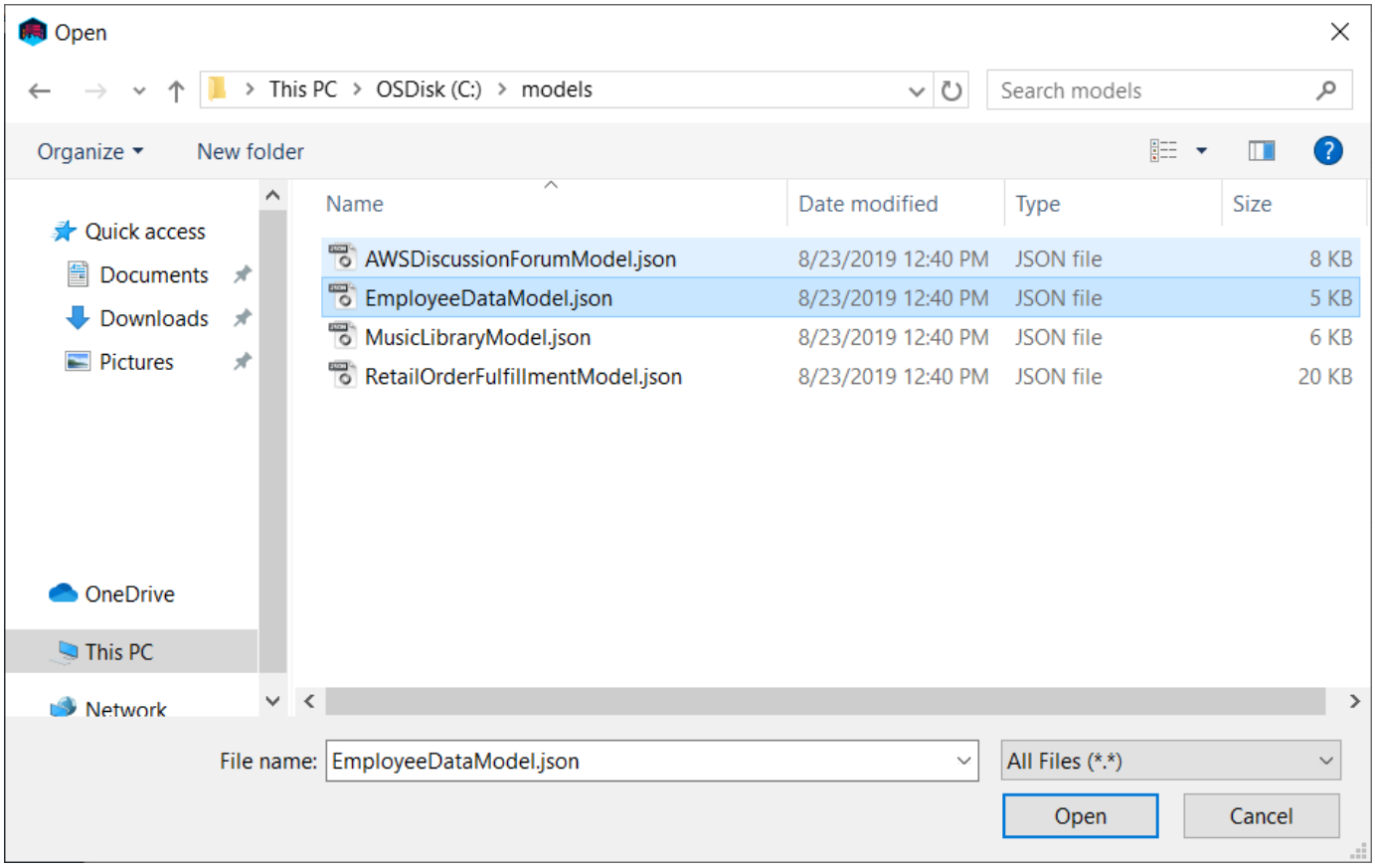
2. Passe o mouse sobre Import data model (Importar modelo de dados).



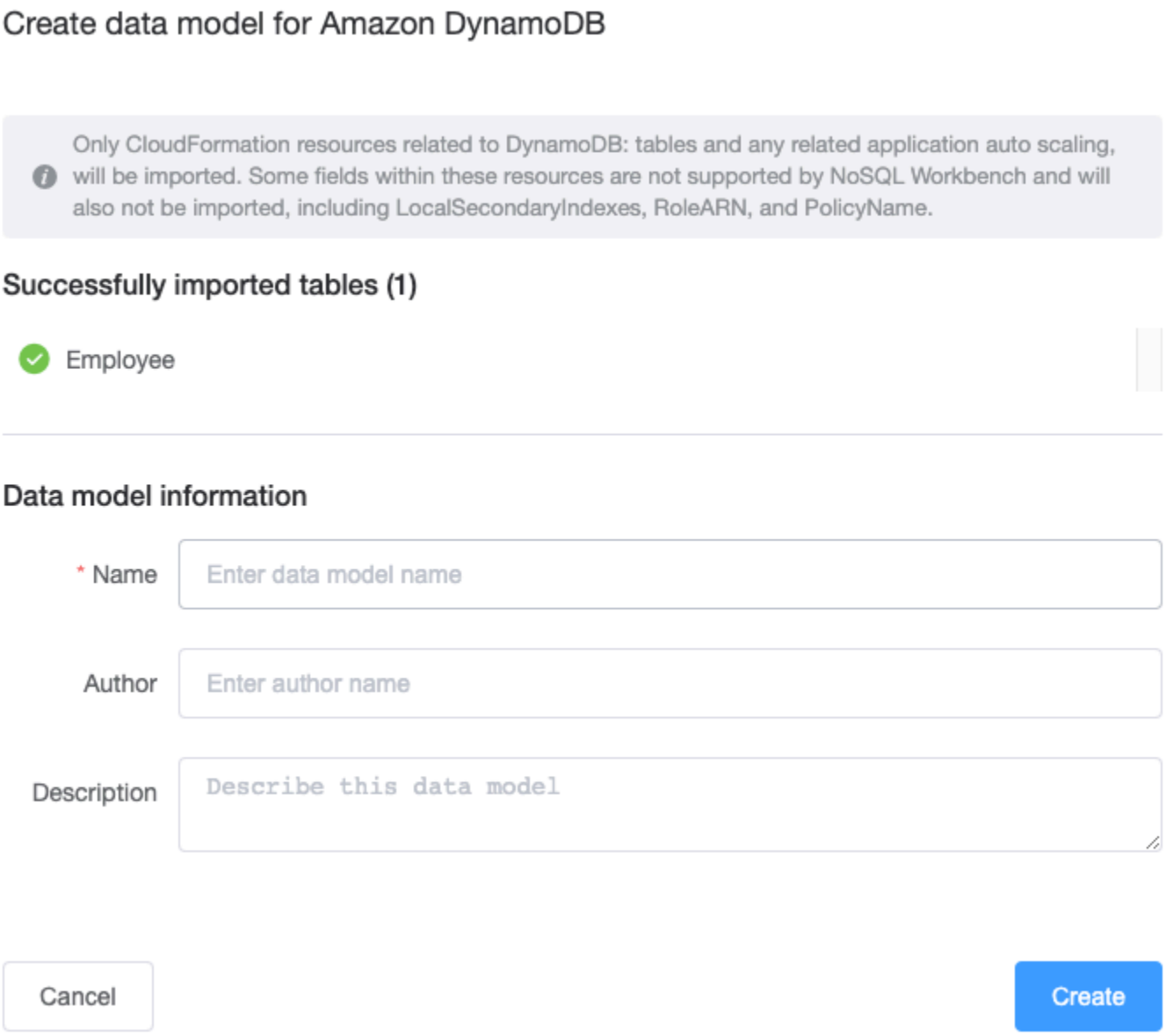
Na lista suspensa, escolha se o modelo que deseja importar está no formato de modelo NoSQL Workbench ou no formato de modelo CloudFormation JSON. Se você tiver um modelo de dados existente aberto no NoSQL Workbench, terá a opção de importar um modelo do CloudFormation para o modelo atual.



3. Escolha um modelo para importar.



4. Se o modelo que você está importando estiver no formato de modelo do CloudFormation, você verá uma lista de tabelas a serem importadas e terá a oportunidade de especificar um nome, autor e descrição do modelo de dados.

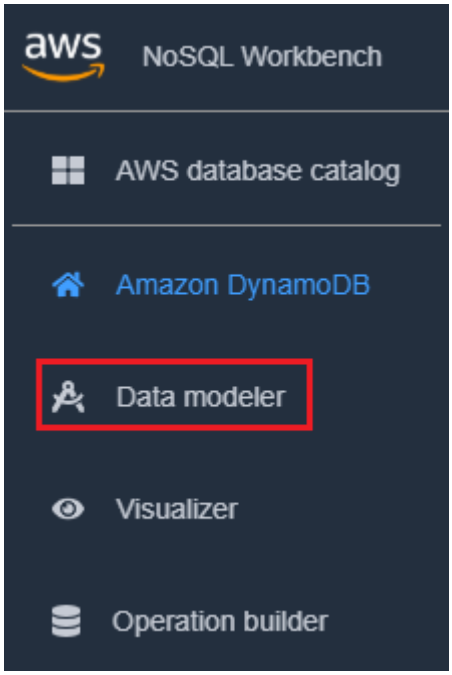


Exportar um modelo de dados

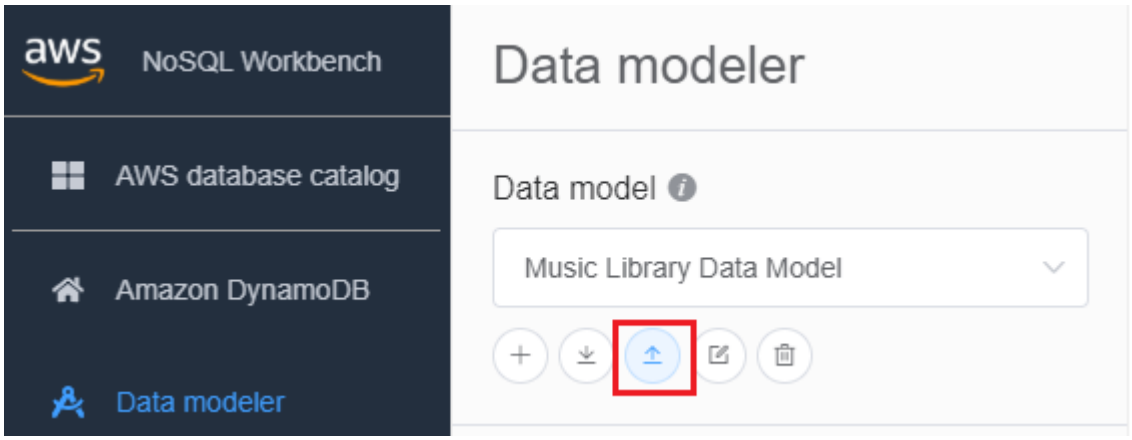
Depois de criar um modelo de dados usando o NoSQL Workbench para Amazon DynamoDB, você poderá salvar e exportar o modelo no formato de modelo NoSQL Workbench ou Formato de modelo JSON do AWS CloudFormation.

Para exportar um modelo de dados

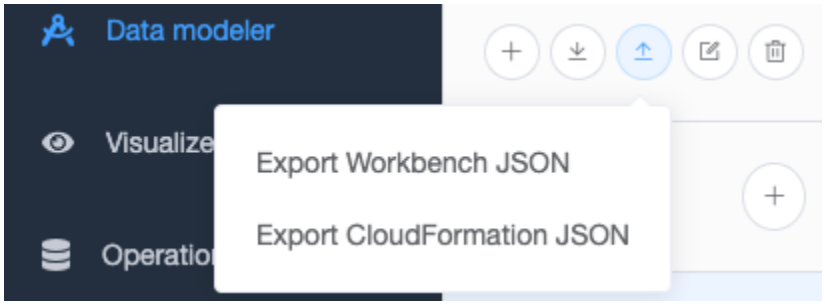
1. No NoSQL Workbench, no painel de navegação no lado esquerdo, escolha o ícone Data modeler (Modelador de dados).



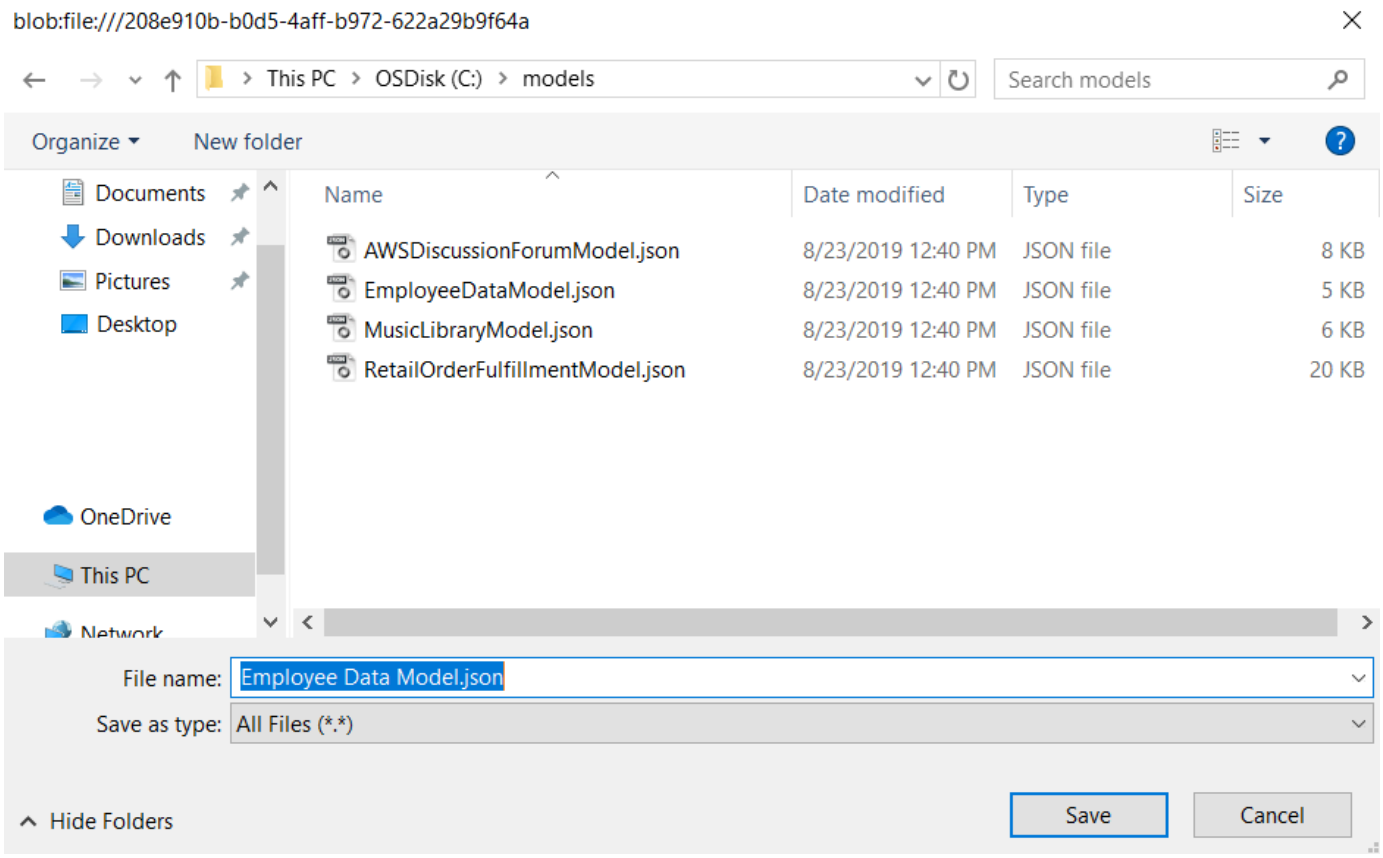
2. Passe o mouse sobre Export data model (Exportar modelo de dados).



Na lista suspensa, escolha se deseja importar seu modelo de dados no formato de modelo NoSQL Workbench ou no formato de modelo CloudFormation JSON.



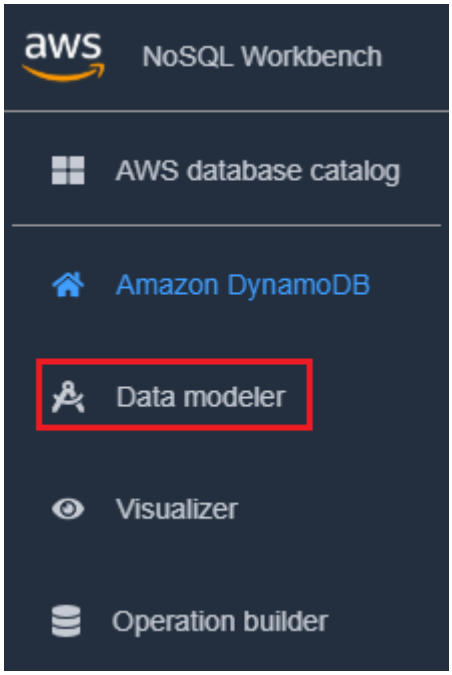
3. Escolha um local para salvar seu modelo.



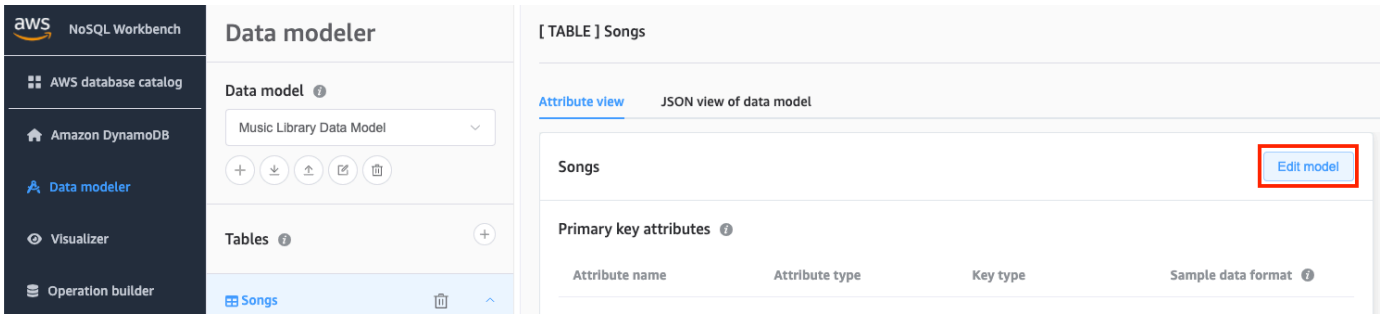
Editar um modelo de dados existente

Como editar um usuário existente

1. No NoSQL Workbench, no painel de navegação no lado esquerdo, escolha o botão Data modeler (Modelador de dados).



2. Selecione o modelo de dados e escolha a tabela que você deseja editar. Escolha Editar modelo.



3. Faça as edições necessárias e escolha Save edits (Salvar edições).

Como editar manualmente um modelo existente e adicionar uma faceta

1. Exporte seu modelo. Para ter mais informações, consulte Exportar um modelo de dados.
2. Abra o arquivo exportado em um editor.
3. Localize o objeto `DataModel` da tabela para a qual deseja criar uma faceta.

   Adicione uma matriz `TableFacets` representando todas as facetas da tabela.

   Para cada faceta, adicione um objeto à matriz `TableFacets`. Cada elemento da matriz tem as seguintes propriedades:

   - `FacetName`: um nome para sua faceta. Esse valor deve ser exclusivo em todo o modelo.

- `PartitionKeyAlias`: um nome amigável para a chave de partição da tabela. Esse alias é exibido quando você visualiza a faceta no NoSQL Workbench.
- `SortKeyAlias`: um nome amigável para a chave de classificação da tabela. Esse alias é exibido quando você visualiza a faceta no NoSQL Workbench. Essa propriedade não será necessária se a tabela não tiver nenhuma chave de classificação definida.
- `NonKeyAttributes`: uma matriz de nomes de atributos que são necessários para o padrão de acesso. Esses nomes devem ser mapeados para os nomes de atributo definidos para a tabela.

```
{
  "ModelName": "Music Library Data Model",
  "DataModel": [
    {
      "TableName": "Songs",
      "KeyAttributes": {
        "PartitionKey": {
          "AttributeName": "Id",
          "AttributeType": "S"
        },
        "SortKey": {
          "AttributeName": "Metadata",
          "AttributeType": "S"
        }
      },
      "NonKeyAttributes": [
        {
          "AttributeName": "DownloadMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloadsInMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Title",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Artist",
          "AttributeType": "S"
```

```
        },
        {
          "AttributeName": "TotalDownloads",
          "AttributeType": "S"
        },
        {
          "AttributeName": "DownloadTimestamp",
          "AttributeType": "S"
        }
      ],
      "TableFacets": [
        {
          "FacetName": "SongDetails",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "Title",
            "Artist",
            "TotalDownloads"
          ]
        },
        {
          "FacetName": "Downloads",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "DownloadTimestamp"
          ]
        }
      ]
    }
  ]
}
```

4. Agora você pode importar o modelo modificado para o NoSQL Workbench. Para ter mais informações, consulte Importar um modelo de dados existente.

Visualizar padrões de acesso a dados

Você pode usar a ferramenta de visualização no NoSQL Workbench para Amazon DynamoDB para mapear consultas e visualizar diferentes padrões de acesso (conhecidos como facetas) de uma aplicação. Cada faceta corresponde a um padrão de acesso diferente no DynamoDB. Também é possível adicionar dados manualmente ao seu modelo de dados ou importar dados do MySQL.

Tópicos

- Adicionar dados de exemplo a um modelo de dados
- Importar dados de amostra de um arquivo CSV
- Exibir padrões de acesso a dados
- Visualizar todas as tabelas de um modelo de dados usando visualização
- Confirmação de um modelo de dados no DynamoDB

Adicionar dados de exemplo a um modelo de dados

Com a adição de dados de exemplo ao modelo, você pode exibir dados ao visualizar o modelo e seus vários padrões de acesso a dados ou facetas.

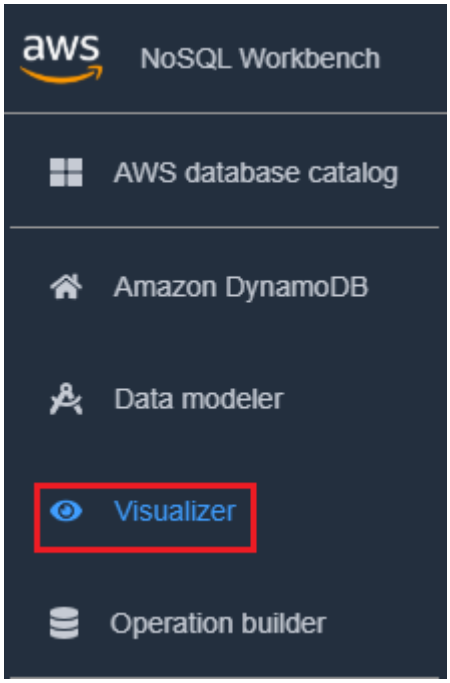
Há duas maneiras de adicionar dados de amostra. Uma delas é usar nossa ferramenta de geração automática de dados de amostra. A outra é adicionar dados um de cada vez.

Siga estas etapas para adicionar dados de exemplo a um modelo de dados usando o NoSQL Workbench para Amazon DynamoDB.

Como gerar dados de amostra automaticamente

A geração automática de dados de amostra ajuda você a gerar entre 1 e 5.000 linhas de dados para uso imediato. Você pode especificar um tipo de dados de amostra granular para criar dados realistas com base em suas necessidades de projeto e teste. Para utilizar a capacidade de gerar dados realistas, você precisa especificar o formato do tipo de dados de amostra para seus atributos no modelador de dados. Consulte Criar um novo modelo de dados para especificar exemplos de formatos de tipo de dados.

1. No painel de navegação no lado esquerdo, escolha o ícone de visualizer (visualizador).

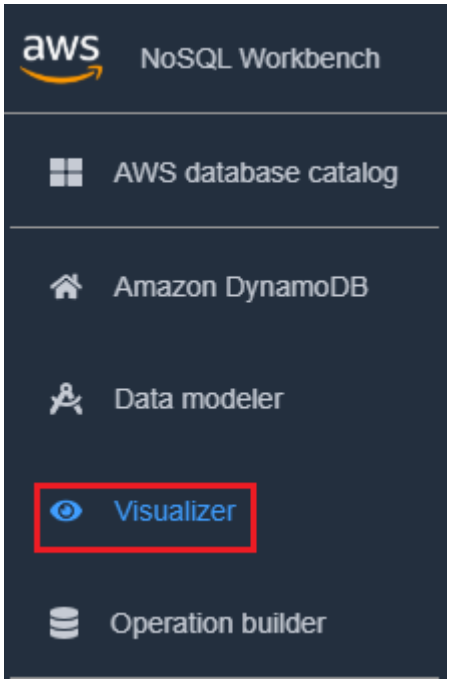


2. No visualizador, selecione o modelo de dados e escolha a tabela.
3. Escolha o menu suspenso Ação e selecione Adicionar dados de amostra.

4. Insira o número ou os itens de dados de amostra que você gostaria de gerar e selecione Confirmar.

Como adicionar dados de amostra individualmente

1. No painel de navegação no lado esquerdo, escolha o ícone de visualizer (visualizador).

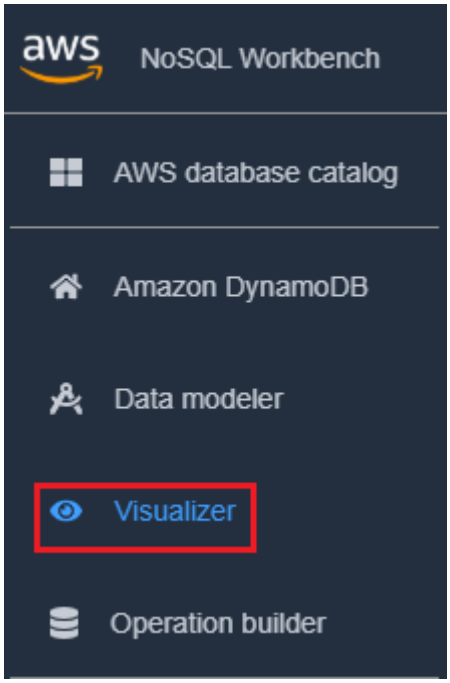


2. No visualizador, selecione o modelo de dados e escolha a tabela.
3. Escolha o menu suspenso Ação e selecione Editar dados.

4. Escolha Adicionar nova linha. Insira os dados de exemplo nas caixas de texto vazias e escolha Adicionar nova linha novamente para adicionar linhas extras. Quando concluir, escolha Salvar alterações.

Como excluir dados de amostra

1. No painel de navegação no lado esquerdo, escolha o ícone de visualizer (visualizador).



2. No visualizador, selecione o modelo de dados e escolha a tabela.
3. Escolha o menu suspenso Ação e selecione Editar dados.

4. Marque o ícone de exclusão ao lado de cada linha de dados que deseja excluir.

Importar dados de amostra de um arquivo CSV

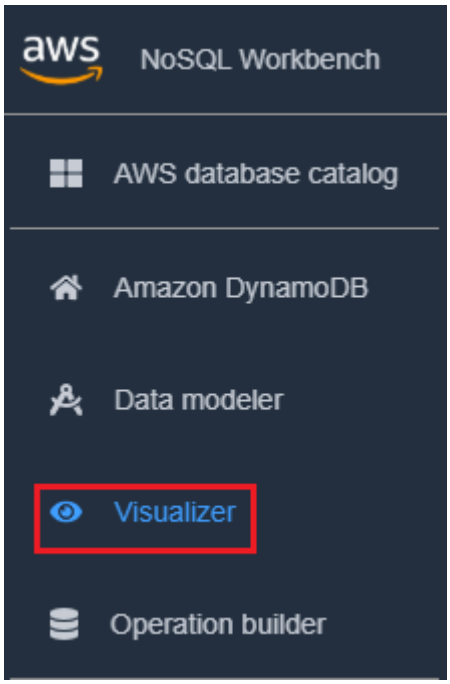
Se você tiver dados de amostra preexistentes em um arquivo CSV, poderá importá-los para o NoSQL Workbench. Isso permite que você preencha rapidamente seu modelo com dados de amostra sem ter que inseri-los linha a linha.

Os nomes de coluna no arquivo CSV devem corresponder aos nomes de atributo no modelo de dados, mas não precisam estar na mesma ordem. Por exemplo, se o modelo de dados tiver atributos chamados`LoginAlias`, `FirstName` e `LastName`, suas colunas CSV podem ser `LastName`, `FirstName` e `LoginAlias`.

A importação de dados de um arquivo CSV é limitada a 150 linhas por vez.

Importar dados de um arquivo CSV para o NoSQL Workbench

1. No painel de navegação no lado esquerdo, escolha o ícone de visualizer (visualizador).



2. No visualizador, selecione o modelo de dados e escolha a tabela.
3. Escolha o menu suspenso Ação e selecione Editar dados.
4. Escolha novamente o menu suspenso Ação e selecione Importar arquivo CSV.
5. Selecione o arquivo CSV e escolha Open (Abrir). Os dados no arquivo CSV serão anexados à sua tabela.

> **Note**
>
> Se o arquivo CSV contiver uma ou mais linhas com as mesmas chaves que os itens já na tabela, você terá a opção de substituir os itens existentes ou anexá-las ao final da tabela. Se você optar por anexar os itens, o sufixo “-Copy” será adicionado à chave de cada item duplicado para diferenciá-los dos itens que já estavam na tabela.

Exibir padrões de acesso a dados

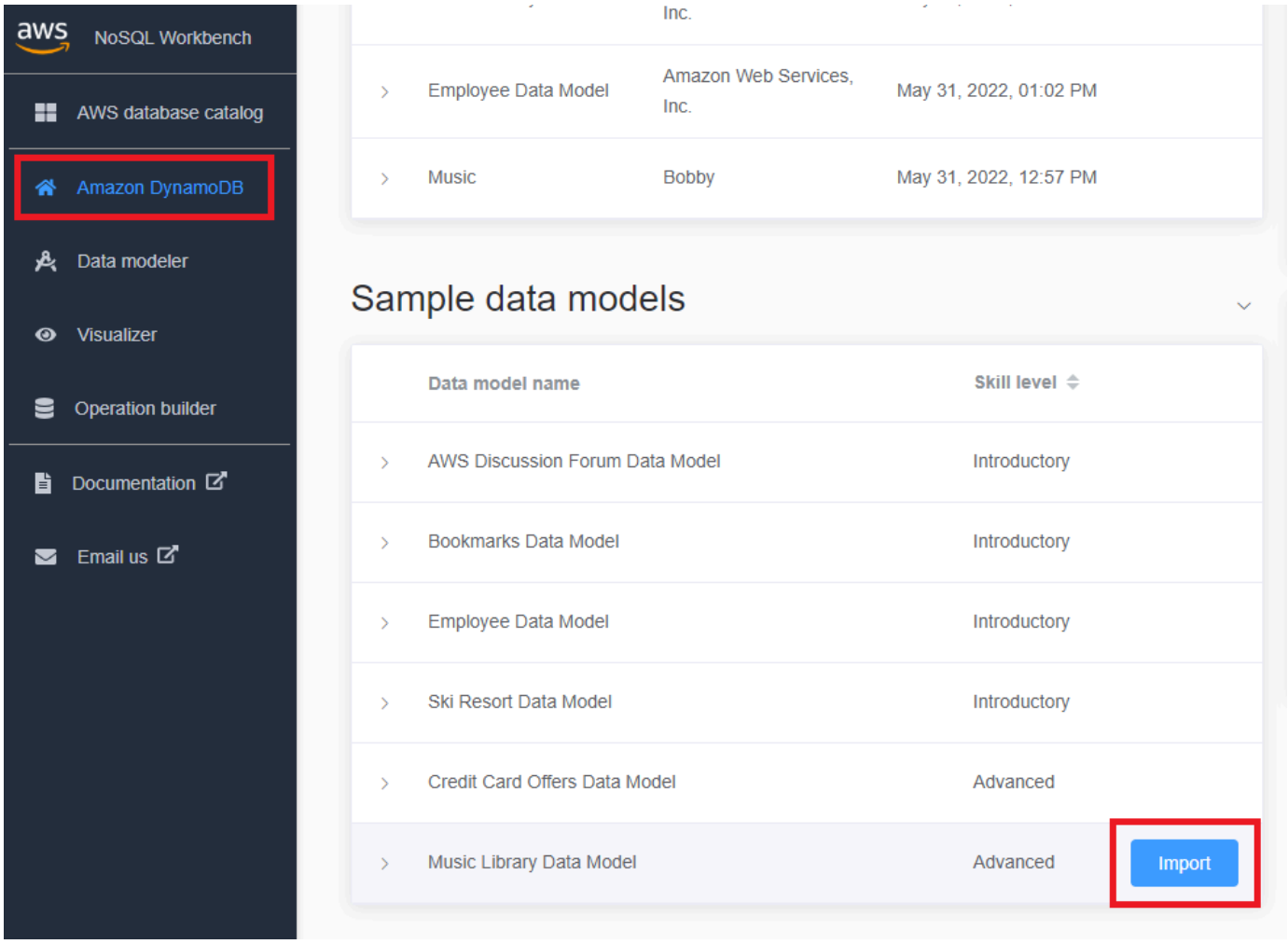
No NoSQL Workbench, as facetas representam os diferentes padrões de acesso a dados de uma aplicação para o Amazon DynamoDB. As facetas podem ajudar você a visualizar o modelo de dados quando vários tipos de dados são representados por uma chave de classificação. As facetas

oferecem uma maneira de visualizar um subconjunto dos dados em uma tabela, sem precisar ver os registros que não atendam às restrições da faceta. As facetas são consideradas uma ferramenta visual de modelagem de dados e não existem como uma estrutura utilizável no DynamoDB, pois são puramente uma ajuda para a modelagem de padrões de acesso.

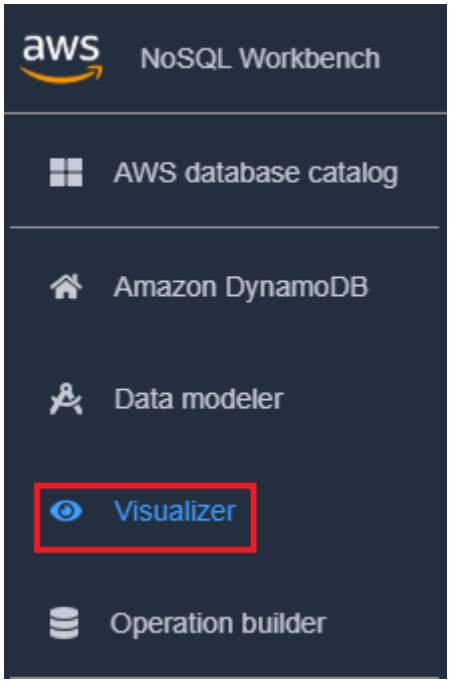
Para ver um exemplo de facetas, você pode importar um dos nossos modelos de dados de amostra com facetas como parte do modelo de dados.

Importar modelo de dados de exemplo

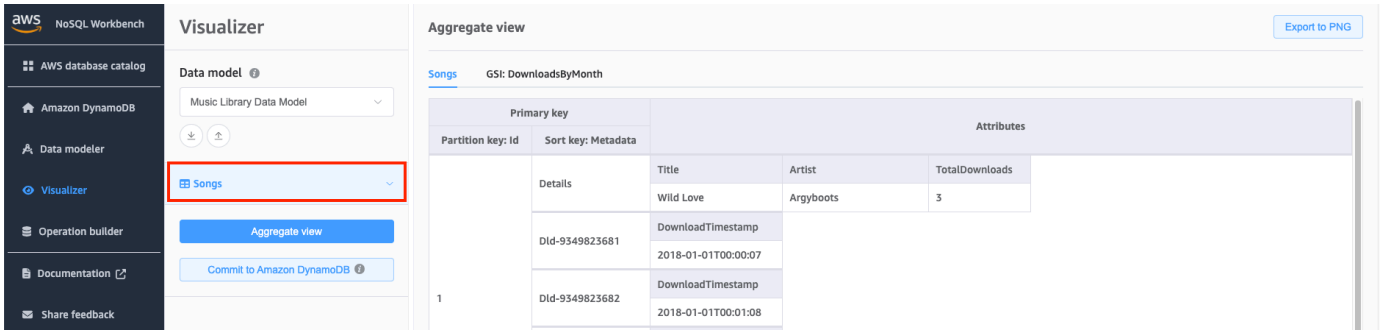
1. À esquerda, escolha Amazon DynamoDB.
2. Na seção Sample data models (Modelos de dados de exemplo), passe o mouse sobre Music Library Data Model (Modelo de dados da biblioteca de músicas) e escolha Import (Importar).



3. No painel de navegação no lado esquerdo, escolha o ícone de visualizer (visualizador).

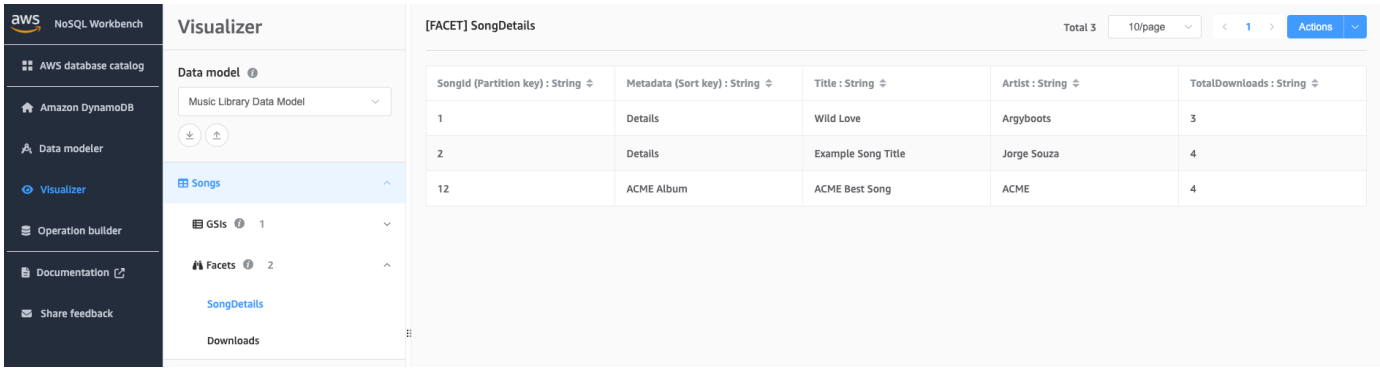


4. Escolha a tabela Songs (Músicas) para expandi-la. Será exibida uma visualização agregada de seus dados.



5. Selecione a seta suspensa de Facets (Facetas) para expandir as facetas disponíveis.

6. Escolha a faceta SongDetails para visualizar os dados com a faceta SongDetails aplicada.



Você também pode editar as definições de facetas usando o Modelador de dados. Para ter mais informações, consulte Editar um modelo de dados existente.

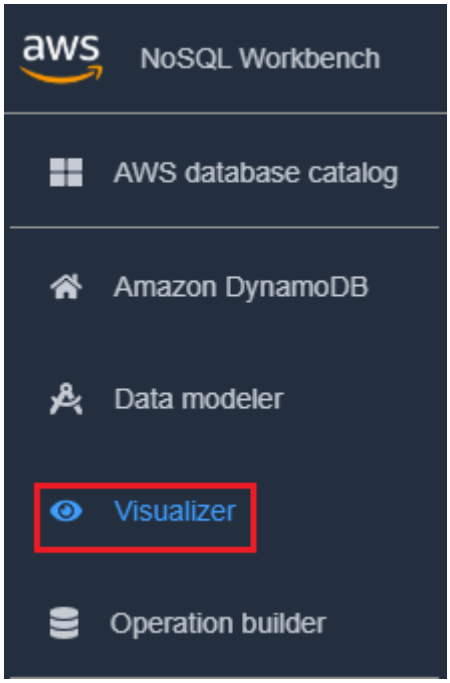
Visualizar todas as tabelas de um modelo de dados usando visualização

A exibição agregada no NoSQL Workbench para Amazon DynamoDB representa todas as tabelas de um modelo de dados. Para cada tabela, são exibidas as seguintes informações:

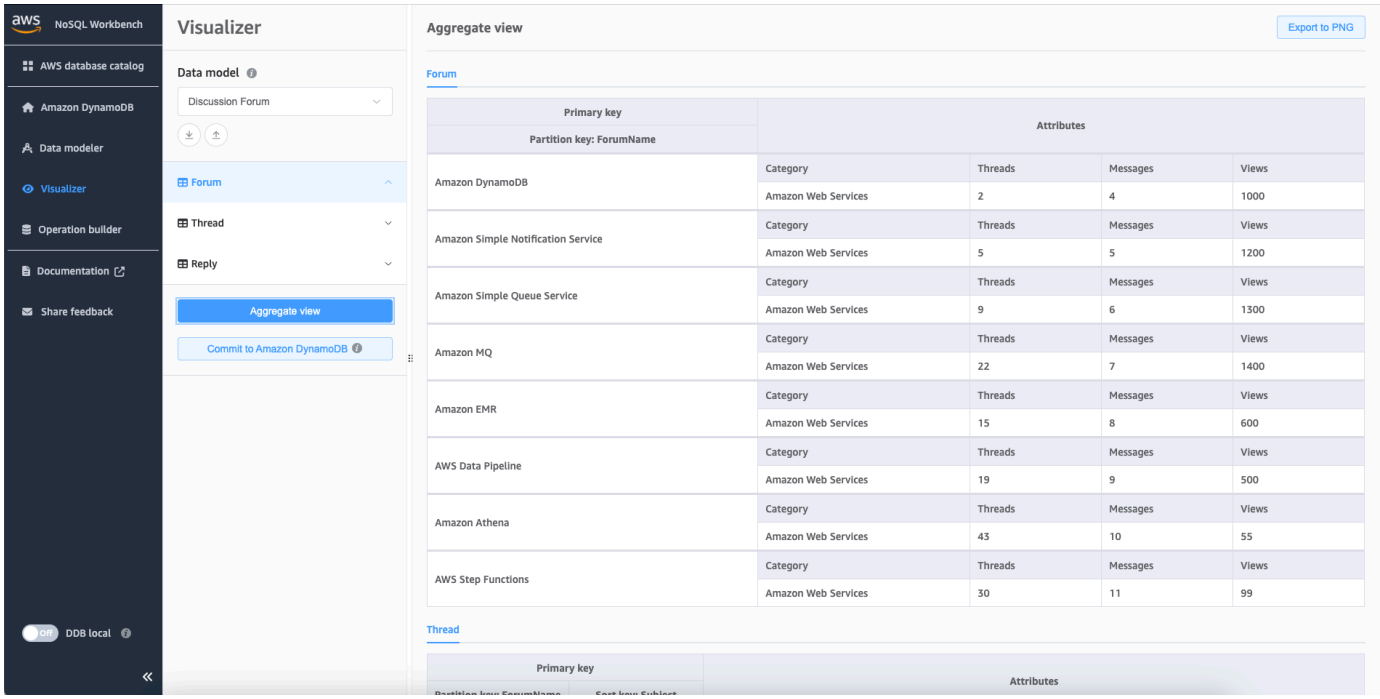
- Nomes de colunas da tabela
- Dados de exemplo
- Todos os índices secundários globais associados à tabela. As informações a seguir são exibidas para cada índice:
  - Nomes de colunas do índice
  - Dados de exemplo

Para visualizar todas as informações da tabela

1. No painel de navegação no lado esquerdo, escolha o ícone de visualizer (visualizador).



2. No visualizador, escolha Aggregate view (Visualização agregada).



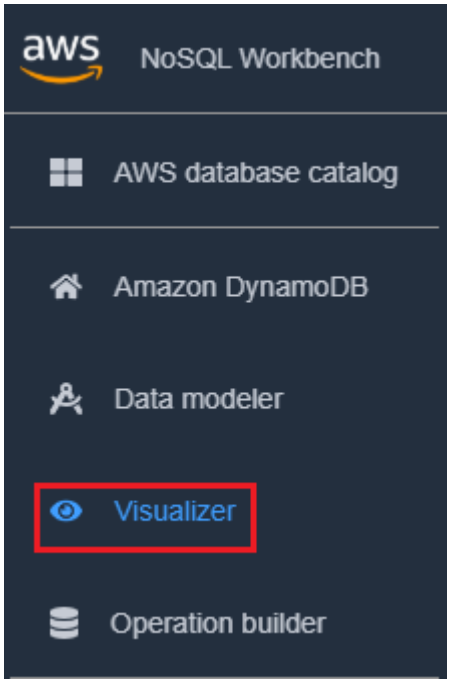
Confirmação de um modelo de dados no DynamoDB

Quando estiver satisfeito com seu modelo de dados, você poderá confirmá-lo no Amazon DynamoDB.

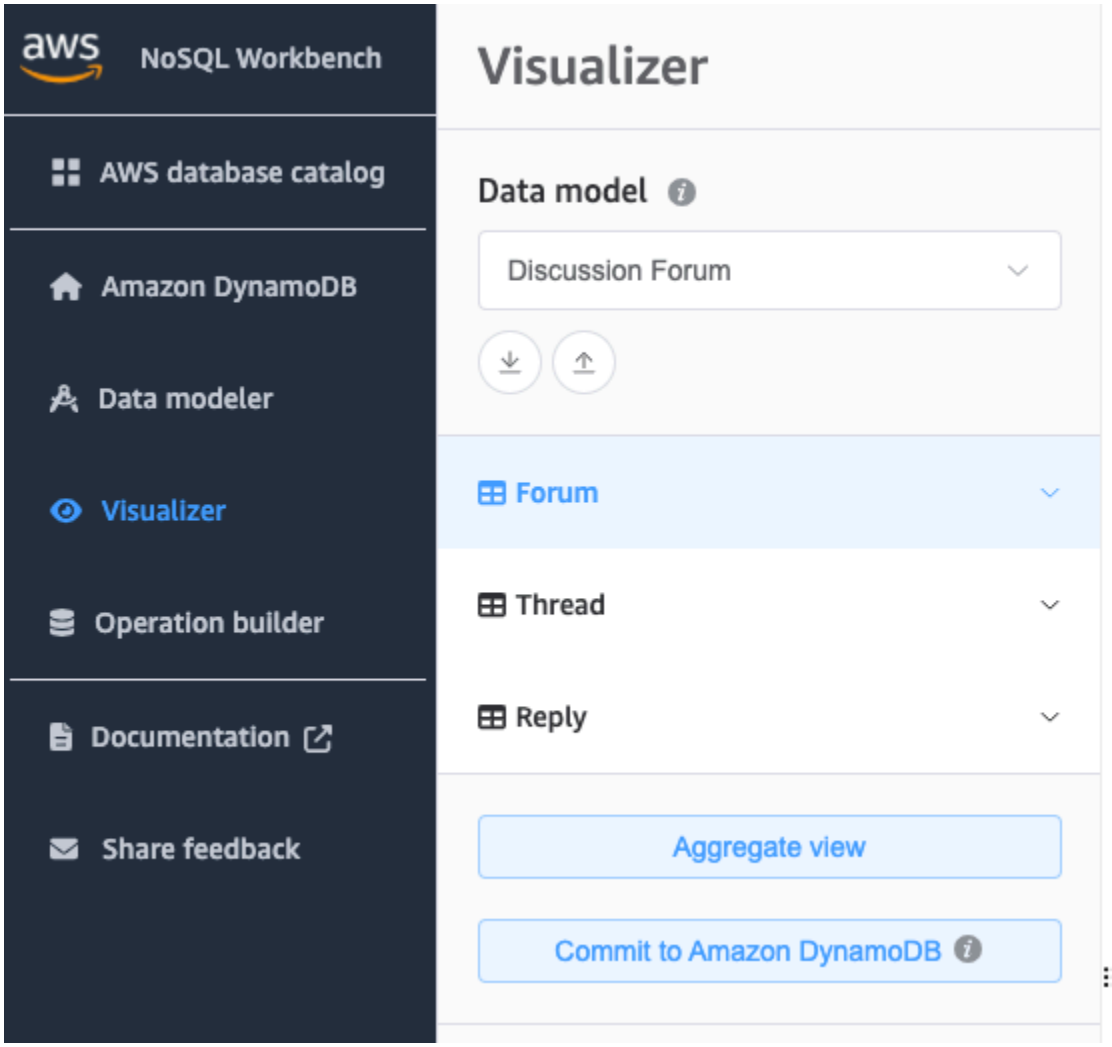
> **Note**
>
> - Essa ação resulta na criação de recursos no lado do servidor na AWS para as tabelas e os índices secundários globais representados no modelo de dados.
> - As tabelas são criadas com as seguintes características:
>   - O Auto scaling é definido em 70% de utilização de destino.
>   - A capacidade provisionada é definida em 5 unidades de capacidade de leitura e 5 unidades de capacidade de gravação.
> - Os índices secundários globais são criados com a capacidade provisionada de 10 unidades de capacidade de leitura e 5 unidades de capacidade de gravação.

Para confirmar o modelo de dados no DynamoDB

1. No painel de navegação no lado esquerdo, escolha o ícone de visualizer (visualizador).



2. Escolha Commit to DynamoDB (Confirmar ao DynamoDB).



3. Escolha uma conexão já existente ou crie uma conexão escolhendo a guia Add new connection remote (Adicionar nova conexão remota).

   - Para adicionar uma nova conexão, especifique as seguintes informações:
     - Account Alias (Alias da conta)
     - Região da AWS
     - Access key ID (ID da chave de acesso)
     - Secret access key (Chave de acesso secreta)

     Para obter mais informações sobre como obter as chaves de acesso, consulte Obter uma chave de acesso da AWS.
   - Opcionalmente, você pode especificar o seguinte:
     - Session token (Token da sessão)

- ARN do perfil do IAM
- Se não quiser se cadastrar em uma conta de nível gratuito e preferir usar o DynamoDB Local (versão disponível para download):
  1. Escolha a guia DynamoDB local connection (Adicionar uma nova conexão local do DynamoDB).
  2. Especifique o Connection name (Nome da conexão) e a Port (Porta).

4. Escolha Commit (Confirmar).

> **Note**
>
> Se instalou o DynamoDB local como parte da configuração do NoSQL Workbench, você deverá ativá-lo usando o botão Servidor do DynamoDB local no canto inferior esquerdo da tela do NoSQL Workbench. Consulte Instalar NoSQL Workbench para DynamoDB para obter mais informações sobre essa alternância.

Explorar conjuntos de dados e criar operações com o NoSQL Workbench

O NoSQL Workbench para Amazon DynamoDB oferece uma interface gráfica do usuário avançada para desenvolvimento e teste de consultas. Você pode usar o criador de operações no NoSQL Workbench para visualizar, explorar e consultar conjuntos de dados dinâmicos. O criador de operações estruturadas é compatível com expressão de projeção e expressão de condição, além de gerar código de amostra em vários idiomas. Você pode clonar tabelas diretamente de uma conta do Amazon DynamoDB para outra em diferentes regiões. Também é possível clonar tabelas diretamente entre o DynamoDB local e uma conta do Amazon DynamoDB para uma cópia mais rápida do esquema de chaves da tabela (e, opcionalmente, do esquema e dos itens do GSI) entre seus ambientes de desenvolvimento. Você pode salvar até 50 operações de dados do DynamoDB no criador de operações.

Tópicos

- Conectar-se a conjuntos de dados dinâmicos
- Criar operações complexas
- Clonar tabelas com NoSQL Workbench

- Exportar dados para um arquivo CSV

Conectar-se a conjuntos de dados dinâmicos

Para se conectar às suas tabelas do Amazon DynamoDB com o NoSQL Workbench, você deve primeiro se conectar à sua conta da AWS.

Para adicionar uma conexão ao seu banco de dados

1. No NoSQL Workbench, no painel de navegação no lado esquerdo, escolha o ícone Operation builder (Criador de operações).
2. Escolha Adicionar conexão.
3. Especifique as seguintes informações:

   - Connection name (Nome da conexão)
   - Região da AWS
   - Access key ID (ID da chave de acesso)
   - Secret access key (Chave de acesso secreta)

   Para obter mais informações sobre como obter as chaves de acesso, consulte Obter uma chave de acesso da AWS.

   Opcionalmente, você pode especificar o seguinte:

   - Session token (Token da sessão)
   - IAM role ARN (ARN do perfil do IAM)
4. Selecione Connect (Conectar).

   Se não quiser se cadastrar em uma conta de nível gratuito e preferir usar o DynamoDB Local (versão disponível para download):

   a. Escolha a guia Local na tela de conexão.
   b. Especifique as seguintes informações:

      - Connection name (Nome da conexão)
      - Port (Porta)

   c. Selecione o botão Connect (Conectar).

> **Note**
>
> Para se conectar ao DynamoDB local, inicialize manualmente o DynamoDB local usando o terminal (consulte Deploying DynamoDB local on your computer) ou inicialize o DynamoDB local diretamente usando a opção de DDB local no menu de navegação do NoSQL Workbench. Verifique se a porta de conexão é a mesma porta local do DynamoDB.

5. Na conexão criada, escolha Open (Abrir).

Após a conexão ao banco de dados do DynamoDB, a lista de tabelas disponíveis é exibida no painel esquerdo. Escolha uma das tabelas para retornar um exemplo dos dados armazenados na tabela.

Agora você pode executar consultas na tabela selecionada.

Para executar consultas em uma tabela, consulte a próxima seção sobre operações de criação. Consulte Criar operações complexas.

Criar operações complexas

O criador de operações no NoSQL Workbench para Amazon DynamoDB oferece uma interface visual na qual você pode executar operações de plano de dados complexas. Ele inclui suporte para expressões de projeção e de condição. Depois de criar uma operação, você pode salvá-la para uso posterior (até 50 operações podem ser salvas). Em seguida, você pode procurar uma lista de suas operações de plano de dados usadas com frequência no menu Saved Operations (Operações salvas) e usá-los para preencher e criar automaticamente uma nova operação. Você também pode optar por gerar código de exemplo para essas operações, em várias linguagens.

O NoSQL Workbench é compatível com a criação de instruções PartiQL para DynamoDB, o que permite interagir com o DynamoDB usando uma linguagem de consultas compatível com SQL. O NoSQL Workbench também é compatível com a criação de operações de API CRUD do DynamoDB.

Para usar o NoSQL Workbench para criar operações, no painel de navegação no lado esquerdo, escolha o ícone Operation builder (Criador de operações).

Tópicos

- Criar instruções em PartiQL

- Criar operações de API

Criar instruções em PartiQL

Para usar o NoSQL Workbench para criar declarações do PartiQL para DynamoDB, selecione Editor PartiQL no canto superior da interface de usuário do NoSQL Workbench.

É possível criar os tipos de instrução PartiQL a seguir no criador de operações.

Tópicos

- Instruções em Singleton
- Transações
- Lote

Instruções em Singleton

Para executar ou gerar código para uma instrução PartiQL, faça o seguinte:

1. Selecione Editor PartiQL próximo à parte superior da janela.
2. Insira uma PartiQL statement (Instrução PartiQL) válida.
3. Se a sua instrução usar parâmetros:
    a. Escolha Optional request parameters (Parâmetros de solicitação opcionais).
    b. Escolha Add new parameters (Adicionar novos parâmetros).
    c. Insira o tipo e o valor do atributo.
    d. Para adicionar parâmetros extras, repita as etapas b e c.
4. Se desejar gerar código, escolha Generate code (Gerar código).

    Selecione a linguagem desejada nas guias exibidas. Agora você pode copiar esse código e usá-lo na sua aplicação.
5. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).
6. Se desejar salvar esta operação para uso posterior, escolha Save operation (Salvar operação). Depois, insira um nome para sua operação e escolha Save (Salvar).

Transações

Para executar ou gerar código para uma transação do PartiQL, faça o seguinte:

1. Selecione PartiQLTransaction no menu suspenso Mais operações.
2. Escolha Add a new statement (Adicionar uma nova instrução).
3. Insira uma PartiQL statement (Instrução PartiQL) válida.

> **Note**
>
> Não há suporte a operações de leitura e gravação na mesma solicitação de transação PartiQL. Uma instrução SELECT não pode estar presente na mesma solicitação que instruções INSERT, UPDATE e DELETE. Consulte Executar transações com PartiQL para DynamoDB para obter mais detalhes.

4. Se a sua instrução usar parâmetros
   a. Escolha Optional request parameters (Parâmetros de solicitação opcionais).
   b. Escolha Add new parameters (Adicionar novos parâmetros).
   c. Insira o tipo e o valor do atributo.
   d. Para adicionar parâmetros extras, repita as etapas b e c.
5. Para adicionar mais instruções, repita as etapas 2 a 4.
6. Se desejar gerar código, escolha Generate code (Gerar código).

   Selecione a linguagem desejada nas guias exibidas. Agora você pode copiar esse código e usá-lo na sua aplicação.
7. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).
8. Se desejar salvar esta operação para uso posterior, escolha Save operation (Salvar operação). Depois, insira um nome para sua operação e escolha Save (Salvar).

Lote

Para executar ou gerar código para um lote de PartiQL, faça o seguinte:

1. Selecione PartiQLBatch no menu suspenso Mais operações.
2. Escolha Add a new statement (Adicionar uma nova instrução).

3. Insira uma PartiQL statement (Instrução PartiQL) válida.

> **Note**
>
> Não há suporte a operações de leitura e gravação na mesma solicitação de lote de PartiQL, o que significa que uma instrução SELECT não pode estar na mesma solicitação que instruções INSERT, UPDATE e DELETE. Operações de gravação no mesmo item não são permitidas. Assim como na operação BatchGetItem, apenas operações de leitura singleton são aceitas. Operações de verificação e consulta não são aceitas. Consulte Executar operações em lote com PartiQL para DynamoDB para obter mais detalhes.

4. Se a sua instrução usar parâmetros:
   a. Escolha Optional request parameters (Parâmetros de solicitação opcionais).
   b. Escolha Add new parameters (Adicionar novos parâmetros).
   c. Insira o tipo e o valor do atributo.
   d. Para adicionar parâmetros extras, repita as etapas b e c.
5. Para adicionar mais instruções, repita as etapas 2 a 4.
6. Se desejar gerar código, escolha Generate code (Gerar código).

   Selecione a linguagem desejada nas guias exibidas. Agora você pode copiar esse código e usá-lo na sua aplicação.
7. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).
8. Se desejar salvar esta operação para uso posterior, escolha Save operation (Salvar operação). Depois, insira um nome para sua operação e escolha Save (Salvar).

Criar operações de API

Para usar o NoSQL Workbench para criar APIs CRUD do DynamoDB, selecione Criador de operações à esquerda da interface de usuário do NoSQL Workbench.

Depois, selecione Abrir e escolha uma conexão.

Você pode executar as operações a seguir no criador de operações.

- Excluir tabela

- Criar tabela
- Atualizar tabela

- Colocar item
- Atualizar item
- Excluir item
- Query
- Scan
- Transação Obter itens
- Transação Gravar itens

Excluir tabela

Para executar uma operação `Delete Table`, faça o seguinte:

1. Encontre a tabela que você deseja excluir na seção Tabelas.
2. Selecione Excluir tabela no menu de reticências horizontais.
3. Confirme se você deseja excluir a tabela inserindo o Nome da tabela.
4. Selecione Excluir.

Para obter mais informações essa operação, consulte Delete table (Excluir tabela) na Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB).

Excluir o GSI

Para executar uma operação `Delete GSI`, faça o seguinte:

1. Encontre o GSI de uma tabela que você deseja excluir na seção Tabelas.
2. Selecione Excluir GSI no menu de reticências horizontal.
3. Confirme se você deseja excluir o GSI inserindo o Nome do GSI.
4. Selecione Excluir.

Para obter mais informações essa operação, consulte Delete table (Excluir tabela) na Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB).

Create table

Para executar uma operação `Create Table`, faça o seguinte:

1. Selecione o ícone + ao lado da seção Tabelas.
2. Insira o nome da tabela desejado.
3. Crie uma chave de partição.
4. Opcional: crie uma chave de classificação.
5. Para personalizar as configurações de capacidade, desmarque a caixa ao lado de Usar configurações de capacidade padrão.
   - Agora é possível selecionar Provisioned (Provisionada) ou On-demand capacity (Capacidade sob demanda).

     Com a opção Provisionada selecionada, você pode definir unidades de capacidade mínima e máxima de leitura e gravação. Você também pode habilitar ou desabilitar a escalabilidade automática.
   - Se, no momento, a tabela estiver definida como sob demanda, você não poderá especificar um throughput provisionado.
   - Se você mudar de sob demanda para throughput provisionado, o ajuste de escala automático será aplicado automaticamente a todos os GSIs com: mín.: 1, máx.: 10; meta: 70%.
6. Selecione Ignorar GSIs e criar para criar essa tabela sem um GSI. Você também pode selecionar Próximo para criar um GSI com essa nova tabela.

Para obter mais informações sobre essa operação, consulte Create table (Criar tabela) na Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB).

Criar o GSI

Para executar uma operação `Create GSI`, faça o seguinte:

1. Encontre uma tabela à qual você deseja adicionar um GSI.
2. No menu de reticências horizontal, selecione Criar GSI.
3. Nomeie o GSI em Nome do índice.
4. Crie uma chave de partição.
5. Opcional: crie uma chave de classificação.

6. Selecione uma opção de tipo de projeção no menu suspenso.
7. Selecione Criar GSI.

Para obter mais informações sobre essa operação, consulte Create table (Criar tabela) na Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB).

Atualizar tabela

Para atualizar as configurações de capacidade de uma tabela com uma operação `Update Table`, faça o seguinte:

1. Encontre a tabela para a qual você deseja atualizar as configurações de capacidade.
2. No menu de reticências horizontal, selecione Atualizar configurações de capacidade.
3. Selecione Provisionada ou Capacidade sob demanda.

   Com a opção Provisionada selecionada, você pode definir unidades de capacidade mínima e máxima de leitura e gravação. Você também pode habilitar ou desabilitar a escalabilidade automática.
4. Selecione Atualizar.

Para obter mais informações sobre essa operação, consulte Atualizar tabela na Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB).

Atualizar o GSI

Para atualizar as configurações de capacidade de um GSI com uma operação `Update Table`, faça o seguinte:

> **Note**
>
> Por padrão, os índices secundários globais herdam as configurações de capacidade da tabela de base. Os índices secundários globais podem ter um modo de capacidade diferente somente quando a tabela básica está no modo de capacidade provisionada. Ao criar um índice secundário global em uma tabela de modo provisionado, você deve especificar unidades de capacidade de leitura e gravação para a workload esperada no índice. Para obter mais informações, consulte Considerações sobre throughput provisionado para índices secundários globais.

1. Encontre o GSI cujas configurações de capacidade você deseja atualizar.
2. No menu de reticências horizontal, selecione Atualizar configurações de capacidade.
3. Agora é possível selecionar Provisioned (Provisionada) ou On-demand capacity (Capacidade sob demanda).

   Com a opção Provisionada selecionada, você pode definir unidades de capacidade mínima e máxima de leitura e gravação. Você também pode habilitar ou desabilitar a escalabilidade automática.
4. Selecione Atualizar.

Para obter mais informações sobre essa operação, consulte Atualizar tabela na Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB).

Colocar item

Você pode criar um item usando a operação `Put Item`. Para executar ou gerar código para uma operação `Put Item`, faça o seguinte:

1. Encontre a tabela na qual você deseja criar um item.
2. No menu suspenso Ações, selecione Criar item.
3. Insira o valor da chave de partição.
4. Insira o valor da chave de classificação, se houver.
5. Se desejar adicionar atributos que não sejam de chave, faça o seguinte:

   a. Selecione + Adicionar outros atributos.

   b. Especifique o Attribute name (Nome do atributo), o Type (Tipo) e o Value (Valor).
6. Se for necessário cumprir a expressão de condição para que a operação `Put Item` tenha êxito, faça o seguinte:

   a. Escolha Condition (Condição).

   b. Especifique o nome do atributo, o operador de comparação, o tipo de atributo e o valor do atributo.

   c. Se outras condições forem necessárias, escolha Condition (Condição) novamente.

   Para obter mais informações, consulte Exemplo de expressão de condição do DynamoDB na CLI.

7. Se desejar gerar código, escolha Generate code (Gerar código).

   Selecione a linguagem desejada nas guias exibidas. Agora você pode copiar esse código e usá-lo na sua aplicação.
8. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).
9. Se desejar salvar esta operação para uso posterior, escolha Save operation (Salvar operação). Em seguida, insira um nome para a operação e clique em Save (Salvar).

Para obter mais informações sobre essa operação, consulte PutItem na Referência da API do Amazon DynamoDB.

Atualizar item

Para executar ou gerar código para uma operação `Update Item`, faça o seguinte:

1. Encontre a tabela na qual você deseja atualizar um item.
2. Selecione o item.
3. Insira o nome e o valor do atributo para a expressão selecionada.
4. Se desejar adicionar mais expressões, escolha outra expressão na lista suspensa Atualizar expressão e selecione o ícone +.
5. Se for necessário cumprir a expressão de condição para que a operação `Update Item` tenha êxito, faça o seguinte:

   a. Escolha Condition (Condição).

   b. Especifique o nome do atributo, o operador de comparação, o tipo de atributo e o valor do atributo.

   c. Se outras condições forem necessárias, escolha Condition (Condição) novamente.

   Para obter mais informações, consulte Exemplo de expressão de condição do DynamoDB na CLI.
6. Se desejar gerar código, escolha Generate code (Gerar código).

   Escolha a guia da linguagem desejada. Agora você pode copiar esse código e usá-lo na sua aplicação.
7. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).

8. Se desejar salvar esta operação para uso posterior, escolha Save operation (Salvar operação). Em seguida, insira um nome para a operação e clique em Save (Salvar).

Para obter mais informações sobre essa operação, consulte UpdateItem na Referência da API do Amazon DynamoDB.

Excluir item

Para executar uma operação `Delete Item`, faça o seguinte:

1. Encontre a tabela na qual você deseja excluir um item.
2. Selecione o item.
3. No menu suspenso Ações, selecione Excluir item.
4. Confirme se você deseja excluir o item selecionando Excluir.

Para obter mais informações sobre essa operação, consulte DeleteItem na Referência da API do Amazon DynamoDB.

Duplicar item

É possível duplicar um item criando um item com os mesmos atributos. Para duplicar um item, faça o seguinte:

1. Encontre a tabela na qual você deseja duplicar um item.
2. Selecione o item.
3. No menu suspenso Ações, selecione Duplicar item.
4. Especifique uma nova chave de partição.
5. Especifique uma nova chave de classificação (se necessário).
6. Selecione Executar.

Para obter mais informações sobre essa operação, consulte DeleteItem na Referência da API do Amazon DynamoDB.

Consulta

Para executar ou gerar código para uma operação `Query`, faça o seguinte:

1. Selecione Consultar na parte superior da interface de usuário do NoSQL Workbench.
2. Especifique o valor da chave de partição.
3. Se uma chave de classificação for necessária para a operação `Query`:
   a. Selecione Sort key (Chave de classificação).
   b. Especifique o operador de comparação e o valor do atributo.
4. Selecione Consultar para realizar essa operação de consulta. Se forem necessárias mais opções, marque a caixa de seleção Mais opções e continue com as etapas a seguir.
5. Se nem todos os atributos precisarem ser retornados com o resultado da operação, selecione Projection expression (Expressão de projeção).
6. Escolha o ícone +.
7. Insira o atributo a ser retornado com o resultado da consulta.
8. Se mais atributos forem necessários, escolha o +.
9. Se for necessário cumprir a expressão de condição para que a operação `Query` tenha êxito, faça o seguinte:
   a. Escolha Condition (Condição).
   b. Especifique o nome do atributo, o operador de comparação, o tipo de atributo e o valor do atributo.
   c. Se outras condições forem necessárias, escolha Condition (Condição) novamente.

   Para obter mais informações, consulte Exemplo de expressão de condição do DynamoDB na CLI.
10. Se desejar gerar código, escolha Generate code (Gerar código).

    Escolha a guia da linguagem desejada. Agora você pode copiar esse código e usá-lo na sua aplicação.
11. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).
12. Se desejar salvar esta operação para uso posterior, escolha Save operation (Salvar operação). Em seguida, insira um nome para a operação e clique em Save (Salvar).

Para obter mais informações sobre essa operação, consulte Query na Referência da API do Amazon DynamoDB.

Verificar

Para executar ou gerar código para uma operação `Scan`, faça o seguinte:

1. Selecione Verificar na parte superior da interface de usuário do NoSQL Workbench.
2. Selecione o botão Verificar para realizar essa operação básica de verificação. Se forem necessárias mais opções, marque a caixa de seleção Mais opções e continue com as etapas a seguir.
3. Especifique um nome de atributo para filtrar os resultados da verificação.
4. Se nem todos os atributos precisarem ser retornados com o resultado da operação, selecione Projection expression (Expressão de projeção).
5. Se a expressão de condição precisar ser satisfeita para a operação scan ser bem-sucedida, faça o seguinte:

   a. Escolha Condition (Condição).
   b. Especifique o nome do atributo, o operador de comparação, o tipo de atributo e o valor do atributo.
   c. Se outras condições forem necessárias, escolha Condition (Condição) novamente.

   Para obter mais informações, consulte Exemplo de expressão de condição do DynamoDB na CLI.
6. Se desejar gerar código, escolha Generate code (Gerar código).

   Escolha a guia da linguagem desejada. Agora você pode copiar esse código e usá-lo na sua aplicação.
7. Se desejar que a operação seja executada imediatamente, escolha Run (Executar).
8. Se desejar salvar esta operação para uso posterior, escolha Save operation (Salvar operação). Em seguida, insira um nome para a operação e clique em Save (Salvar).

TransactGetItems

Para executar ou gerar código para uma operação `TransactGetItems`, faça o seguinte:

1. No menu suspenso Mais operações na parte superior da interface de usuário do NoSQL Workbench, selecione TransactGetItems.
2. Selecione o ícone + ao lado de TransactGetItem.

3. Especifique uma chave de partição.
4. Especifique uma chave de classificação (se necessário).
5. Selecione Executar para realizar a operação, Salvar operação para salvá-la ou Gerar código para gerar código para ela.

Para obter mais informações, consulte Amazon DynamoDB Transactions.

TransactWriteItems

Para executar ou gerar código para uma operação `TransactWriteItems`, faça o seguinte:

1. No menu suspenso Mais operações na parte superior da interface de usuário do NoSQL Workbench, selecione TransactWriteItems.
2. Selecione uma operação no menu suspenso Ações.
3. Selecione o ícone + ao lado de TransactWriteItem.
4. Na lista suspensa Ações, selecione a operação desejada.

   - Para `DeleteItem`, siga as instruções para a operação Excluir item.
   - Para `PutItem`, siga as instruções para a operação Colocar item.
   - Para `UpdateItem`, siga as instruções para a operação Atualizar item.

   Para alterar a ordem das ações, escolha uma ação na lista à esquerda e selecione as setas para cima ou para baixo para movê-la para cima ou para baixo na lista.

   Para excluir uma ação, escolha a ação na lista e selecione o ícone Delete (Excluir) (lixeira).
5. Selecione Executar para realizar a operação, Salvar operação para salvá-la ou Gerar código para gerar código para ela.

Para obter mais informações, consulte Amazon DynamoDB Transactions.

Clonar tabelas com NoSQL Workbench

A clonagem de tabelas copiará o esquema de chaves de uma tabela (e, opcionalmente, o esquema e os itens do GSI) entre seus ambientes de desenvolvimento. Você pode clonar uma tabela entre o DynamoDB local e uma conta do Amazon DynamoDB, e até mesmo clonar uma tabela de uma conta para outra em diferentes regiões a fim de acelerar a experimentação.

Como clonar uma tabela

1. Em Criador de operações, selecione sua conexão e região (a seleção da região não está disponível para o DynamoDB local).
2. Depois de se conectar ao DynamoDB, navegue pelas tabelas e selecione a que deseja clonar.
3. No menu de reticências horizontal, selecione a opção Clonar.
4. Insira os detalhes de destino do clone:

    a. Selecione uma conexão.

    b. Selecione uma região (a seleção da região não está disponível para o DynamoDB local).

    c. Insira um novo nome de tabela.

    d. Selecione uma opção de clonagem:

        i. A opção Esquema de chaves é selecionada por padrão e não pode ser desmarcada. Por padrão, a clonagem de uma tabela copiará a chave primária e a chave de classificação, se estiverem disponíveis.

        ii. A opção Esquema do GSI será selecionada por padrão se a tabela a ser clonada tiver um GSI. A clonagem de uma tabela copiará a chave primária e a chave de classificação do GSI, se estiverem disponíveis. Você tem a opção de desmarcar a opção Esquema do GSI para pular a clonagem do esquema do GSI. A clonagem de uma tabela copiará as configurações de capacidade da tabela base como as configurações de capacidade do GSI. Você pode usar a operação `UpdateTable` no criador de operações para atualizar a configuração de capacidade do GSI da tabela após a conclusão da clonagem.
5. Insira o número de itens a serem clonados. Para clonar somente o esquema de chaves e, opcionalmente, o esquema do GSI, você pode manter o valor de Itens a serem clonados em 0. O número máximo de itens que podem ser clonados é 5.000.
6. Escolha um modo de capacidade:

    a. A opção Modo sob demanda está selecionada por padrão. O DynamoDB sob demanda oferece definição de preço “pagamento por solicitação” para solicitações de leitura e gravação, para que você pague apenas pelo o que usar. Para saber mais, consulte DynamoDB On-demand mode.

    b. A opção Modo provisionado permite especificar o número de leituras e gravações por segundo que você precisa para a aplicação. Você pode usar Auto Scaling para ajustar a

capacidade provisionada da tabela automaticamente em resposta às alterações de tráfego. Para saber mais, consulte DynamoDB Provisioned mode.

7. Selecione Clonar para começar a clonagem.
8. O processo de clonagem será executado em segundo plano. A guia Criador de operações mostrará uma notificação quando houver uma alteração no status da tabela de clonagem. Você pode acessar esse status selecionando a guia Criador de operações e o botão de seta. O botão de seta está localizado no widget de status da tabela de clonagem, localizado na parte inferior da barra lateral de menu.

Exportar dados para um arquivo CSV

Você pode exportar os resultados de uma consulta do Operation Builder para um arquivo CSV. Isso permite que você carregue os dados em uma planilha ou os processe usando a linguagem de programação que preferir.

Exportar para CSV

1. No Operation Builder, execute uma operação à sua escolha, como uma varredura ou uma consulta.

> **Note**
>
> - Só é possível exportar os resultados de operações da API de leitura e instruções PartiQL para um arquivo CSV. Não é possível exportar os resultados de instruções de leitura de transações.
> - No momento, você pode exportar uma página de resultados por vez para um arquivo CSV. Se houver várias páginas de resultados, você deverá exportar cada página individualmente.

2. Selecione os itens que você deseja exportar dos resultados.
3. No menu suspenso Ações, selecione Exportar como CSV.
4. Escolha um nome de arquivo e um local para o arquivo CSV e selecione Save (Salvar).

Modelos de dados de amostra para o NoSQL Workbench

A página inicial do modelador e visualizador exibe uma série de exemplos de modelos fornecidos com o NoSQL Workbench. Esta seção descreve esses modelos e seus usos potenciais.

Tópicos

- Modelo de dados de funcionários
- Modelo de dados de fórum de discussão
- Modelo de dados de biblioteca de música
- Modelo de dados de estação de esqui
- Modelo de dados de ofertas de cartão de crédito
- Modelo de dados de favoritos

Modelo de dados de funcionários

Este modelo de dados é um modelo introdutório. Ele representa os detalhes básicos de um funcionário, como um alias exclusivo, nome, sobrenome, designação, gerente e habilidades.

Este modelo de dados descreve algumas técnicas, como lidar com atributos complexos, por exemplo, ter mais de uma habilidade. Esse modelo também é um exemplo de relação de um para muitos entre o gerente e seus funcionários subordinados, alcançado pelo índice secundário DirectReports.

Os padrões de acesso facilitados por este modelo de dados são:

- Recuperação de um registro de funcionário usando o alias de login do funcionário, facilitada por uma tabela chamada `Employee`.
- Pesquisa de funcionários por nome, facilitada pelo índice secundário global da tabela Employee (Funcionários) chamado `Name`.
- Recuperação de todos os relatórios diretos de um gerente usando o alias de login do gerente, facilitada pelo índice secundário global da tabela Employee (Funcionários) chamado `DirectReports`.

Modelo de dados de fórum de discussão

Este modelo de dados representa fóruns de discussão. Usando este modelo, os clientes podem se envolver com a comunidade de desenvolvedores, fazer perguntas e responder às postagens de outros clientes. Cada serviço da AWS tem um fórum dedicado. Qualquer um pode iniciar um novo tópico de discussão postando uma mensagem em um fórum, e cada assunto de tópico recebe um número de respostas.

Os padrões de acesso facilitados por este modelo de dados são:

- Recuperação de um registro de fórum usando o nome do fórum, facilitada por uma tabela chamada `Forum`.
- Recuperação de um thread específico ou de todos os threads de um fórum, facilitada por uma tabela chamada `Thread`.
- Pesquisa de respostas usando o endereço de e-mail do usuário de publicação, facilitada pelo índice secundário global da tabela Reply (Responder) chamado `PostedBy-Message-Index`.

Modelo de dados de biblioteca de música

Este modelo de dados representa uma biblioteca de músicas que tem uma grande coleção de músicas e mostra as mais baixadas quase em tempo real.

Os padrões de acesso facilitados por este modelo de dados são:

- Recuperação de um registro de música, facilitada por uma tabela chamada `Songs`.
- Recuperação de um registro de download específico ou de todos os registros de download de uma música, facilitada por uma tabela chamada `Songs`.
- Recuperação de um registro de número de downloads mensais ou de todos os registros de download mensais de uma música, facilitada por uma tabela chamada `Song`.
- Recuperação de todos os registros (incluindo gravação de músicas, registros de download e registros de contagem de downloads mensais) de uma música, facilitada por uma tabela chamada `Songs`.
- Pesquisa das músicas mais baixadas, facilitada pelo índice secundário global da tabela Músicas chamado `DownloadsByMonth`.

Modelo de dados de estação de esqui

Este modelo de dados representa uma estância de esqui que tem uma grande coleção de dados para cada subida mecânica recolhidos diariamente.

Os padrões de acesso facilitados por este modelo de dados são:

- Recuperação de todos os dados para um determinado teleférico ou o resort em geral, dinâmicos e estáticos, facilitado por uma tabela chamada `SkiLifts`.
- Recuperação de todos os dados dinâmicos (incluindo usuários do teleférico, cobertura de neve, perigo de avalanche e status do teleférico) para um teleférico ou o resort em geral em uma data específica, facilitada por uma tabela chamada `SkiLifts`.
- Recuperação de todos os dados estáticos (incluindo se o teleférico for apenas para esquiadores experientes, altura na subida do teleférico e tempo de subida do teleférico) para um teleférico específico, facilitada por uma tabela chamada `SkiLifts`.
- Recuperação da data dos dados registrados para uma subida mecânica específica ou a instância geral classificada pelo total de esquiadores únicos, facilitada pelo índice secundário global da tabela SkiLifts, chamado `SkiLiftsByRiders`.

Modelo de dados de ofertas de cartão de crédito

Este modelo de dados é utilizado por uma aplicação de ofertas de cartão de crédito.

Um provedor de cartão de crédito produz ofertas ao longo do tempo. Essas ofertas incluem transferências de saldo sem taxas, aumento do limite de crédito, redução das taxas de juros, reembolso e milhas aéreas. Depois que um cliente aceita ou recusa essas ofertas, o respectivo status da oferta é atualizado de forma correspondente.

Os padrões de acesso facilitados por este modelo de dados são:

- Recuperação de registros de conta usando `AccountId`, facilitada pela tabela principal.
- Recuperação de todas as contas com poucos itens projetados, facilitada pelo índice secundário `AccountIndex`.
- Recuperação de contas e todos os registros de oferta associados a essas contas usando `AccountId`, facilitada pela tabela principal.
- Recuperação de contas e registros de ofertas específicas associados a essas contas usando `AccountId` e `OfferId`, facilitada pela tabela principal.

- Recuperação de todos os registros de oferta `ACCEPTED/DECLINED` do `OfferType` específico associado a contas usando `AccountId`, `OfferType` `eStatus`, conforme facilitada pelo índice secundário `GSI1`.
- Recuperação de ofertas e registros de itens de oferta associados usando `OfferId`, facilitada pela tabela principal.

Modelo de dados de favoritos

Este modelo de dados é usado para armazenar favoritos para os clientes.

Um cliente pode ter vários favoritos e um favorito pode pertencer a vários clientes. Este modelo de dados representa uma relação do tipo “muitos para muitos”.

Os padrões de acesso facilitados por este modelo de dados são:

- Agora, uma única consulta com o `customerId` pode retornar dados do cliente e dos favoritos.
- Um índice `ByEmail` da consulta retorna os dados do cliente por endereço de e-mail. Observe que os marcadores não são recuperados por esse índice.
- Um índice de consulta `ByUrl` obtém dados de favoritos por URL. Observe que customerId é a chave de classificação para o índice porque o mesmo URL pode ser adicionado aos favoritos por vários clientes.
- Um índice `ByCustomerFolder` da consulta obtém marcadores por pasta para cada cliente.

Histórico de versões do NoSQL Workbench

A tabela a seguir descreve as alterações importantes em cada versão da ferramenta do cliente do NoSQL Workbench.

Versão	Alteração	Descrição	Data
3.13.5	O modo de capacidade para as configurações de tabela padrão agora é sob demanda	Quando você cria uma tabela com configurações padrão, o DynamoDB cria uma tabela que usa o modo de capacidad	24 de fevereiro de 2025

Versão	Alteração	Descrição	Data
		e sob demanda em vez do modo de capacidade provision ada.	
3.13.0	Melhorias no criador de operações do NoSQL Workbench	O NoSQL Workbench agora inclui suporte nativo para o modo escuro. Operações aprimoradas de tabelas e de itens no criador de operações . As informações de solicitações do criador de operações e de resultados de itens estão disponíveis no formato JSON.	24 de abril de 2024

Versão	Alteração	Descrição	Data
3.12.0	Clonar tabelas com o NoSQL Workbench e exibir a capacidade consumida	Agora é possível clonar tabelas entre o DynamoDB local e uma conta de serviço web do DynamoDB ou entre contas de serviço web do DynamoDB para acelerar as iterações de desenvolvimento. Visualize a RCU ou a WCU consumida após a execução de uma operação usando o Operation s Builder. Corrigimo s o problema de substituição de dados ao importar de um arquivo CSV.	26 de fevereiro de 2024
3.11.0	Aprimoramentos do DynamoDB local	Agora você pode especificar a porta ao iniciar a instância integrada do DynamoDB local. O NoSQL Workbench agora pode ser instalado no Windows sem direitos de administr ador. Atualizamos os modelos de dados.	17 de janeiro de 2024

Versão	Alteração	Descrição	Data
3.10.0	Suporte nativo para o chip da Apple	O NoSQL Workbench agora inclui suporte nativo para Mac com chip da Apple. Agora você pode configurar o formato de geração de dados de amostra para atributos do tipo `Number`.	5 de dezembro de 2023
3.9.0	Melhorias no modelador de dados	O Visualizador agora aceita a confirmação de modelos de dados no DynamoDB local com a opção de sobrescrever tabelas existentes.	3 de novembro de 2022
3.8.0	Geração de dados de amostra	O NoSQL Workbench agora comporta a geração de dados de amostra para os modelos de dados do DynamoDB.	25 de setembro de 2023
3.6.0	Melhorias no criador de operações	Melhorias no gerenciamento de conexões no criador de operações. Os nomes de atributos no modelador de dados agora podem ser alterados sem excluir dados. Outras correções de erros.	11 de abril de 2023

Versão	Alteração	Descrição	Data
3.5.0	Suporte para novas regiões da AWS	O NoSQL Workbench agora é compatível com as regiões ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1 e me-west-1.	23 de fevereiro de 2023
3.4.0	Suporte ao DynamoDB local	O NoSQL Workbench agora comporta a instalação do DynamoDB local como parte do processo de instalação.	6 de dezembro de 2022
3.3.0	Suporte para operações do ambiente de gerenciamento	O Operation Builder agora é compatível com operações do ambiente de gerenciamento.	1º de junho de 2022
3.2.0	Importação e exportação de arquivo CSV	Agora você pode importar dados de amostra de um arquivo CSV na ferramenta Visualizer e também exportar os resultados de uma consulta do Operation Builder para um arquivo CSV.	11 de outubro de 2021

Versão	Alteração	Descrição	Data
3.1.0	Salvar operações	O Criador de operações no NoSQL Workbench agora oferece suporte a operações de salvar para uso posterior.	12 de julho de 2021
3.0.0	Configurações de capacidade e importação/exportação do CloudFormation	O NoSQL Workbench for Amazon DynamoDB agora oferece suporte à especificação de um modo de capacidade e de leitura/gravação para tabelas e pode importar e exportar modelos de dados no formato AWS CloudFormation.	21 de abril de 2021
2.2.0	Suporte a PartiQL	O NoSQL Workbench para Amazon DynamoDB adiciona suporte à criação de instruções PartiQL para DynamoDB.	4 de dezembro de 2020
1.1.0	Suporte a Linux.	O NoSQL Workbench para Amazon DynamoDB oferece suporte às distribuições Ubuntu, Fedora e Debian do Linux.	4 de maio de 2020

Versão	Alteração	Descrição	Data
1.0.0	NoSQL Workbench para Amazon DynamoDB – GA.	O NoSQL Workbench for Amazon DynamoDB está disponível para o público em geral.	2 de março de 2020
0.4.1	Suporte para funções do IAM e credencia is de segurança temporárias.	O NoSQL Workbench para Amazon DynamoDB adiciona suporte para funções e credenciais de segurança temporárias do AWS Identity and Access Management (IAM).	19 de dezembro de 2019
0.3.1	Suporte para o DynamoDB local (versão disponível para download).	Agora o NoSQL Workbench comporta a conexão com o DynamoDB local (versão para download) para desenvolver, criar, consultar e gerenciar tabelas do DynamoDB.	8 de novembro de 2019

Versão	Alteração	Descrição	Data
0.2.1	A demonstração do NoSQL Workbench foi lançada.	Essa é a versão inicial do NoSQL Workbench for DynamoDB. Use o NoSQL Workbench para projetar, criar, consultar e gerenciar tabelas do DynamoDB.	16 de setembro de 2019

Backup e restauração para o DynamoDB

O DynamoDB oferece backups sob demanda e para um ponto no tempo (PITR) para ajudar a proteger os dados do DynamoDB de eventos de desastre e oferece arquivamento de dados para retenção de longo prazo. Você pode fazer backup de tabelas de alguns megabytes para centenas de terabytes de dados, sem afetar a performance e a disponibilidade das aplicações de produção. Todos os backups são automaticamente criptografados, catalogados e facilmente detectáveis.

Com backups sob demanda, é possível criar um backup de snapshot da sua tabela armazenada e gerenciada pelo DynamoDB. A cobrança é feita com base no tamanho e na duração dos backups. Usando o backup sob demanda, é possível restaurar toda a tabela do DynamoDB para o estado exato em que estava quando o backup foi criado.

Há duas opções para criar e gerenciar backups sob demanda do DynamoDB:

- DynamoDB
- AWS Backup

Use o recurso de backup sob demanda do DynamoDB para criar backups completos de suas tabelas para retenção e arquivamento em longo prazo, de modo a atender às necessidades de conformidade regulatória. É possível fazer backup dos dados da tabela e restaurá-los a qualquer momento do AWS Management Console ou com uma única chamada de API.

Os backups de recuperação para um ponto no tempo (PITR) são totalmente gerenciados pelo DynamoDB e oferecem até 35 dias de pontos de recuperação em uma granularidade de segundo. Para usar a recuperação para um ponto no tempo, que são backups contínuos, ative a recuperação para um ponto no tempo (PITR) na tabela do DynamoDB. Você é cobrado com base no tamanho da tabela do DynamoDB durante o período em que a PITR esteve ativada na tabela. Ao habilitar a recuperação para um ponto no tempo (PITR) na tabela do DynamoDB, é realizado backup contínuo dos dados. Isso ajuda você a restaurar a tabela em um ponto no tempo específico dentro do período de recuperação da PITR criando uma tabela do DynamoDB com o estado exato da tabela original naquele ponto no tempo.

A recuperação em um ponto anterior no tempo ajuda a proteger as tabelas do DynamoDB contra operações acidentais de gravação ou exclusão. Com a recuperação pontual, você não precisa se preocupar com a criação, a manutenção ou a programação de backups sob demanda. Por exemplo,

suponhamos que um script de teste seja gravado acidentalmente em uma tabela do DynamoDB de produção.

Com a recuperação para um ponto no tempo, é possível recuperar a tabela para qualquer ponto no tempo durante os últimos 35 dias. É possível definir o período de recuperação para qualquer valor entre 1 e 35 dias. Depois de habilitar a recuperação para um ponto no tempo, você poderá restaurar para qualquer ponto no tempo entre cinco minutos antes da hora atual até o período de recuperação configurado. O DynamoDB mantém backups incrementais da tabela.

Além disso, as operações point-in-time não afetam o desempenho ou as latências de API.

Você pode restaurar uma tabela do DynamoDB para um ponto no tempo usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API do DynamoDB. O processo de recuperação em um ponto anterior no tempo sempre restaura uma nova tabela.

Para obter mais informações sobre disponibilidade de regiões da AWS e preços, consulte Preços do Amazon DynamoDB.

> **Note**
>
> - A marcação e o controle de acesso por atributo (ABAC) não são aceitos em backups do DynamoDB. Para usar o ABAC com backups, recomendamos usar AWS Backup.
> - As tags não são preservadas nas tabelas restauradas. É necessário adicionar tags às tabelas restauradas para usar condições baseadas em tags em políticas.

O vídeo a seguir dará a você uma visão introdutória do conceito de backup e restauração e falará mais sobre a recuperação para um ponto no tempo.

Backup e restauração

Tópicos

- Backups para um ponto no tempo do DynamoDB
- Usar backup e restauração sob demanda do DynamoDB
- Conceitos básicos sobre faturamento do Amazon DynamoDB para backups
- Restaurar uma tabela do DynamoDB
- Como usar a AWS Backup com o DynamoDB

Backups para um ponto no tempo do DynamoDB

A recuperação para um ponto no tempo (PITR) do Amazon DynamoDB fornece backups contínuos dos dados de tabelas do DynamoDB. Os backups de recuperação para um ponto no tempo (PITR) são totalmente gerenciados pelo DynamoDB e oferecem até 35 dias de pontos de recuperação em uma granularidade de segundo. Com a recuperação pontual, você não precisa se preocupar com a criação, a manutenção ou a programação de backups sob demanda. Esta seção apresenta uma visão geral de como o processo funciona no DynamoDB.

Antes de começar

Antes de habilitar a recuperação em um ponto anterior no tempo (PITR) em uma tabela do Amazon DynamoDB, considere o seguinte:

- A configuração de `RecoveryPeriodinDays` permite reduzir o período durante o qual os backups contínuos são realizados. Por padrão, `RecoveryPeriodinDays` é 35. No entanto, é possível configurá-lo como qualquer valor entre 1 e 35. Reduzir o `RecoveryPeriodinDays` não tem impacto no preço da PITR, pois ele se baseia no tamanho da tabela e dos índices secundários locais.
- Se desabilitar a recuperação pontual depois reabilitá-la em uma tabela, você redefinirá a hora de início para a qual pode recuperar essa tabela. Dessa forma, você só pode restaurar imediatamente essa tabela usando a tabela `LatestRestorableDateTime`.
- Você pode habilitar a recuperação point-in-time em cada réplica local de uma tabela global. Quando você restaure a tabela, o backup restaura uma tabela independente que não faz parte da tabela global. Se estiver usando Global Tables versão 2019.11.21 (atual) das tabelas globais, você poderá criar uma tabela global usando a tabela restaurada. Para ter mais informações, consulte Como funcionam as tabelas globais do DynamoDB.
- Você também pode restaurar os dados da tabela do DynamoDB entre regiões da AWS para que a tabela restaurada seja criada em uma região diferente daquela na qual a tabela de origem reside. Você pode fazer restaurações entre regiões comerciais da AWS, regiões da AWS China e regiões da AWS GovCloud (EUA). Você paga somente pelos dados transferidos para fora da região de origem e pela restauração para uma nova tabela na região de destino.
- O AWS CloudTrail registra todas as ações de console e API da recuperação point-in-time para habilitar registro, monitoramento contínuo e auditoria. Para ter mais informações, consulte Registrar em log as operações do DynamoDB usando o AWS CloudTrail.

Tópicos

- Habilitar a recuperação para um ponto no tempo no DynamoDB

Habilitar a recuperação para um ponto no tempo no DynamoDB

A recuperação em um ponto anterior no tempo (PITR) do Amazon DynamoDB fornece backups automáticos dos dados de tabelas do DynamoDB. Esta seção apresenta uma visão geral de como o processo funciona no DynamoDB.

> **Note**
>
> O DynamoDB cobra pela PITR com base no tamanho de cada tabela do DynamoDB, incluindo dados das tabelas e índices secundários locais. Alterar a janela de recuperação (por exemplo, de 35 dias para 1 dia) não reduz o preço. O custo permanece o mesmo, independentemente do período de recuperação escolhido. O período máximo de recuperação configurado não afeta o preço cobrado pela ativação da PITR. Para determinar suas cobranças de backup, o DynamoDB monitora continuamente o tamanho das tabelas que têm a PITR ativada. Você será cobrado pelo uso da PITR até desativá-la em cada tabela.

Tópicos

- Habilitar a recuperação para um ponto no tempo
- Habilitar a PITR (console)
- Habilitar a PITR (AWS CLI)
- Habilitar a PITR (AWS CloudFormation)
- Habilitar a PITR (API)
- Período de recuperação
- Editar PITR
- Excluir uma tabela com a PITR habilitada

Habilitar a recuperação para um ponto no tempo

Você pode habilitar a recuperação pontual usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API do DynamoDB. Quando habilitada, a recuperação pontual oferece backups contínuos até que você a desative explicitamente.

Depois de habilitar a recuperação para um ponto no tempo, será possível restaurar para qualquer ponto no tempo entre `EarliestRestorableDateTime` e `LatestRestorableDateTime`. `LatestRestorableDateTime` normalmente é cinco minutos antes da hora atual. Para obter mais informações, consulte Restaurar uma tabela do DynamoDB para um ponto no tempo.

> **Note**
>
> O processo de recuperação pontual sempre restaura uma nova tabela.

Habilitar a PITR (console)

Para habilitar a PITR usando o console do DynamoDB

1. Navegue até o console do DynamoDB.
2. Selecione Tabelas no painel de navegação à esquerda e selecione sua tabela do DynamoDB.
3. Na guia Backups, para a opção Recuperação para um ponto no tempo, selecione Editar.
4. Escolha Ativar recuperação para um ponto no tempo.
5. Escolha um valor entre 1 e 35 para o período de recuperação de backup. Isso indica o período máximo durante o qual o backup contínuo pode ser recuperado.

Habilitar a PITR (AWS CLI)

> **Note**
>
> Se você receber erros ao executar comandos da AWS CLI, consulte Troubleshoot AWS CLI errors. Verifique se está usando a versão mais recente da AWS CLI.

Execute o comando update-continuous-backups com a configuração point-in-time-recovery-specification ativada:

```
aws dynamodb update-continuous-backups \
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

Habilitar a PITR (AWS CloudFormation)

Use o recurso AWS::DynamoDB::Table com a propriedade `PointInTimeRecoverySpecification` ativada:

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

Exemplo de sintaxe de solicitação:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
   },
   "TableName": "string"
}
```

Habilitar a PITR (API)

Execute a operação da API UpdateContinuousBackups com o parâmetro `PointInTimeRecoverySpecification` ativado.

Exemplo de sintaxe de solicitação:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
```

```
}
```

Exemplo de sintaxe de resposta:

```
{
   "ContinuousBackupsDescription": {
      "ContinuousBackupsStatus": "string",
      "PointInTimeRecoveryDescription": {
         "PointInTimeRecoveryStatus": "string",
         "EarliestRestorableDateTime": number,
         "RecoveryPeriodInDays": number,
         "LatestRestorableDateTime": number
      }
   }
}
```

Python

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.update_continuous_backups(
    TableName=<table_name>,
    PointInTimeRecoverySpecification={
        'PointInTimeRecoveryEnabled': True,
        'RecoveryPeriodInDays': 35
    }
)
```

Período de recuperação

É possível definir o período de recuperação para backups contínuos como qualquer número entre 1 e 35 dias. Esse `RecoveryPeriodInDays` determina o período durante o qual os backups contínuos são mantidos. Por exemplo, se você definir esse valor como 30 dias, só poderá restaurar sua tabela para qualquer momento dos últimos 30 dias.

> **Note**
>
> O DynamoDB cobra pela PITR com base no tamanho de cada tabela do DynamoDB, incluindo dados das tabelas e índices secundários locais. O período máximo de recuperação

configurado não afeta o preço cobrado pela ativação da PITR. Para ter detalhes de preço, consulte Preço do DynamoDB.

Editar PITR

É possível editar a configuração de PITR na tabela e alterar o período de recuperação. Se você alterar o período de recuperação e aumentá-lo para um valor maior do que o definido anteriormente, o `EarliestRestorePoint` não mudará imediatamente. Como o período de recuperação é uma janela contínua, o DynamoDB continuará fazendo backups automáticos até que o novo período aumentado seja atingido. Se você alterar o período de recuperação e diminuí-lo para um valor menor do que o definido anteriormente, o `EarliestRestorePoint` diminuirá imediatamente para corresponder ao período de recuperação, e quaisquer backups contínuos que estejam fora do novo valor definido não serão recuperáveis.

Excluir uma tabela com a PITR habilitada

Quando você exclui uma tabela com a recuperação pontual habilitada, o DynamoDB cria automaticamente um backup (chamado backup do sistema) e o mantém por 35 dias (sem custo adicional). É possível usar o backup do sistema para restaurar a tabela excluída no estado em que estava antes da exclusão. Todos os backups do sistema seguem uma convenção de nomenclatura padrão: *`table-name`*`$DeletedTableBackup`.

> **Note**
>
> Depois que uma tabela com a recuperação para um ponto no tempo habilitada for excluída, você poderá usar o backup do sistema para restaurar essa tabela em um único ponto no tempo. O backup do sistema será criado após a exclusão da tabela e é um snapshot da tabela logo antes de ela ser excluída.

Usar backup e restauração sob demanda do DynamoDB

O Amazon DynamoDB oferece suporte aos recursos autônomos de backup e restauração sob demanda. Esses recursos estão disponíveis para você, independentemente de você usar o AWS Backup ou não.

Você pode usar o recurso de backup sob demanda do DynamoDB para criar backups completos de suas tabelas para retenção e arquivamento em longo prazo de modo a atender às necessidades de

conformidade regulatória. Você pode fazer backup dos dados de tabelas e restaurá-los a qualquer momento com um único clique no Console de Gerenciamento da AWS ou com uma única chamada de API. As ações de backup e restauração são executadas sem causar impactos na performance nem na disponibilidade da tabela.

Você pode criar backups de tabela usando o console, a AWS Command Line Interface (AWS CLI) ou a API do DynamoDB. Para obter mais informações, consulte Fazer backup de uma tabela do DynamoDB.

Para obter mais informações sobre a restauração de tabelas a partir de backups, consulte Restaurar uma tabela do DynamoDB de um backup.

Fazer backup e restaurar tabelas do DynamoDB com o DynamoDB: como funciona

Você pode usar o recurso de backup sob demanda do DynamoDB para criar backups completos das suas tabelas do Amazon DynamoDB. Esse recurso está disponível independentemente do backup da AWS. Esta seção oferece uma visão geral do que ocorre durante o processo de backup e restauração do DynamoDB.

Backups

Ao criar um backup sob demanda com o DynamoDB, um marcador de tempo da solicitação é catalogado. O backup é criado de forma assíncrona aplicando todas as alterações desde o momento da solicitação até o último snapshot da tabela completa. As solicitações de backup do DynamoDB são processadas instantaneamente e a restauração é disponibilizada em minutos.

> **Note**
>
> Toda vez que você criar um backup sob demanda, será feito backup de todos os dados da tabela. Não há limite para o número de backups sob demanda que podem ser realizados.

Todos os backups no DynamoDB funcionam sem consumir o throughput provisionado na tabela.

No DynamoDB, os backups não garantem consistência causal entre os itens; no entanto, a distorção entre as atualizações em um backup geralmente é muito inferior a um segundo.

Enquanto um backup estiver em andamento, você não poderá fazer o seguinte:

- Pausar ou cancelar a operação de backup.
- Excluir a tabela de origem do backup.
- Desativar backups em uma tabela se houver um backup em andamento para essa tabela.

Caso não queira criar scripts de programação e trabalhos de limpeza, você poderá usar o AWS Backup para criar planos de backup com programações e políticas de retenção para suas tabelas do DynamoDB. O AWS Backup executa os backups e os exclui quando eles expiram. Para mais informações, consulte o Guia do desenvolvedor do AWS Backup.

Além do AWS Backup, você pode programar backups periódicos ou futuros usando funções de AWS Lambda. Para obter mais informações, veja a postagem do blog A serverless solution to schedule your Amazon DynamoDB On-Demand Backup (Uma solução com tecnologia sem servidor para agendar o backup sob demanda do Amazon DynamoDB).

Se estiver usando o console, todo backup criado usando o AWS Backup será listado na guia Backups com o Backup type (Tipo de backup) definido como `AWS`.

> **Note**
>
> Você não pode excluir backups marcados com um Backup type (Tipo de backup) AWS usando o console do DynamoDB. Para gerenciar esses backups, use o console do AWS Backup.

Para saber como realizar um backup, consulte Fazer backup de uma tabela do DynamoDB.

Restaurações

Você restaura uma tabela sem consumir o throughput provisionado na tabela. Você pode fazer uma restauração completa da tabela usando o backup do DynamoDB ou pode definir as configurações da tabela de destino. Ao fazer uma restauração, você pode alterar as seguintes configurações de tabela:

- Índices secundários globais (GSIs)
- Índices secundários locais (LSIs)
- Modo de faturamento
- Capacidade de leitura e gravação provisionada
- Configurações de criptografia

> **Important**
>
> Quando você faz uma restauração completa da tabela, a tabela de destino é definida com as mesmas unidades de capacidade de leitura e unidades de capacidade de gravação provisionadas da tabela de origem, conforme gravado no momento em que o backup foi solicitado. O processo de restauração também restaura os índices secundários locais e os índices secundários globais.

Você também pode restaurar os dados da tabela do DynamoDB em regiões da AWS de modo que a tabela restaurada seja criada em uma região diferente daquela na qual o backup reside. Você pode fazer restaurações entre regiões comerciais da AWS, regiões da AWS China e regiões da AWS GovCloud (EUA). Você paga somente pelos dados transferidos para fora da região de origem e pela restauração para uma nova tabela na região de destino.

As restaurações poderão ser mais rápidas e econômicas se você optar por excluir a criação de alguns ou todos os índices secundários na nova tabela restaurada.

Você deve configurar manualmente os itens a seguir na tabela restaurada:

- Políticas de Auto Scaling
- Políticas do AWS Identity and Access Management (IAM)
- Métricas e alarmes do Amazon CloudWatch
- Tags
- Configurações de fluxo
- Configurações de vida útil (TTL)
- Configurações de proteção contra exclusão
- Configurações de recuperação para um ponto no tempo (PITR)

Só é possível restaurar os dados completos da tabela para uma nova tabela por meio de backup. Você pode gravar na tabela restaurada somente depois que ela fica ativa.

> **Note**
>
> Você não pode substituir uma tabela existente durante uma operação de restauração.

As métricas de serviço mostram que 95% das restaurações da tabela dos clientes são concluídas em menos de uma hora. No entanto, os tempos de restauração estão diretamente relacionados à configuração das tabelas (como o tamanho das tabelas e o número de partições subjacentes) e outras variáveis relacionadas. Uma prática recomendada ao planejar a recuperação de desastres é documentar regularmente os tempos médios de conclusão da restauração e estabelecer como esses tempos afetam seu objetivo geral de tempo de recuperação.

Para saber como realizar uma restauração, consulte Restaurar uma tabela do DynamoDB de um backup.

Você pode usar políticas do IAM para controle de acesso. Para ter mais informações, consulte Usar o IAM com backup e restauração do DynamoDB.

Todas as ações de backup e restauração de console e API são capturadas e registradas no AWS CloudTrail para registro, monitoramento contínuo e auditoria.

Fazer backup de uma tabela do DynamoDB

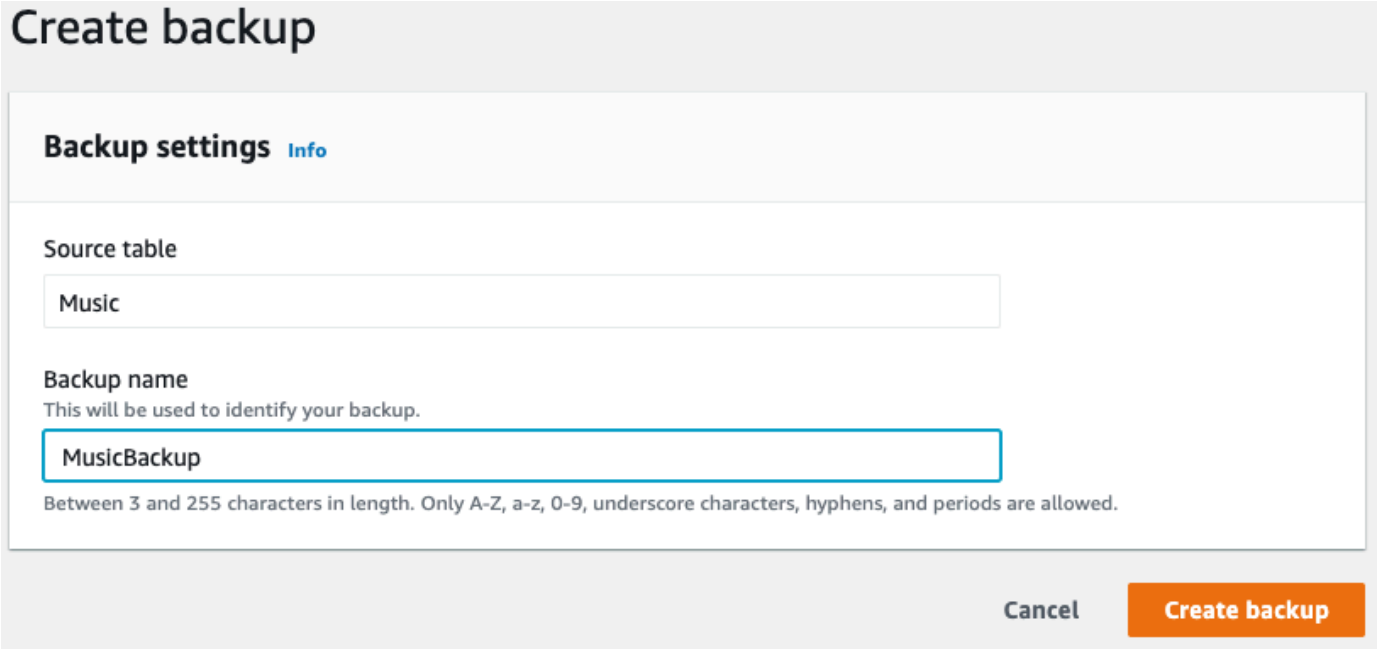
Esta seção descreve como usar o console do Amazon DynamoDB ou a AWS Command Line Interface para fazer backup de uma tabela.

Criar backup de uma tabela (console)

Siga estas etapas para criar um backup chamado `MusicBackup` de uma tabela `Music` existente usando o AWS Management Console.

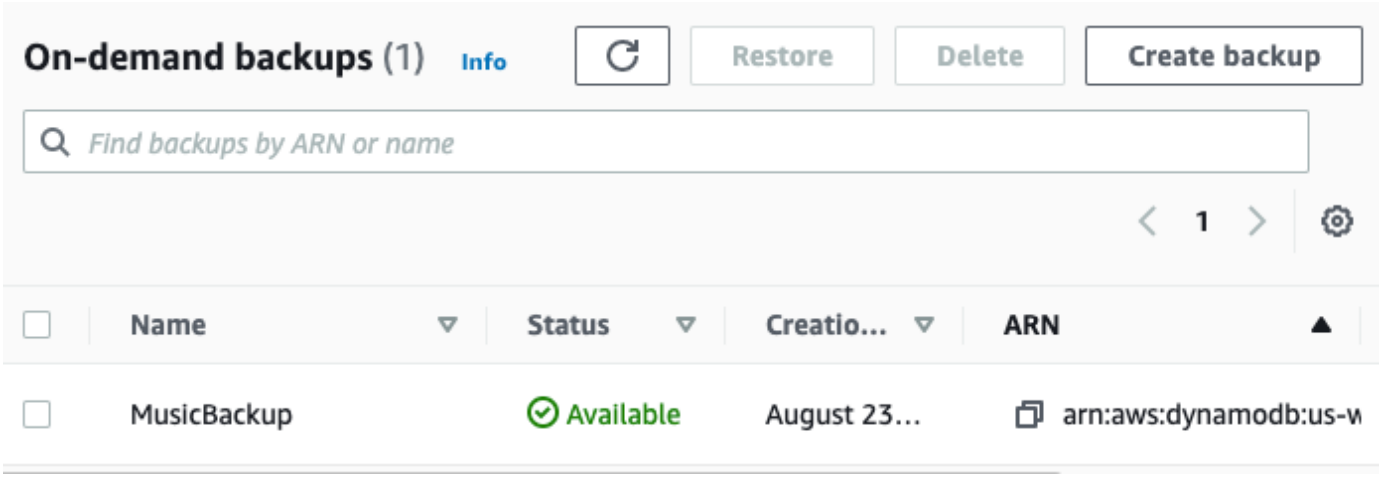
Para criar um backup de tabela

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. Para criar um backup, realize um dos seguintes procedimentos:
   - Na guia Backups da tabela `Music`, selecione Create backup (Criar backup).
   - No painel de navegação, no lado esquerdo do console, selecione Backups. Em seguida, escolha Create backup.
3. Verifique se `Music` é o nome da tabela e insira **`MusicBackup`** como o nome do backup. Em seguida, escolha Criar backup para criar o backup.



> **Note**
>
> Se você criar backups usando a seção Backups no painel de navegação, a tabela não ficará pré-selecionada para você. Você precisará escolher manualmente o nome da tabela de origem para o backup.

Enquanto o backup estiver sendo criado, o status do backup será Creating. Depois que o backup for finalizado, o status dele mudará para Available (Disponível).



Criar backup de uma tabela (AWS CLI)

Siga estas etapas para criar um backup de uma tabela `Music` existente usando a AWS CLI.

Para criar um backup de tabela

- Crie um backup com o nome `MusicBackup` para a tabela `Music`.

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

Durante a criação do backup, o status dele será `CREATING`.

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

Quando o processo de backup for concluído, o `BackupStatus` deverá mudar para `AVAILABLE`. Para confirmar, use o comando `describe-backup`. Você pode obter o valor de entrada de `backup-arn` com a saída da etapa anterior ou usando o comando `list-backups`.

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

Para acompanhar os backups, você pode usar o comando `list-backups`. Ele lista todos os backups em status `CREATING` ou `AVAILABLE`.

```
aws dynamodb list-backups
```

Os comandos `list-backups` e `describe-backup` são úteis para verificar informações sobre a tabela de origem do backup.

Restaurar uma tabela do DynamoDB de um backup

Esta seção descreve como restaurar uma tabela de um backup usando o console do Amazon DynamoDB ou a AWS Command Line Interface (AWS CLI).

> **Note**
>
> Se desejar usar a AWS CLI, você precisará configurá-la primeiro. Para ter mais informações, consulte Acessar o DynamoDB.

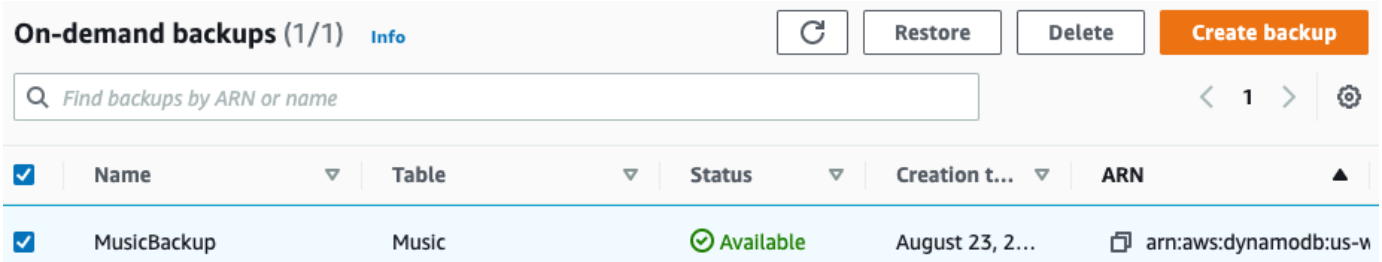
Restaurar uma tabela de um backup (console)

O procedimento a seguir mostra como restaurar a tabela `Music` usando o arquivo `MusicBackup` que é criado no tutorialFazer backup de uma tabela do DynamoDB.

> **Note**
>
> Este procedimento supõe que a tabela `Music` não existe mais antes de restaurá-la usando o arquivo `MusicBackup`.

Para restaurar uma tabela de um backup

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. Na lista de backups, escolha `MusicBackup`.



4. Escolha Restore.
5. Insira **`Music`** como o novo nome da tabela. Confirme o nome do backup e outros detalhes. Em seguida, escolha Restore table (Restaurar tabela) para iniciar o processo de restauração.

> **Note**
>
> Você pode restaurar a tabela para a mesma região da AWS ou para uma região diferente daquela na qual o backup reside. Você também pode excluir a criação de índices secundários na nova tabela restaurada. Além disso, você pode especificar um modo de criptografia diferente.
>
> As tabelas restauradas a partir de backups são sempre criadas usando a classe de tabela do DynamoDB Standard.

Restore table from backup Info

Restoring a table from a backup will restore it as a new table.

Restore settings

Name of restored table

This name will identify your restored table.

Music

Between 3 and 255 characters in length. Only A–Z, a–z, 0–9, underscore characters, hyphens, and periods allowed.

Secondary indexes

- (•) Restore the entire table
  Your restored table will include all local and global secondary indexes.
- ( ) Restore the table without secondary indexes
  Your restored table will exclude all local and global secondary indexes. Restoring this way can be faster and more cost efficient.

Destination AWS Region

- (•) Same Region (Oregon)
  Restore the table to the same Region as the original table.
- ( ) Cross-Region
  Restore the table to a different Region for greater redundancy but with higher data transfer costs.

▼ Encryption at rest - *optional*

All user data stored in Amazon DynamoDB is fully encrypted at rest. By default, Amazon DynamoDB manages the encryption key, and you are not charged any fee for using it.

Encryption key management Info

- (•) Owned by Amazon DynamoDB
  The key is owned and managed by DynamoDB. You are not charged an additional fee for using this customer master key (CMK).
- ( ) AWS managed CMK
  The key is stored in your account and is managed by AWS Key Management Service (AWS KMS). AWS KMS charges apply.
- ( ) Stored in your account, and owned and managed by you
  Choose a key that is owned and managed by you, and stored in AWS KMS.

ⓘ The time it takes to restore a table from a backup can vary and is based on multiple variables. After your table is restored from the backup, you might need to reapply configuration settings. Learn more

Cancel Restore

A tabela que está sendo restaurada é mostrada com o status Creating (Em criação). Quando o processo de restauração for concluído, o status da tabela `Music` mudará para Active (Ativo).

Restaurar uma tabela de um backup (AWS CLI)

Siga estas etapas para usar a AWS CLI para restaurar a tabela `Music` usando o `MusicBackup` que é criado no tutorial Fazer backup de uma tabela do DynamoDB.

Para restaurar uma tabela de um backup

1. Confirme o backup que você deseja restaurar usando o comando `list-backups`. Este exemplo usa `MusicBackup`.

   ```
   aws dynamodb list-backups
   ```

   Para obter outros detalhes sobre o backup, use o comando `describe-backup`. É possível obter a entrada `backup-arn` da etapa anterior:

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. Restaure a tabela por meio do backup. Nesse caso, o `MusicBackup` restaura a tabela `Music` para a mesma região da AWS.

   ```
   aws dynamodb restore-table-from-backup \
   --target-table-name Music \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

3. Restaure a tabela do backup com configurações de tabela personalizadas. Nesse caso, o `MusicBackup` restaura a tabela `Music` e especifica um modo de criptografia para a tabela restaurada.

> **Note**
>
> O parâmetro `sse-specification-override` assume os mesmos valores do parâmetro `sse-specification-override` usado no comando `CreateTable`. Para saber mais, consulte Gerenciar tabelas criptografadas no DynamoDB.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Você pode restaurar a tabela para outra região da AWS diferente daquela na qual o backup reside.

> **Note**
>
> - O parâmetro `sse-specification-override` é obrigatório para restaurações entre regiões, mas opcional para restaurações na mesma região da tabela de origem.
> - Ao realizar uma restauração entre regiões na linha de comando, você deve definir a região padrão da AWS como a região de destino desejada. Para saber mais, consulte Opções de linha de comando no Guia do usuário do AWS Command Line Interface.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Você pode substituir o modo de faturamento e o throughput provisionado para a tabela restaurada.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
```

```
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d \
--billing-mode-override PAY_PER_REQUEST
```

Você pode excluir a criação de alguns ou todos os índices secundários na tabela restaurada.

> **Note**
>
> As restaurações poderão ser mais rápidas e econômicas se você excluir a criação de alguns ou todos os índices secundários na tabela restaurada.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581081403719-db9c1f91 \
--global-secondary-index-override '[]' \
--sse-specification-override Enabled=true,SSEType=KMS
```

> **Note**
>
> Os índices secundários fornecidos devem corresponder a índices existentes correspondentes. Você não pode criar novos índices no momento da restauração.

Você pode usar uma combinação de diferentes substituições. Por exemplo, você pode usar um único índice secundário global e alterar o throughput provisionado ao mesmo tempo, da seguinte forma.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:eu-west-1:123456789012:table/Music/
backup/01581082594992-303b6239 \
--billing-mode-override PROVISIONED \
--provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100 \
--global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEYS_
 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Para verificar a restauração, use o comando `describe-table` para descrever a tabela `Music`.

```
aws dynamodb describe-table --table-name Music
```

A tabela que está sendo restaurada do backup é mostrada com o status Creating (Em criação). Quando o processo de restauração for concluído, o status da tabela `Music` mudará para Active (Ativo).

> **Important**
>
> Enquanto a restauração estiver em andamento, não modifique nem exclua a política de função do IAM. Caso contrário, poderá haver um comportamento inesperado. Por exemplo, suponha que você tenha removido as permissões de gravação para uma tabela enquanto essa tabela estava sendo restaurada. Nesse caso, a operação `RestoreTableFromBackup` subjacente não conseguiria gravar na tabela nenhum dos dados restaurados.
> Assim que a operação de restauração estiver concluída, você poderá modificar ou excluir a política de função do IAM.
> Políticas do IAM envolvendo restrições de IP de origem para acessar a tabela de restauração de destino devem ter a chave `aws:ViaAWSService` definida como `false` para garantir que as restrições se apliquem apenas a solicitações feitas diretamente por um responsável principal. Caso contrário, a restauração será cancelada.
> Se o backup for criptografado com uma Chave gerenciada pela AWS ou uma chave gerenciada pelo cliente, não desabilite nem exclua a chave enquanto a restauração estiver em andamento. Caso contrário, ocorrerá falha.
> Depois que a operação de restauração for concluída, você poderá alterar a chave de criptografia da tabela restaurada e desabilitar ou excluir a chave antiga.

Excluir um backup de tabela do DynamoDB

Esta seção descreve como usar o AWS Management Console ou a AWS Command Line Interface (AWS CLI) para excluir um backup de tabela do Amazon DynamoDB.

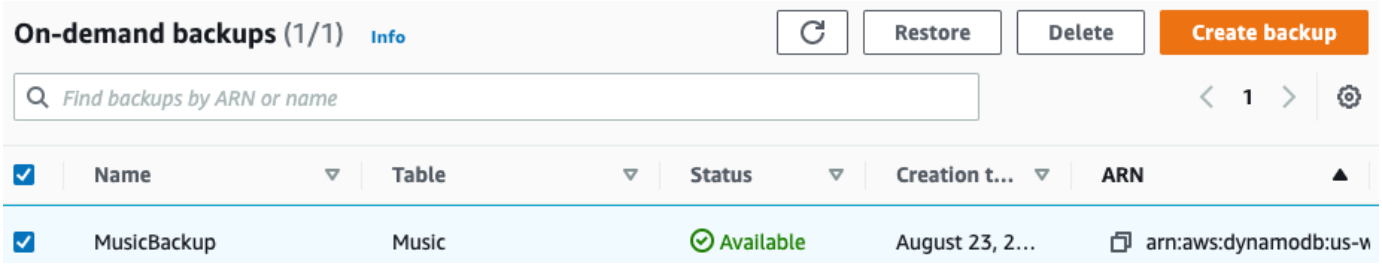
> **Note**
>
> Se desejar usar a AWS CLI, você precisará configurá-la primeiro. Para ter mais informações, consulte Uso do AWS CLI.

Excluir o backup de uma tabela (console)

O procedimento a seguir mostra como usar o console para excluir o `MusicBackup` que é criado no tutorial Fazer backup de uma tabela do DynamoDB.

Para excluir um backup

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. Na lista de backups, escolha `MusicBackup`.



4. Escolha Excluir. Confirme que você deseja excluir o backup digitando **`delete`** e clicando em Delete (Excluir).

Excluir um backup da tabela (AWS CLI)

O exemplo a seguir exclui o backup de uma tabela existente `Music` usando a AWS CLI.

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

Usar o IAM com backup e restauração do DynamoDB

Você pode usar o AWS Identity and Access Management (IAM) para restringir as ações de backup e restauração do Amazon DynamoDB a alguns recursos. As APIs `CreateBackup` e `RestoreTableFromBackup` operam por tabela.

Para obter mais informações sobre como usar as políticas do IAM no DynamoDB, consulte Políticas baseadas em identidade para o DynamoDB.

Veja a seguir exemplos de políticas do IAM que você pode usar para configurar funcionalidades específicas de backup e restauração no DynamoDB.

Exemplo 1: permitir ações CreateBackup e RestoreTableFromBackup

Esta política do IAM concede permissões para execução das ações `CreateBackup` e `RestoreTableFromBackup` do DynamoDB em todas as tabelas:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

Important

As permissões RestoreTableFromBackup do DynamoDB são necessárias no backup de origem, e as de leitura e gravação do DynamoDB na tabela de destino são necessárias para a funcionalidade de restauração.
As permissões RestoreTableToPointInTime do DynamoDB são necessárias no backup de origem, e as de leitura e gravação do DynamoDB na tabela de destino são necessárias para a funcionalidade de restauração.

Exemplo 2: permitir CreateBackup e negar RestoreTableFromBackup

Esta política do IAM concede permissões para a ação `CreateBackup` e nega a ação `RestoreTableFromBackup`:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

Exemplo 3: permitir ListBackups e negar CreateBackup e RestoreTableFromBackup

Esta política do IAM concede permissões para a ação `ListBackups` e recusa as ações `CreateBackup` e `RestoreTableFromBackup`:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

Exemplo 4: permitir ListBackups e negar DeleteBackup

Esta política do IAM concede permissões para a ação `ListBackups` e nega a ação `DeleteBackup`:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

Exemplo 5: permitir RestoreTableFromBackup e DescribeBackup para todos os recursos e recusar DeleteBackup para um backup específico

Esta política do IAM concede permissões para as ações `RestoreTableFromBackup` e `DescribeBackup` e nega a ação `DeleteBackup` para um recurso de backup específico:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DeleteBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        }
    ]
```

```
}
```

> **Important**
>
> As permissões RestoreTableFromBackup do DynamoDB são necessárias no backup de origem, e as de leitura e gravação do DynamoDB na tabela de destino são necessárias para a funcionalidade de restauração.
> As permissões RestoreTableToPointInTime do DynamoDB são necessárias no backup de origem, e as de leitura e gravação do DynamoDB na tabela de destino são necessárias para a funcionalidade de restauração.

Exemplo 6: permitir CreateBackup para uma tabela específica

Esta política do IAM concede permissões para a ação `CreateBackup` somente na tabela `Movies`:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

Exemplo 7: permitir ListBackups

Esta política do IAM concede permissões para a ação `ListBackups`:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> Não é possível conceder permissões para a ação `ListBackups` em uma tabela específica.

Exemplo 8: permitir acesso a características do AWS Backup

Você precisará de permissões de API para a ação `StartAwsBackupJob` para obter um backup bem-sucedido com recursos avançados, e a ação `dynamodb:RestoreTableFromAwsBackup` para restaurar esse backup com êxito.

A seguinte política do IAM concede ao AWS Backup as permissões para acionar backups com recursos avançados e restaurações. Observe também que, se as tabelas forem criptografadas, a política precisará de acesso à chave do KMS da AWS.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:StartAwsBackupJob",
                "dynamodb:DescribeTable",
```

```
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        },
        {
            "Sid": "AllowRestoreFromAwsBackup",
            "Effect": "Allow",
            "Action": [
                "dynamodb:RestoreTableFromAwsBackup"
            ],
            "Resource": "*"
        }
    ]
}
```

Exemplo 9: negar RestoreTableToPointInTime para uma tabela de origem específica

A seguinte política do IAM nega permissões para a ação `RestoreTableToPointInTime` a uma tabela de origem específica:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

Exemplo 10: negar RestoreTableFromBackup para uma tabela de origem específica

A seguinte política do IAM nega permissões para a ação `RestoreTableToPointInTime` a todos os backups de uma tabela de origem específica:

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/*"
        }
    ]
}
```

Conceitos básicos sobre faturamento do Amazon DynamoDB para backups

Este guia fornece detalhes sobre como o faturamento do DynamoDB funciona para backups. Analisaremos os vários componentes que contribuem para o custo geral, fornecendo explicações claras e exemplos práticos.

O DynamoDB oferece backups sob demanda e para um ponto no tempo (PITR) para ajudar a proteger os dados do DynamoDB de eventos de desastre e oferece arquivamento de dados para retenção de longo prazo.

Como funciona

Os backups sob demanda do DynamoDB são cobrados mensalmente. Se você fizer um backup em um determinado dia do mês, verá uma única cobrança por esse backup calculada para os dias restantes do mês (exemplo: ao criar um backup no dia 27, você só receberá cobrança pelos poucos dias restantes desse mês, aplicada como uma única cobrança no dia 27).

Se você retiver os backups feitos anteriormente nos meses subsequentes, sempre verá uma cobrança mensal completa por esse backup aplicada no primeiro dia. Se o backup for removido antes do final do mês, as cobranças serão ajustadas com base no uso real.

Por exemplo, se você tiver criado um backup em 27 de julho e ele for mantido até o mês de agosto, você verá as seguintes cobranças por esse backup:

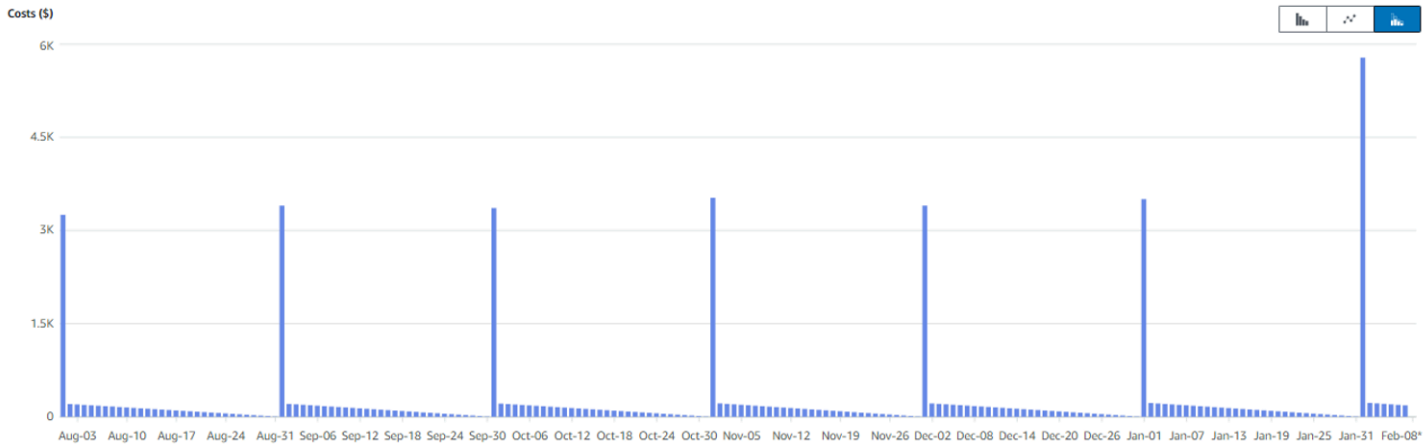
- Uma cobrança em 27 de julho pelos dias restantes de julho.
- Uma cobrança em 1.º de agosto para todo o mês de agosto.
- Uma cobrança no dia 1.º de cada mês subsequente em que o backup existir.

Quando os backups são mantidos para tabelas do DynamoDB, você pode observar que a despesa com a métrica de uso `DynamoDB (Region)-TimedBackupStorage-ByteHrs` parece anormalmente alta no primeiro dia do mês. Além disso, se você verificar essa métrica no início de um novo mês e compará-la com os ciclos de faturamento anteriores, poderá observar um aumento aparentemente grande no uso. Isso faz parte do design. No primeiro dia de cada mês, todos os backups existentes do DynamoDB terão cobranças de uso do mês inteiro aplicadas. Todos os backups do DynamoDB removidos durante o mês terão suas despesas de uso rateadas para refletir o uso real. Por isso, você poderá ver a cobrança (aplicada no dia 1.º) diminuir ao longo do mês. Isso ocorre porque as políticas de retenção aplicam expirações ou exclusões manuais aos backups transferidos. Isso será examinado em um cenário abaixo.

Da mesma forma, você notará aumentos menores ao longo do mês à medida que novos backups forem criados, e a respectiva cobrança será aplicada no dia em que eles forem criados até o restante do mês.

Exemplo de faturamento de backup do DynamoDB

Veja um exemplo do que você pode ver no Explorador de Custos no início do mês:



Observe que 1.º de fevereiro parece ter um pico bem maior em relação aos meses anteriores. Vamos explicar por que isso ocorre.

Na página de preços do DynamoDB:

“O tamanho total do armazenamento de backup cobrado a cada mês é a soma de todos os backups de tabelas do DynamoDB. O DynamoDB monitora o tamanho dos backups sob demanda continuamente durante todo o mês para determinar as cobranças de backup”.

Isso explica por que a fatura mostra consistentemente um grande aumento no uso no primeiro dia de cada mês. Todos os backups existentes que entrarem em um novo mês terão as cobranças de um mês inteiro aplicadas ao primeiro. Em outras palavras, se você começar o mês com 300 backups do DynamoDB, verá as cobranças de uso de um mês inteiro aplicadas no primeiro dia do mês para todos os 300 backups.

Novos backups feitos ao longo do mês incorrerão em cobranças de uso desde o dia da criação até o final do mês.

E se o backup for excluído no meio do mês?

Veja alguns cenários a serem considerados:

1. Se um backup do mês anterior for excluído no dia 15 do mês atual, as cobranças de uso desse backup, ainda aplicadas no dia primeiro, serão ajustadas de acordo com o uso real, e não com base no mês completo de uso presumido que foi aplicado anteriormente. O exemplo abaixo explica isso mais detalhadamente.
2. Quando você cria um backup durante o mês, as cobranças de uso relativas ao restante do mês são aplicadas no dia em que ele foi criado. No entanto, se você excluir esse backup antes do final

do mês, suas cobranças de uso serão ajustadas para incluir somente os dias em que o backup estava ativo, mas continuam sendo aplicadas na data da respectiva criação.

Por que o uso do mês atual parece ser muito maior no primeiro mês do que no mês anterior e o que acontece se eu remover os backups?

Para responder a essa importante pergunta em duas partes, vamos configurar um cenário de exemplo usando as seguintes informações:

- Duração do mês: 30 dias
- Frequência de backup do DynamoDB: 10/dia, 300/mês
- Política de retenção de backup do DynamoDB: 30 dias
- Custo por backup do DynamoDB: USD 2/dia, USD 60 /mês
- Total do 1.º dia do mês anterior (`TimedBackupStorage-ByteHrs`, verificado no 1.º dia do mês atual): USD 9.300
- Total do mês anterior (`TimedBackupStorage-ByteHrs`): USD 18.600
- Total do 1.º dia do mês atual (`TimedBackupStorage-ByteHrs`, verificado no 1.º dia): USD 18.000
- Mudanças no uso do DynamoDB mês a mês: nenhuma

Usando as informações acima, podemos ver que 300 backups foram criados no mês anterior com uma política de mantê-los por 30 dias. No primeiro dia de um novo mês, todos esses backups ainda permanecem, pois ainda não atingiram o fim do período de recuperação. No entanto, a cada dia que passa, os conjuntos de backups mais antigos começarão a ser retirados, conforme mostrado aqui:

Tabela de retirada de backup do DynamoDB

Novo mês	Dia 1	Dia 2	Dia 3	Dia 4	Dia 5
Total de backups transferidos do mês anterior	300	290	280	270	260

- No primeiro dia, podemos ver 300 backups no valor de USD 60/mês por backup, totalizando USD 18.000 de `TimedBackupStorage-ByteHrs` aplicados. Isso contrasta com o mês anterior, no qual o total do mês inteiro foi de USD 18.600.
- No dia 2, 10 desses backups expirarão e serão retirados. Quando isso ocorrer, a cobrança aplicada por esses backups será ajustada para o uso real em vez do uso presumido. Isso faz esses 10 backups, anteriormente com uma cobrança de USD 600 (10 backups x 30 dias) aplicada no primeiro dia, serem ajustados para USD 20 (10 backups x 1 dia).
- No dia seguinte, o próximo bloco de 10 expirará e será retirado, mudando seu uso de 30 dias para 2 dias, o que reduz a cobrança para USD 40 (10 backups x 2 dias).

A cada dia que passa, veremos que o pico “maior do que o do mês anterior” começa a diminuir. Se ampliarmos isso para abranger o mês inteiro, observaremos o seguinte:

Progressão das cobranças de backup do DynamoDB (1.º dia do mês)

300 backups em blocos de 10	1.º	10.º	20.º	30.º
Bloco 1	USD 600	USD 20	USD 20	USD 20
Bloco 2	USD 600	USD 40	USD 40	USD 40
Bloco 3	USD 600	USD 60	USD 60	USD 60
Bloco 4	USD 600	USD 80	USD 80	USD 80
Bloco 5	USD 600	100 USD	100 USD	100 USD
Bloco 6	USD 600	USD 120	USD 120	USD 120
Bloco 7	USD 600	USD 140	USD 140	USD 140
Bloco 8	USD 600	USD 160	USD 160	USD 160
Bloco 9	USD 600	USD 180	USD 180	USD 180
Bloco 10	USD 600	USD 600	200 USD	200 USD
Bloco 11	USD 600	USD 600	USD 220	USD 220
Bloco 12	USD 600	USD 600	USD 240	USD 240

300 backups em blocos de 10	1.º	10.º	20.º	30.º
Bloco 13	USD 600	USD 600	USD 260	USD 260
Bloco 14	USD 600	USD 600	USD 280	USD 280
Bloco 15	USD 600	USD 600	$300	$300
Bloco 16	USD 600	USD 600	USD 320	USD 320
Bloco 17	USD 600	USD 600	USD 340	USD 340
Bloco 18	USD 600	USD 600	USD 360	USD 360
Bloco 19	USD 600	USD 600	USD 380	USD 380
Bloco 20	USD 600	USD 600	USD 600	400 USD
Bloco 21	USD 600	USD 600	USD 600	USD 420
Bloco 22	USD 600	USD 600	USD 600	USD 440
Bloco 23	USD 600	USD 600	USD 600	USD 460
Bloco 24	USD 600	USD 600	USD 600	USD 480
Bloco 25	USD 600	USD 600	USD 600	USD 500
Bloco 26	USD 600	USD 600	USD 600	USD 520
Bloco 27	USD 600	USD 600	USD 600	USD 540
Bloco 28	USD 600	USD 600	USD 600	USD 560
Bloco 29	USD 600	USD 600	USD 600	USD 580
Bloco 30	USD 600	USD 600	USD 600	USD 600
Total do 1.º dia do mês (USD)	USD 18.000	USD 13.500	USD 10.400	USD 9.300

À medida que um novo bloco é retirado a cada dia, seu uso é ajustado para quantos dias existiu versus o valor total do mês. Por isso, até o final do mês, as cobranças observadas no dia 1.º terão caído dos USD 18.000 iniciais para os USD 9.300 esperados. Esse número, combinado com os backups recém-criados ao longo do mês (que terão uma tabela de faturamento semelhante à anterior, mas invertida), resultará em uma despesa mensal de acordo com os USD 18.600 do mês anterior.

Restaurar uma tabela do DynamoDB

É possível restaurar uma tabela do DynamoDB por meio do backup da PITR ou dos backups sob demanda usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API do DynamoDB. O processo de recuperação restaura para uma nova tabela do DynamoDB.

Restaurar uma tabela usando a recuperação pontual

É possível restaurar a tabela para qualquer ponto no tempo até o `EarliestRestoreableDateTime`.

> **Important**
>
> Se você desabilitar a recuperação para um ponto no tempo e depois habilitá-la novamente em uma tabela, redefinirá a hora de início para a qual pode recuperar essa tabela. Dessa forma, você só pode restaurar imediatamente essa tabela usando a tabela `LatestRestorableDateTime`.

Quando você restaura usando a recuperação para um ponto no tempo, o DynamoDB restaura os dados da tabela para o estado com base na data e hora selecionadas (day:hour:minute:second) em uma nova tabela. Você restaura uma tabela sem consumir o throughput provisionado na tabela. Você pode fazer uma restauração completa da tabela usando a recuperação pontual ou pode definir as configurações da tabela de destino. Você pode alterar as seguintes configurações de tabela na tabela restaurada:

- Índices secundários globais (GSIs)
- Índices secundários locais (LSIs)
- Modo de faturamento
- Capacidade de leitura e gravação provisionada

- Configurações de criptografia

> **Important**
>
> Ao fazer uma restauração completa da tabela, a tabela de destino é definida com as mesmas unidades de capacidade de leitura e unidades de capacidade de gravação provisionadas da tabela-fonte, conforme gravado no momento em que o backup foi solicitado. Por exemplo, suponha que o throughput provisionado de uma tabela tenha sido reduzido recentemente para 50 unidades de capacidade de leitura e 50 unidades de capacidade de gravação. Você, então, restaura o estado da tabela para três semanas atrás, quando o throughput provisionado estava definido como 100 unidades de capacidade de leitura e 100 unidades de capacidade de gravação. Nesse caso, o DynamoDB restaura os dados da tabela para esse ponto anterior no tempo com o throughput provisionado desse momento (100 unidades de capacidade de leitura e 100 unidades de capacidade de gravação).

Também é possível restaurar os dados da tabela do DynamoDB entre Regiões da AWS para que a tabela restaurada seja criada em uma região diferente daquela na qual a tabela de origem reside. É possível fazer restaurações entre regiões comerciais da AWS, regiões da AWS na China e AWS GovCloud (US). Você paga somente pelos dados transferidos para fora da região de origem e pela restauração para uma nova tabela na região de destino.

> **Note**
>
> A restauração entre regiões não funcionará se a região de origem ou de destino for a Ásia-Pacífico (Hong Kong) ou o Oriente Médio (Bahrein).

As restaurações poderão ser mais rápidas e econômicas se você excluir a criação de alguns ou todos os índices na tabela restaurada. Você deve configurar manualmente os itens a seguir na tabela restaurada:

- Políticas de Auto Scaling
- Políticas do AWS Identity and Access Management
- Métricas e alarmes do Amazon CloudWatch Events
- Tags

- Configurações de fluxo
- Configurações de vida útil (TTL)
- Configurações de recuperação pontual

O tempo necessário para restaurar uma tabela é baseado em vários fatores e nem sempre está correlacionado com o tamanho da tabela.

Restaurar uma tabela do DynamoDB para um ponto no tempo

A recuperação em um ponto anterior no tempo (PITR) do Amazon DynamoDB fornece backups contínuos dos dados de tabelas do DynamoDB. Você pode restaurar uma tabela para um ponto no tempo usando o console do DynamoDB ou a AWS Command Line Interface (AWS CLI). O processo de recuperação em um ponto anterior no tempo sempre restaura uma nova tabela.

Se desejar usar a AWS CLI, você precisará configurá-la primeiro. Para ter mais informações, consulte Acessar o DynamoDB.

Tópicos

- Restaurar uma tabela do DynamoDB table para um ponto no tempo (console)
- Restaurar uma tabela para um ponto no tempo (AWS CLI)

Restaurar uma tabela do DynamoDB table para um ponto no tempo (console)

O exemplo a seguir demonstra como usar o console do DynamoDB para restaurar uma tabela existente chamada `Music` para um ponto no tempo.

> **Note**
>
> Esse procedimento supõe que você habilitou a recuperação em um ponto anterior no tempo. Para habilitá-lo para a tabela de `Music`, na guia de Backups, na seção de Recuperação em um ponto anterior no tempo (Point-in-time recovery – PITR), escolha Edit (Editar) e depois marque a caixa de seleção ao lado de Enable point-in-time recovery (Habilitar recuperação em um ponto anterior no tempo).

Para restaurar uma tabela para um ponto no tempo

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Na lista de tabelas, escolha a tabela `Music`.
4. Na guia Backups da tabela de `Music`, na seção Recuperação em um ponto anterior no tempo (Point-in-time recovery – PITR), escolha Restore (Restaurar).
5. Insira **`MusicMinutesAgo`** como o novo nome da tabela.

   > **Note**
   >
   > Você pode restaurar a tabela para a mesma região da AWS ou para uma região diferente daquela na qual a tabela de origem reside. Você também pode excluir a criação de índices secundários na tabela restaurada. Além disso, você pode especificar um modo de criptografia diferente.

6. Para confirmar a hora restaurável, defina a data e hora da restauração como Earliest (Mais antigo). Em seguida, escolha Restore (Restaurar) para iniciar o processo de restauração.

   A tabela que está sendo restaurada é mostrada com o status Restoring (Em restauração). Quando o processo de restauração for concluído, o status da tabela `MusicMinutesAgo` mudará para Active (Ativo).

Restaurar uma tabela para um ponto no tempo (AWS CLI)

O seguinte procedimento mostra como usar a AWS CLI para restaurar uma tabela existente chamada `Music` para um ponto no tempo.

> **Note**
>
> Esse procedimento supõe que você habilitou a recuperação em um ponto anterior no tempo. Para habilitar esse recurso para a tabela `Music`, execute o seguinte comando.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Para restaurar uma tabela para um ponto no tempo

1. Confirme se a recuperação point-in-time está habilitada para a tabela `Music` usando o comando `describe-continuous-backups`.

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

   Os backups contínuos (habilitados automaticamente na criação da tabela) e a recuperação point-in-time estão habilitados.

   ```
   {
       "ContinuousBackupsDescription": {
           "PointInTimeRecoveryDescription": {
               "PointInTimeRecoveryStatus": "ENABLED",
               "EarliestRestorableDateTime": 1519257118.0,
               "LatestRestorableDateTime": 1520018653.01
           },
           "ContinuousBackupsStatus": "ENABLED"
       }
   }
   ```

2. Restaure a tabela para um ponto. Nesse caso, a tabela `Music` é restaurada para `LatestRestorableDateTime` (aproximadamente 5 minutos atrás) para a mesma região da AWS.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time
   ```

   > **Note**
   >
   > Você também restaurar para um ponto específico. Para isso, execute o comando usando o argumento `--restore-date-time` e especifique um timestamp. É possível especificar qualquer momento dentro do período de recuperação configurado, que pode ser definido para qualquer valor entre 1 e 35 dias. Por exemplo, o comando a seguir restaura a tabela para o `EarliestRestorableDateTime`.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicEarliestRestorableDateTime \
    --no-use-latest-restorable-time \
    --restore-date-time 1519257118.0
```

Especificar o argumento `--no-use-latest-restorable-time` é opcional durante a restauração para um ponto específico.

3. Restaure a tabela para um momento determinado com configurações de tabela personalizadas. Nesse caso, a tabela `Music` é restaurada para o `LatestRestorableDateTime` (há aproximadamente cinco minutos).

   Você pode especificar um modo de criptografia diferente para a tabela restaurada, conforme mostrado a seguir.

   > **Note**
   >
   > O parâmetro `sse-specification-override` assume os mesmos valores do parâmetro `sse-specification-override` usado no comando `CreateTable`. Para saber mais, consulte Gerenciar tabelas criptografadas no DynamoDB.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time \
       --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
   abcd-1234-a123-ab1234a1b234
   ```

   Você pode restaurar a tabela para outra região da AWS diferente daquela na qual a tabela de origem reside.

   > **Note**
   >
   > - O parâmetro `sse-specification-override` é obrigatório para restaurações entre regiões, mas opcional para restaurações para a mesma região da tabela de origem.

- O parâmetro `source-table-arn` deve ser fornecido para restaurações entre regiões.
- Ao realizar uma restauração entre regiões na linha de comando, você deve definir a região padrão da AWS como a região de destino desejada. Para saber mais, consulte Opções de linha de comando no Guia do usuário do AWS Command Line Interface.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Você pode substituir o modo de faturamento e o throughput provisionado para a tabela restaurada.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --billing-mode-override PAY_PER_REQUEST
```

Você pode excluir a criação de alguns ou todos os índices secundários na tabela restaurada.

> **Note**
>
> As restaurações poderão ser mais rápidas e econômicas se você excluir a criação de alguns ou todos os índices secundários na nova tabela restaurada.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

Você pode usar uma combinação de diferentes substituições. Por exemplo, você pode usar um único índice secundário global e alterar o throughput provisionado ao mesmo tempo, da seguinte forma.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --billing-mode-override PROVISIONED \
    --provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100
 \
    --global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEY
 \
    --sse-specification-override Enabled=true,SSEType=KMS \
    --use-latest-restorable-time
```

Para verificar a restauração, use o comando `describe-table` para descrever a tabela `MusicEarliestRestorableDateTime`.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

A tabela que está sendo restaurada é mostrada com o status Creating (Em criação) e restauração em andamento como true (verdadeiro). Quando o processo de restauração for concluído, o status da tabela `MusicEarliestRestorableDateTime` mudará para Active (Ativo).

> **Important**
>
> Enquanto uma restauração estiver em andamento, não modifique nem exclua as políticas do AWS Identity and Access Management (IAM) que concedem à entidade do IAM (por exemplo, usuário, grupo ou função) permissão para realizar a restauração. Do contrário, pode haver um comportamento inesperado. Por exemplo, suponha que você tenha removido as permissões de gravação de uma tabela enquanto essa tabela estava sendo restaurada. Nesse caso, a operação `RestoreTableToPointInTime` subjacente não conseguirá gravar na tabela nenhum dos dados restaurados. Da mesma forma, as políticas do IAM que envolvem restrições de IP de origem para acessar a tabela de restauração de destino podem causar problemas.

Você pode modificar ou excluir permissões somente depois que a operação de restauração é concluída.

Como usar a AWS Backup com o DynamoDB

O Amazon DynamoDB pode ajudar você a atender aos requisitos de conformidade regulatória e continuidade de negócios por meio de recursos aprimorados de backup em AWS Backup. O AWS Backup é um serviço de proteção de dados totalmente gerenciado que facilita a centralização e a automação de backups por serviços da AWS, na nuvem e on-premises. Usando este serviço, você pode configurar políticas de backup e monitorar a atividade para seus recursos de AWS em um só lugar. Para usar o AWS Backup, você deve concordar em optar pela adoção. As opções de aceitação se aplicam à conta específica e região de AWS, então pode ser necessário optar por várias regiões usando a mesma conta. Para obter mais informações, consulte o Guia do Desenvolvedor de Backup do AWS.

O Amazon DynamoDB é integrado nativamente com o AWS Backup. Use o AWS Backup para agendar, copiar, etiquetar e determinar o ciclo de vida de seus backups sob demanda do DynamoDB automaticamente. Você pode continuar visualizando e restaurando esses backups a partir do console do DynamoDB. Você pode usar o console, a API e a AWSInterface de linha de comando (AWSCLI) do DynamoDB para habilitar backups automáticos para suas tabelas do DynamoDB.

Note

Todos os backups feitos pelo DynamoDB permanecerão inalterados. Você ainda poderá criar backups por meio do fluxo de trabalho atual do DynamoDB.

Recursos de backup aprimorados disponíveis por meio da inclusão do AWS Backup:

Backups agendados: configure backups agendados de suas tabelas do DynamoDB usando planos de backup.

Cópia entre contas e regiões: copie automaticamente seus backups para outro cofre de backup em uma região ou conta de AWS diferente, o que permite que você atenda aos requisitos de proteção de dados.

Armazenamento de baixa atividade em camadas: configure seus backups para implementar regras de ciclo de vida de modo a excluir backups ou fazer sua transição para armazenamento de menor atividade. Isso pode ajudar você a otimizar seus custos de backup.

Etiquetas: faça marcação automática de seus backups para fins de cobrança e alocação de custos.

Criptografia: backups sob demanda do DynamoDB gerenciados por meio de AWS Backup serão armazenados no cofre AWS Backup. Isso permite criptografar e proteger seus backups usando um AWS KMS key que é independente da chave de criptografia da tabela do DynamoDB.

Backups de auditoria: use o AWS Backup Audit Manager para auditar a conformidade das suas políticas de AWS Backup e para encontrar atividades de backup e recursos que ainda não estão compatíveis com os controles definidos. Você também pode usá-lo para gerar automaticamente uma trilha de auditoria de relatórios diários e sob demanda para fins de governança de backup.

Backups seguros usando o modelo WORM: use o AWS Backup Vault Lock para habilitar uma configuração de gravação única e várias leituras (write-once-read-many – WORM) para seus backups. Com a Trava de Segurança do AWS Backup, é possível adicionar uma camada extra de defesa que protege os backups contra operações acidentais ou mal-intencionadas de exclusão, alteração nos períodos de recuperação de backup e atualização nas configurações do ciclo de vida. Saiba mais: AWS Backup Vault Lock.

Esses recursos de backup aprimorados estão disponíveis em todas as regiões de AWS. Saiba mais sobre esses recursos: AWS Backup Guia do desenvolvedor.

Tópicos

- Backup e restauração de tabelas do DynamoDB com o AWS Backup: como funciona
- Criar backups de tabelas do DynamoDB com o AWS Backup
- Copiar um backup de uma tabela do DynamoDB com o AWS Backup
- Restaurar um backup de uma tabela do DynamoDB do AWS Backup
- Excluir um backup de uma tabela do DynamoDB com o AWS Backup
- Observação de uso: diferenças entre os backups sob demanda gerenciados pelo AWS Backup e DynamoDB

Backup e restauração de tabelas do DynamoDB com o AWS Backup: como funciona

Você pode usar o recurso de backup sob demanda para criar backups completos de suas tabelas do Amazon DynamoDB. Esta seção oferece uma visão geral do que ocorre durante o processo de backup e restauração.

Backups

Ao criar um backup sob demanda com o AWS Backup, um marcador de tempo da solicitação é catalogado. O backup é criado de forma assíncrona aplicando todas as alterações desde o momento da solicitação até o último snapshot da tabela completa.

Toda vez que você criar um backup sob demanda, será feito backup de todos os dados da tabela. Não há limite para o número de backups sob demanda que podem ser realizados.

> **Note**
>
> Ao contrário dos backups do DynamoDB, backups feitos com o AWS Backup não são instantâneos.

Enquanto um backup estiver em andamento, você não poderá fazer o seguinte:

- Pausar ou cancelar a operação de backup.
- Excluir a tabela de origem do backup.
- Desativar backups em uma tabela se houver um backup em andamento para essa tabela.

O AWS Backup fornece cronogramas de backup automatizados, gerenciamento de retenção e gerenciamento do ciclo de vida. Isso elimina a necessidade de scripts personalizados e processos manuais. O AWS Backup executa os backups e os exclui quando eles expiram. Para mais informações, consulte o Guia do desenvolvedor do AWS Backup.

Se estiver usando o console, todo backup criado usando o AWS Backup será listado na guia Backups com o Backup type (Tipo de backup) definido como `AWS_BACKUP`.

> **Note**
>
> Você não pode excluir backups marcados com um Backup type (Tipo de backup) `AWS_BACKUP` usando o console do DynamoDB. Para gerenciar esses backups, use o console do AWS Backup.

Para saber como realizar um backup, consulte Fazer backup de uma tabela do DynamoDB.

Restaurações

Você restaura uma tabela sem consumir o throughput provisionado na tabela. Você pode fazer uma restauração completa da tabela usando o backup do DynamoDB ou pode definir as configurações da tabela de destino. Ao fazer uma restauração, você pode alterar as seguintes configurações de tabela:

- Índices secundários globais (GSIs)
- Índices secundários locais (LSIs)
- Modo de faturamento
- Capacidade de leitura e gravação provisionada
- Configurações de criptografia

> **Important**
>
> Ao fazer uma restauração completa da tabela, a tabela de destino é definida com as mesmas unidades de capacidade de leitura e unidades de capacidade de gravação provisionadas da tabela-fonte, conforme gravado no momento em que o backup foi solicitado. O processo de restauração também restaura os índices secundários locais e os índices secundários globais.

Copie um backup dos dados da tabela do DynamoDB para outra região de AWS e, em seguida, restaure-o nessa nova região. Você pode fazer restaurações entre regiões comerciais da AWS, regiões da China de AWS e regiões da AWS GovCloud (EUA). Pague somente pelos dados transferidos para fora da região de origem e pelos dados restaurados na nova tabela na região de destino.

O AWS Backup restaurará as tabelas com todos os índices originais.

Você deve configurar manualmente os itens a seguir na tabela restaurada:

- Políticas de Auto Scaling
- Políticas do AWS Identity and Access Management (IAM)
- Métricas e alarmes do Amazon CloudWatch
- Tags
- Configurações de fluxo
- Configurações de vida útil (TTL)
- Configurações de proteção contra exclusão
- Configurações de recuperação para um ponto no tempo (PITR)

Só é possível restaurar os dados completos da tabela para uma nova tabela por meio de backup. Você pode gravar na tabela restaurada somente depois que ela fica ativa.

> **Note**
>
> As restaurações do AWS Backup não são destrutivas. Você não pode substituir uma tabela existente durante uma operação de restauração.

As métricas de serviço mostram que 95% das restaurações da tabela dos clientes são concluídas em menos de uma hora. No entanto, os tempos de restauração estão diretamente relacionados à configuração das tabelas (como o tamanho das tabelas e o número de partições subjacentes) e outras variáveis relacionadas. Uma prática recomendada ao planejar a recuperação de desastres é documentar regularmente os tempos médios de conclusão da restauração e estabelecer como esses tempos afetam seu objetivo geral de tempo de recuperação.

Para saber como realizar uma restauração, consulte Restaurar uma tabela do DynamoDB de um backup.

Você pode usar políticas do IAM para controle de acesso. Para ter mais informações, consulte Usar o IAM com backup e restauração do DynamoDB.

Todas as ações de backup e restauração de console e API são capturadas e registradas no AWS CloudTrail para registro, monitoramento contínuo e auditoria.

Criar backups de tabelas do DynamoDB com o AWS Backup

Esta seção descreve como ativar o AWS Backup para criar backups sob demanda e programados com base em tabelas do DynamoDB.

Tópicos

- Ativar recursos do AWS Backup
- Backups sob demanda
- Backups agendados

Ativar recursos do AWS Backup

É necessário ativar o AWS Backup para usá-lo com o DynamoDB.

Para ativar o AWS Backup, siga estas etapas:

1. Inicie a sessão no Console de gerenciamento da AWS e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. Na janela “Configurações de backup”, escolha Ativar.
4. Uma tela de confirmação será exibida. Escolha Ativar recursos.

Os recursos do AWS Backup ficarão disponíveis para suas tabelas do DynamoDB.

Se você optar por desativar os recursos do AWS Backup depois de ativados, siga estas etapas:

1. Inicie a sessão no Console de gerenciamento da AWS e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. Na janela “Configurações de backup”, escolha Desativar.
4. Uma tela de confirmação será exibida. Escolha Desativar recursos.

Se você não conseguir ativar ou desativar os recursos do AWS Backup, o administrador da AWS precisará realizar essas ações.

Backups sob demanda

Para criar um backup sob demanda de uma tabela do DynamoDB, siga estas etapas:

1. Inicie a sessão no Console de gerenciamento da AWS e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. Escolha Create backup.
4. No menu suspenso exibido, escolha Create a on-demand backup (Criar um backup sob demanda).
5. Para criar um backup gerenciado pelo AWS Backup com armazenamento warm e outros recursos básicos, escolha Default Settings (Configurações padrão). Para criar um backup que possa ser transferido para armazenamento cold ou para criar um backup com recursos do DynamoDB em vez do AWS Backup Backup, escolha Customize Settings (Personalizar as configurações).

   Se você deseja criar esse backup com os recursos anteriores do DynamoDB, selecione Customize settings (Personalizar as configurações) e, depois, selecione Backup with DynamoDB (Backup com o DynamoDB).
6. Quando você concluir as configurações, escolha Create backup (Criar backup).

Backups agendados

Para agendar um backup, siga estas etapas.

1. Inicie a sessão no Console de gerenciamento da AWS e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. No menu suspenso exibido, escolha Agendar backups com o AWS Backup.
4. Você será levado ao AWS Backup Backup para criar um plano de backup.

Copiar um backup de uma tabela do DynamoDB com o AWS Backup

Você pode fazer uma cópia de um backup atual. Você pode copiar backups em várias contas da AWS ou regiões da AWS sob demanda ou automaticamente como parte de um plano de backup programado. Você também pode automatizar uma sequência de cópias entre contas e entre regiões para o Amazon DynamoDB Encryption Client.

A replicação entre regiões é particularmente vantajosa se você tiver requisitos de continuidade de negócios ou de conformidade para armazenar backups a uma distância mínima dos dados de produção.

Backups entre contas são úteis para copiar com segurança seus backups para uma ou mais contas do AWS na sua organização por motivos operacionais ou de segurança. Se o backup original for excluído acidentalmente, você poderá copiar o backup da conta de destino para a conta de origem e, em seguida, iniciar a restauração. Antes de fazer isso, é preciso ter duas contas que pertençam à mesma organização no serviço Organizations.

As cópias herdam a configuração do backup de origem, a menos que você especifique o contrário, com uma exceção: se você especificar que sua nova cópia “Nunca” expirará. Com essa configuração, a nova cópia ainda herda a data de validade da fonte. Se você quiser que sua nova cópia de backup seja permanente, defina seus backups de origem para nunca expirar ou então determine que sua nova cópia expirará 100 anos após a criação.

> **Note**
>
> Se você estiver copiando para outra conta, primeiro você deve ter permissão dessa conta.

Para copiar um backup, faça o seguinte:

1. Inicie a sessão no Console de gerenciamento da AWS e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. Marque a caixa de seleção ao lado do backup que você deseja copiar.
   - Se o backup que você deseja copiar estiver esmaecido, habilite os recursos avançados com AWS Backup. Em seguida, crie um novo backup. Agora você pode copiar esse novo backup para outras regiões e contas e copiar quaisquer outros novos backups daqui para a frente.
4. Escolha Copiar.
5. Se você deseja copiar o backup para outra conta ou região, marque a caixa de seleção ao lado de Copy the recovery point to another destination (Copie o ponto de recuperação para outro destino). Em seguida, escolha entre copiar para outra região em sua conta ou para uma conta diferente em uma região diferente.

> **Note**
>
> Para restaurar um backup para outra região ou conta, primeiro você deve copiar o backup para essa região ou conta.

6. Selecione o cofre desejado para o qual o arquivo será copiado. Se desejar, é possível criar um novo cofre de backup.
7. Selecione Copy backup (Copiar backup).

Restaurar um backup de uma tabela do DynamoDB do AWS Backup

Esta seção descreve como restaurar um backup de uma tabela do DynamoDB do AWS Backup.

Tópicos

- Restaurar uma tabela do DynamoDB do AWS Backup
- Restaurar uma tabela do DynamoDB em outra região ou conta

Restaurar uma tabela do DynamoDB do AWS Backup

Para restaurar suas tabelas do DynamoDB do AWS Backup, siga estas etapas:

1. Inicie a sessão no Console de Gerenciamento da AWS e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Escolha a guia Backups.
4. Marque a caixa de seleção ao lado do backup anterior, do qual você deseja restaurar.
5. Escolha Restore. Você será direcionado para a tela Restore table from backup (Restaurar tabela do backup).
6. Digite o nome da tabela recém-restaurada, a criptografia que esta nova tabela terá, a chave com a qual você deseja que a restauração seja criptografada e outras opções.
7. Quando terminar, escolha Restore (Restaurar).

Restaurar uma tabela do DynamoDB em outra região ou conta

Para restaurar uma tabela do DynamoDB para outra região ou conta, primeiro será necessário copiar o backup para essa nova região ou conta. Para copiar para outra conta, essa conta deve primeiro conceder permissão. Depois de copiar o backup do DynamoDB para a nova região ou conta, ele poderá ser restaurado com o processo da seção anterior.

Excluir um backup de uma tabela do DynamoDB com o AWS Backup

Esta seção descreve como excluir um backup de uma tabela do DynamoDB com o AWS Backup.

Um backup do DynamoDB criado por meio dos recursos do AWS Backup é armazenado em um cofre de backup do AWS.

Para excluir esse tipo de backup, faça o seguinte:

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Backups.
3. Na tela a seguir, selecione Prosseguir com o AWS Backup.

   O Console do AWS Backup será aberto. Para saber mais sobre como excluir backups no Console do AWS Backup, consulte Deleting backups.

   Para ter mais informações sobre AWS Backup, consulte Backup and recovery using AWS Backup em AWS Prescriptive Guidance.

Observação de uso: diferenças entre os backups sob demanda gerenciados pelo AWS Backup e DynamoDB

Esta seção descreve as diferenças técnicas entre backups sob demanda gerenciados pelo AWS Backup e pelo DynamoDB.

O AWS Backup tem alguns fluxos de trabalho e comportamentos diferentes do DynamoDB. Isso inclui:

Criptografia: backups criados com o AWS Backup são armazenados em um cofre criptografado com uma chave gerenciada pelo serviço do AWS Backup. O cofre tem políticas de controle de acesso para segurança adicional.

ARN de Backup: os arquivos de backup criados pelo AWS Backup têm um AWS Backup ARN, que pode afetar o modelo de permissão do usuário. Os Backup resource names (Nomes de recursos de Backup – ARNs) serão alterados de `arn:aws:dynamodb` para `arn:aws:backup`.

Exclusão de backups: backups criados com o AWS Backup só podem ser excluído do cofre do AWS Backup. Você não poderá excluir arquivos do AWS Backup a partir do console do DynamoDB.

Processo de backup: ao contrário dos backups do DynamoDB, backups feitos com o AWS Backup não são instantâneos.

Faturamento: backups de tabelas do DynamoDB com recursos do AWS Backup são cobrados a partir do AWS Backup.

Funções do IAM: se você estiver gerenciando o acesso por meio de funções do IAM, também precisará configurar uma nova função do IAM com estas novas permissões:

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

O `dynamodb:StartAwsBackupJob` é necessário para um backup bem-sucedido com recursos do AWS Backup, e o `dynamodb:RestoreTableFromAwsBackup` é necessário para restaurar com base em um backup feito com recursos do AWS Backup.

Para ver essas permissões em uma política completa do IAM, consulte o Exemplo 8 em Using IAM (Uso de IAM).

Exemplos de código do DynamoDB usando AWS SDKs

Os exemplos de código a seguir mostram como usar o DynamoDB com um Kit de Desenvolvimento de Software (SDK) da AWS.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Contribuições da comunidade da AWS são exemplos criados e mantidos por várias equipes na AWS. Para deixar seu feedback, use o mecanismo fornecido nos repositórios vinculados.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Conceitos básicos

Olá, DynamoDB

O exemplo de código a seguir mostra como começar a usar o DynamoDB.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
```

```
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
```

```
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Código para o arquivo CMakeLists.txt do CMake.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
```

```
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Código para o arquivo de origem hello_dynamodb.cpp.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>
```

```
/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());
```

```
        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API do AWS SDK para C ++.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
```

```
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
```

```
        }
        System.out.println("\nDone!");
    }
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Para ter mais detalhes sobre como trabalhar com o DynamoDB no AWS SDK para JavaScript, consulte Programming DynamoDB with JavaScript.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end

    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
```

```
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Ruby.

Exemplos de código

- Exemplos básicos do DynamoDB usando SDKs da AWS
  - Olá, DynamoDB
  - Conheça os conceitos básicos do DynamoDB com um SDK da AWS
  - Ações do DynamoDB usando AWS SDKs
    - Usar BatchExecuteStatement com um SDK da AWS
    - Usar BatchGetItem com o AWS SDK ou a CLI
    - Usar BatchWriteItem com o AWS SDK ou a CLI
    - Usar CreateTable com o AWS SDK ou a CLI
    - Usar DeleteItem com o AWS SDK ou a CLI
    - Usar DeleteTable com o AWS SDK ou a CLI
    - Usar DescribeTable com o AWS SDK ou a CLI
    - Usar DescribeTimeToLive com o AWS SDK ou a CLI
    - Usar ExecuteStatement com um SDK da AWS
    - Usar GetItem com o AWS SDK ou a CLI
    - Usar ListTables com o AWS SDK ou a CLI
    - Usar PutItem com o AWS SDK ou a CLI
    - Usar Query com o AWS SDK ou a CLI
    - Usar Scan com o AWS SDK ou a CLI
    - Usar UpdateItem com o AWS SDK ou a CLI
    - Usar UpdateTable com o AWS SDK ou a CLI
    - Usar UpdateTimeToLive com o AWS SDK ou a CLI

- Cenários do DynamoDB usando AWS SDKs
  - Acelerar as leituras do DynamoDB com o DAX usando um AWS SDK
  - Trabalhar com cenários avançados do índice secundário global do DynamoDB usando a AWS Command Line Interface v2
  - Criar uma aplicação para enviar dados para uma tabela do DynamoDB
  - Comparar vários valores com um único atributo no DynamoDB com um SDK da AWS
  - Atualizar condicionalmente um item do DynamoDB com TTL usando um SDK da AWS
  - Conecte-se a uma instância local do DynamoDB usando um SDK da AWS
  - Contar operadores de expressão no DynamoDB com um SDK da AWS
  - Criar uma API REST do API Gateway para monitorar dados da COVID-19
  - Criar uma aplicação de mensageiro com o Step Functions
  - Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos
  - Crie uma tabela do DynamoDB com um índice secundário global usando o SDK da AWS.
  - Criar uma tabela do DynamoDB com uma configuração de throughput a quente usando um AWS SDK
  - Criar uma aplicação Web para monitorar dados do DynamoDB
  - Criar uma aplicação de chat websocket com o API Gateway
  - Criar um item do DynamoDB com TTL usando um SDK da AWS
  - Crie e gerencie tabelas globais do DynamoDB com consistência forte multirregional usando um SDK da AWS.
  - Crie e gerencie tabelas globais do DynamoDB demonstrando o MREC usando um SDK da AWS
  - Excluir dados do DynamoDB usando declarações DELETE do PartiQL com um SDK da AWS
  - Detectar EPI em imagens com o Amazon Rekognition usando um AWS SDK
  - Inserir dados do DynamoDB usando declarações INSERT do PartiQL com um SDK da AWS
  - Chamar uma função do Lambda em um navegador
  - Gerenciar índices secundários globais no DynamoDB usando a AWS Command Line Interface v2
  - Gerenciar políticas baseadas em recursos do DynamoDB usando a AWS Command Line Interface v2
  - Monitorar a performance do Amazon DynamoDB usando um SDK da AWS

- Executar operações de consulta avançada no DynamoDB usando um SDK da AWS
- Executar operações de lista no DynamoDB com um SDK da AWS
- Executar operações de mapa no DynamoDB com um SDK da AWS
- Executar operações de conjunto no DynamoDB com um SDK da AWS
- Consultar uma tabela do DynamoDB usando lotes de instruções PartiQL e um AWS SDK
- Consultar uma tabela do DynamoDB usando o PartiQL e um AWS SDK
- Consultar uma tabela do DynamoDB usando um índice secundário global por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB usando uma condição begins_with por meio de um SDK da AWS
- Consulte uma tabela do DynamoDB usando um intervalo de datas na chave de classificação por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com uma expressão de filtro complexa por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com uma expressão de filtro dinâmica por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com uma expressão de filtro e limite por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com atributos aninhados por meio de um SDK da AWS
- Consultar uma tabela no DynamoDB com paginação usando um SDK da AWS
- Consultar uma tabela do DynamoDB com leituras altamente consistentes usando um SDK da AWS
- Consultar dados do DynamoDB usando declarações SELECT do PartiQL com um SDK da AWS
- Consultar itens com TTL em uma tabela do DynamoDB usando um SDK da AWS
- Consultar tabelas do DynamoDB usando padrões de data e hora com um SDK da AWS
- Salvar o EXIF e outras informações de imagem usando um AWS SDK
- Configurar o controle de acesso por atributo para o DynamoDB usando a AWS Command Line Interface v2
- Com funciona a ordem da expressão de atualização no DynamoDB com um SDK da AWS
- Atualizar uma configuração de tabela do DynamoDB com throughput a quente usando um AWS SDK
- Atualizar um item do DynamoDB com TTL usando um SDK da AWS

- Atualizar dados do DynamoDB usando declarações UPDATE do PartiQL com um SDK da AWS
- Usar o API Gateway para invocar uma função do Lambda
- Usar Step Functions para invocar funções do Lambda
- Usar um modelo de documento para o DynamoDB usando um AWS SDK
- Usar um modelo de persistência de objetos de alto nível para o DynamoDB usando um AWS SDK
- Usar operações de contador atômico no DynamoDB com um SDK da AWS
- Usar operações condicionais no DynamoDB com um SDK da AWS
- Usar nomes de atributo de expressão no DynamoDB com um SDK da AWS
- Usar eventos programados para invocar uma função do Lambda
- Trabalhar com índices secundários locais do DynamoDB usando a AWS Command Line Interface v2
- Trabalhar com fluxos e vida útil do DynamoDB usando a AWS Command Line Interface v2
- Trabalhar com tabelas globais do DynamoDB e a consistência final de replicação mutirregional (MREC) usando AWS Command Line Interface v2
- Trabalhar com a marcação de recursos do DynamoDB usando a AWS Command Line Interface v2
- Trabalhar com a criptografia de tabelas do DynamoDB usando a AWS Command Line Interface v2

- Exemplos de tecnologia sem servidor para o DynamoDB
  - Invocar uma função do Lambda em um gatilho do DynamoDB
  - Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB
- Contribuições da comunidade da AWS para o DynamoDB
  - Compilar e testar uma aplicação com tecnologia sem servidor

Exemplos básicos do DynamoDB usando SDKs da AWS

Os exemplos de código a seguir mostram como usar os conceitos básicos do Amazon DynamoDB com SDKs da AWS.

Exemplos

- Olá, DynamoDB

- Conheça os conceitos básicos do DynamoDB com um SDK da AWS
- Ações do DynamoDB usando AWS SDKs
  - Usar BatchExecuteStatement com um SDK da AWS
  - Usar BatchGetItem com o AWS SDK ou a CLI
  - Usar BatchWriteItem com o AWS SDK ou a CLI
  - Usar CreateTable com o AWS SDK ou a CLI
  - Usar DeleteItem com o AWS SDK ou a CLI
  - Usar DeleteTable com o AWS SDK ou a CLI
  - Usar DescribeTable com o AWS SDK ou a CLI
  - Usar DescribeTimeToLive com o AWS SDK ou a CLI
  - Usar ExecuteStatement com um SDK da AWS
  - Usar GetItem com o AWS SDK ou a CLI
  - Usar ListTables com o AWS SDK ou a CLI
  - Usar PutItem com o AWS SDK ou a CLI
  - Usar Query com o AWS SDK ou a CLI
  - Usar Scan com o AWS SDK ou a CLI
  - Usar UpdateItem com o AWS SDK ou a CLI
  - Usar UpdateTable com o AWS SDK ou a CLI
  - Usar UpdateTimeToLive com o AWS SDK ou a CLI

Olá, DynamoDB

O exemplo de código a seguir mostra como começar a usar o DynamoDB.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }
```

```
            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Código para o arquivo CMakeLists.txt do CMake.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)
```

```
# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                 # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Código para o arquivo de origem hello_dynamodb.cpp.

```
#include <aws/core/Aws.h>
```

```
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }
```

```
            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API do AWS SDK para C ++.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
```

```
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }
```

```
            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Para ter mais detalhes sobre como trabalhar com o DynamoDB no AWS SDK para JavaScript, consulte Programming DynamoDB with JavaScript.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end

    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
```

```
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Ruby.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Conheça os conceitos básicos do DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como:

- Criar uma tabela que possa conter dados de filmes.
- Colocar, obter e atualizar um único filme na tabela.
- Gravar dados de filmes na tabela usando um arquivo JSON de exemplo.
- Consultar filmes que foram lançados em determinado ano.
- Verificar filmes que foram lançados em um intervalo de anos.
- Exclua um filme da tabela e, depois, exclua a tabela.

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper =
 host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();
```

```
        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous"
+
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
```

```
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie:
{newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
```

```
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName,
movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
```

```
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is
complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using
DynamoDB with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in
a given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released
within a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
created.");
        WaitForEnter();
```

```
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

Cria uma tabela para conter dados de filmes.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
```

```
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
```

```
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

Adiciona um único filme à tabela.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
```

```
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

Atualiza um item único em uma tabela.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
```

```
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} or item was not found.
 {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
 {ex.Message}");
            throw;
        }
    }
```

Recupera um único item de uma tabela de filmes.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
```

```
            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

Grava um lote de itens na tabela de filmes.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
```

```
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
```

```
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Exclui um único item da tabela.

```
    /// <summary>
```

```
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
```

```
    }
```

Consulta a tabela de filmes lançados em determinado ano.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
            Filter = filter,
        };

        // Value used to track how many movies match the
        // supplied criteria.
```

```
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

Busca na tabela os filmes lançados em um intervalo de anos.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
```

```
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
 info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
 LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
```

```
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
 table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
 {ex.Message}");
            throw;
        }
    }
```

Exclui a tabela de filmes.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);
```

```
            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

O cenário de conceitos básicos do DynamoDB.

```
###############################################################################
# function dynamodb_getting_started_movies
#
# Scenario to create an Amazon DynamoDB table and perform a series of operations
 on the table.
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
###############################################################################
function dynamodb_getting_started_movies() {

  source ./dynamodb_operations.sh

  key_schema_json_file="dynamodb_key_schema.json"
  attribute_definitions_json_file="dynamodb_attr_def.json"
  item_json_file="movie_item.json"
  key_json_file="movie_key.json"
  batch_json_file="batch.json"
  attribute_names_json_file="attribute_names.json"
  attributes_values_json_file="attribute_values.json"

  echo_repeat "*" 88
  echo
  echo "Welcome to the Amazon DynamoDB getting started demo."
  echo
  echo_repeat "*" 88
  echo

  local table_name
  echo -n "Enter a name for a new DynamoDB table: "
```

```
  get_input
  table_name=$get_input_result

  echo '[
  {"AttributeName": "year", "KeyType": "HASH"},
   {"AttributeName": "title", "KeyType": "RANGE"}
  ]' >"$key_schema_json_file"

  echo '[
  {"AttributeName": "year", "AttributeType": "N"},
   {"AttributeName": "title", "AttributeType": "S"}
  ]' >"$attribute_definitions_json_file"

  if dynamodb_create_table -n "$table_name" -a "$attribute_definitions_json_file"
\
    -k "$key_schema_json_file" 1>/dev/null; then
    echo "Created a DynamoDB table named $table_name"
  else
    errecho "The table failed to create. This demo will exit."
    clean_up
    return 1
  fi

  echo "Waiting for the table to become active...."

  if dynamodb_wait_table_active -n "$table_name"; then
    echo "The table is now active."
  else
    errecho "The table failed to become active. This demo will exit."
    cleanup "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo -n "Enter the title of a movie you want to add to the table: "
  get_input
  local added_title
  added_title=$get_input_result

  local added_year
  get_int_input "What year was it released? "
```

```
  added_year=$get_input_result

  local rating
  get_float_input "On a scale of 1 - 10, how do you rate it? " "1" "10"
  rating=$get_input_result

  local plot
  echo -n "Summarize the plot for me: "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"},
    "info": {"M" : {"plot": {"S" : "'"$plot"'"}, "rating":
{"N" :"'"$rating"'"} } }
  }' >"$item_json_file"

  if dynamodb_put_item -n "$table_name" -i "$item_json_file"; then
    echo "The movie '$added_title' was successfully added to the table
'$table_name'."
  else
    errecho "Put item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "Let's update your movie '$added_title'."
  get_float_input "You rated it $rating, what new rating would you give it? " "1"
"10"
  rating=$get_input_result

  echo -n "You summarized the plot as '$plot'."
  echo "What would you say now? "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"}
```

```
    }' >"$key_json_file"

  echo '{
    ":r": {"N" :"'"$rating"'"},
    ":p": {"S" : "'"$plot"'"}
   }' >"$item_json_file"

  local update_expression="SET info.rating = :r, info.plot = :p"

  if dynamodb_update_item -n "$table_name" -k "$key_json_file" -e
"$update_expression" -v "$item_json_file"; then
    echo "Updated '$added_title' with new attributes."
  else
    errecho "Update item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "We will now use batch write to upload 150 movie entries into the table."

  local batch_json
  for batch_json in movie_files/movies_*.json; do
    echo "{ \"$table_name\" : $(<"$batch_json") }" >"$batch_json_file"
    if dynamodb_batch_write_item -i "$batch_json_file" 1>/dev/null; then
      echo "Entries in $batch_json added to table."
    else
      errecho "Batch write failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  done

  local title="The Lord of the Rings: The Fellowship of the Ring"
  local year="2001"

  if get_yes_no_input "Let's move on...do you want to get info about '$title'?
(y/n) "; then
    echo '{
  "year": {"N" :"'"$year"'"},
  "title": {"S" :  "'"$title"'"}
```

```
  }' >"$key_json_file"
    local info
    info=$(dynamodb_get_item -n "$table_name" -k "$key_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Get item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$info"
  fi

  local ask_for_year=true
  while [[ "$ask_for_year" == true ]]; do
    echo "Let's get a list of movies released in a given year."
    get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
    year=$get_input_result
    echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

    echo '{
    ":v": {"N" :"'"$year"'"}
    }' >"$attributes_values_json_file"

    response=$(dynamodb_query -n "$table_name" -k "#n=:v" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Query table failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$response"

    if ! get_yes_no_input "Try another year? (y/n) "; then
      ask_for_year=false
    fi
```

```
  done

  echo "Now let's scan for movies released in a range of years. Enter a year: "
  get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
  local start=$get_input_result

  get_int_input "Enter another year: " "1972" "2018"
  local end=$get_input_result

  echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

  echo '{
    ":v1": {"N" : "'"$start"'"},
    ":v2": {"N" : "'"$end"'"}
    }' >"$attributes_values_json_file"

  response=$(dynamodb_scan -n "$table_name" -f "#n BETWEEN :v1 AND :v2" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

  # shellcheck disable=SC2181
  if [[ ${?} -ne 0 ]]; then
    errecho "Scan table failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo "Here is what I found:"
  echo "$response"

  echo
  echo_repeat "*" 88
  echo

  echo "Let's remove your movie '$added_title' from the table."

  if get_yes_no_input "Do you want to remove '$added_title'? (y/n) "; then
    echo '{
  "year": {"N" :"'"$added_year"'"},
  "title": {"S" :  "'"$added_title"'"}
  }' >"$key_json_file"

    if ! dynamodb_delete_item -n "$table_name" -k "$key_json_file"; then
```

```
      errecho "Delete item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  fi

  if get_yes_no_input "Do you want to delete the table '$table_name'? (y/n) ";
 then
    if ! clean_up "$table_name"; then
      return 1
    fi
  else
    if ! clean_up; then
      return 1
    fi
  fi

  return 0
}
```

As funções do DynamoDB usadas nesse cenário.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.
```

```
 ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_create_table"
   echo "Creates an Amazon DynamoDB table with on-demand billing."
   echo " -n table_name  -- The name of the table to create."
   echo " -a attribute_definitions -- JSON file path of a list of attributes and
their types."
   echo " -k key_schema -- JSON file path of a list of attributes and their key
types."
   echo ""
 }

 # Retrieve the calling parameters.
 while getopts "n:a:k:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     a) attribute_definitions="${OPTARG}" ;;
     k) key_schema="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_definitions" ]]; then
   errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
   usage
   return 1
```

```
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}

###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
```

```
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}

###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }
```

```
  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
```

```
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }
```

```
  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi
  if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
```

```
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}

########################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
########################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
```

```
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
```

```
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

if [[ -n "$query" ]]; then
  response=$(aws dynamodb get-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --output text \
    --query "$query")
else
  response=$(
    aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text
  )
fi

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports get-item operation failed.$response"
  return 1
fi

if [[ -n "$query" ]]; then
```

```
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}

###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
```

```
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
```

```
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
```

```
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
```

```
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
```

```
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
```

```
  echo ""
}
while getopts "n:k:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    keys:   $keys"
iecho ""

response=$(aws dynamodb delete-item \
  --table-name "$table_name" \
  --key file://"$keys")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

As funções utilitárias usadas nesse cenário.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}
```

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
   {
       Aws::Client::ClientConfiguration clientConfig;
       //  1. Create a table with partition: year (N) and sort: title (S).
(CreateTable)
       if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

           AwsDoc::DynamoDB::dynamodbGettingStartedScenario(clientConfig);

           // 9. Delete the table. (DeleteTable)
           AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
```

```
        }
    }

//! Scenario to modify and query a DynamoDB table.
/*!
  \sa dynamodbGettingStartedScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::dynamodbGettingStartedScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;
    std::cout << "Welcome to the Amazon DynamoDB getting started demo." <<
 std::endl;
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie.
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
 it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::PutItemRequest putItemRequest;
        putItemRequest.SetTableName(MOVIE_TABLE_NAME);

        putItemRequest.AddItem(YEAR_KEY,

 Aws::DynamoDB::Model::AttributeValue().SetN(year));
        putItemRequest.AddItem(TITLE_KEY,

 Aws::DynamoDB::Model::AttributeValue().SetS(title));
```

```
        // Create attribute for the info map.
        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);

        putItemRequest.AddItem(INFO_KEY, infoMapAttribute);

        Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
                putItemRequest);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add an item: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    // 3. Update the rating and plot of the movie by using an update expression.
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);
        plot = askQuestion(Aws::String("You summarized the plot as '") + plot +
                           "'.\nWhat would you say now? ");

        Aws::DynamoDB::Model::UpdateItemRequest request;
        request.SetTableName(MOVIE_TABLE_NAME);
        request.AddKey(TITLE_KEY,
Aws::DynamoDB::Model::AttributeValue().SetS(title));
```

```
        request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
        std::stringstream expressionStream;
        expressionStream << "set " << INFO_KEY << "." << RATING_KEY << " =:r, "
                         << INFO_KEY << "." << PLOT_KEY << " =:p";
        request.SetUpdateExpression(expressionStream.str());
        request.SetExpressionAttributeValues({
                                                 {":r",
Aws::DynamoDB::Model::AttributeValue().SetN(
                                                         rating)},
                                                 {":p",
Aws::DynamoDB::Model::AttributeValue().SetS(
                                                         plot)}
                                             });

        request.SetReturnValues(Aws::DynamoDB::Model::ReturnValue::UPDATED_NEW);

        const Aws::DynamoDB::Model::UpdateItemOutcome &result =
dynamoClient.UpdateItem(
                request);
        if (!result.IsSuccess()) {
            std::cerr << "Error updating movie " + result.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    // 4. Put 250 movies in the table from moviedata.json.
    {
        std::cout << "Adding movies from a json file to the database." <<
std::endl;
        const size_t MAX_SIZE_FOR_BATCH_WRITE = 25;
        const size_t MOVIES_TO_WRITE = 10 * MAX_SIZE_FOR_BATCH_WRITE;
        Aws::String jsonString = getMovieJSON();
        if (!jsonString.empty()) {
            Aws::Utils::Json::JsonValue json(jsonString);
            Aws::Utils::Array<Aws::Utils::Json::JsonView> movieJsons =
json.View().AsArray();
            Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;

            // To add movies with a cross-section of years, use an appropriate
increment
```

```
            // value for iterating through the database.
            size_t increment = movieJsons.GetLength() / MOVIES_TO_WRITE;
            for (size_t i = 0; i < movieJsons.GetLength(); i += increment) {
                writeRequests.push_back(Aws::DynamoDB::Model::WriteRequest());
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
putItems = movieJsonViewToAttributeMap(
                        movieJsons[i]);
                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(putItems);
                writeRequests.back().SetPutRequest(putRequest);
                if (writeRequests.size() == MAX_SIZE_FOR_BATCH_WRITE) {
                    Aws::DynamoDB::Model::BatchWriteItemRequest request;
                    request.AddRequestItems(MOVIE_TABLE_NAME, writeRequests);
                    const Aws::DynamoDB::Model::BatchWriteItemOutcome &outcome =
dynamoClient.BatchWriteItem(
                            request);
                    if (!outcome.IsSuccess()) {
                        std::cerr << "Unable to batch write movie data: "
                                  << outcome.GetError().GetMessage()
                                  << std::endl;
                        writeRequests.clear();
                        break;
                    }
                    else {
                        std::cout << "Added batch of " << writeRequests.size()
                                  << " movies to the database."
                                  << std::endl;
                    }
                    writeRequests.clear();
                }
            }
        }
    }

    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    // 5. Get a movie by Key (partition + sort).
    {
        Aws::String titleToGet("King Kong");
        Aws::String answer = askQuestion(Aws::String(
                "Let's move on...Would you like to get info about '" + titleToGet
+
                "'? (y/n) "));
```

```
        if (answer == "y") {
            Aws::DynamoDB::Model::GetItemRequest request;
            request.SetTableName(MOVIE_TABLE_NAME);
            request.AddKey(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(titleToGet));
            request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(1933));

            const Aws::DynamoDB::Model::GetItemOutcome &result =
dynamoClient.GetItem(
                    request);
            if (!result.IsSuccess()) {
                std::cerr << "Error " << result.GetError().GetMessage();
            }
            else {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = result.GetResult().GetItem();
                if (!item.empty()) {
                    std::cout << "\nHere's what I found:" << std::endl;
                    printMovieInfo(item);
                }
                else {
                    std::cout << "\nThe movie was not found in the database."
                              << std::endl;
                }
            }
        }
    }

    // 6. Use Query with a key condition expression to return all movies
    //    released in a given year.
    Aws::String doAgain = "n";
    do {
        Aws::DynamoDB::Model::QueryRequest req;

        req.SetTableName(MOVIE_TABLE_NAME);

        // "year" is a DynamoDB reserved keyword and must be replaced with an
        // expression attribute name.
        req.SetKeyConditionExpression("#dynobase_year = :valueToMatch");
        req.SetExpressionAttributeNames({{"#dynobase_year", YEAR_KEY}});

        int yearToMatch = askQuestionForIntRange(
```

```
                "\nLet's get a list of movies released in"
                " a given year. Enter a year between 1972 and 2018 ",
                1972, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
        attributeValues.emplace(":valueToMatch",
                                Aws::DynamoDB::Model::AttributeValue().SetN(
                                        yearToMatch));
        req.SetExpressionAttributeValues(attributeValues);

        const Aws::DynamoDB::Model::QueryOutcome &result =
dynamoClient.Query(req);
        if (result.IsSuccess()) {
            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "\nThere were " << items.size()
                          << " movies in the database from "
                          << yearToMatch << "." << std::endl;
                for (const auto &item: items) {
                    printMovieInfo(item);
                }
                doAgain = "n";
            }
            else {
                std::cout << "\nNo movies from " << yearToMatch
                          << " were found in the database"
                          << std::endl;
                doAgain = askQuestion(Aws::String("Try another year? (y/n) "));
            }
        }
        else {
            std::cerr << "Failed to Query items: " <<
result.GetError().GetMessage()
                      << std::endl;
        }

    } while (doAgain == "y");

    //  7. Use Scan to return movies released within a range of years.
    //     Show how to paginate data using ExclusiveStartKey. (Scan +
FilterExpression)
    {
```

```
        int startYear = askQuestionForIntRange("\nNow let's scan a range of years
"
                                               "for movies in the database. Enter
a start year: ",
                                               1972, 2018);
        int endYear = askQuestionForIntRange("\nEnter an end year: ",
                                             startYear, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
        do {
            Aws::DynamoDB::Model::ScanRequest scanRequest;
            scanRequest.SetTableName(MOVIE_TABLE_NAME);
            scanRequest.SetFilterExpression(
                    "#dynobase_year >= :startYear AND #dynobase_year
<= :endYear");
            scanRequest.SetExpressionAttributeNames({{"#dynobase_year",
YEAR_KEY}});

            Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
            attributeValues.emplace(":startYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            startYear));
            attributeValues.emplace(":endYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            endYear));
            scanRequest.SetExpressionAttributeValues(attributeValues);

            if (!exclusiveStartKey.empty()) {
                scanRequest.SetExclusiveStartKey(exclusiveStartKey);
            }

            const Aws::DynamoDB::Model::ScanOutcome &result = dynamoClient.Scan(
                    scanRequest);
            if (result.IsSuccess()) {
                const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
                if (!items.empty()) {
                    std::stringstream stringStream;
                    stringStream << "\nFound " << items.size() << " movies in one
scan."
                                 << " How many would you like to see? ";
                    size_t count = askQuestionForInt(stringStream.str());
                    for (size_t i = 0; i < count && i < items.size(); ++i) {
```

```
                    printMovieInfo(items[i]);
                }
            }
            else {
                std::cout << "\nNo movies in the database between " <<
startYear <<
                          " and " << endYear << "." << std::endl;
            }

            exclusiveStartKey = result.GetResult().GetLastEvaluatedKey();
            if (!exclusiveStartKey.empty()) {
                std::cout << "Not all movies were retrieved. Scanning for
more."
                          << std::endl;
            }
            else {
                std::cout << "All movies were retrieved with this scan."
                          << std::endl;
            }
        }
        else {
            std::cerr << "Failed to Scan movies: "
                      << result.GetError().GetMessage() << std::endl;
        }
    } while (!exclusiveStartKey.empty());
}

// 8. Delete a movie. (DeleteItem)
{
    std::stringstream stringStream;
    stringStream << "\nWould you like to delete the movie " << title
                 << " from the database? (y/n) ";
    Aws::String answer = askQuestion(stringStream.str());
    if (answer == "y") {
        Aws::DynamoDB::Model::DeleteItemRequest request;
        request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.AddKey(TITLE_KEY,
                       Aws::DynamoDB::Model::AttributeValue().SetS(title));
        request.SetTableName(MOVIE_TABLE_NAME);

        const Aws::DynamoDB::Model::DeleteItemOutcome &result =
dynamoClient.DeleteItem(
                    request);
```

```
            if (result.IsSuccess()) {
                std::cout << "\nRemoved \"" << title << "\" from the database."
                          << std::endl;
            }
            else {
                std::cerr << "Failed to delete the movie: "
                          << result.GetError().GetMessage()
                          << std::endl;
            }
        }
    }

    return true;
}

//! Routine to convert a JsonView object to an attribute map.
/*!
  \sa movieJsonViewToAttributeMap()
  \param jsonView: Json view object.
  \return map: Map that can be used in a DynamoDB request.
 */
Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
AwsDoc::DynamoDB::movieJsonViewToAttributeMap(
        const Aws::Utils::Json::JsonView &jsonView) {
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> result;

    if (jsonView.KeyExists(YEAR_KEY)) {
        result[YEAR_KEY].SetN(jsonView.GetInteger(YEAR_KEY));
    }
    if (jsonView.KeyExists(TITLE_KEY)) {
        result[TITLE_KEY].SetS(jsonView.GetString(TITLE_KEY));
    }
    if (jsonView.KeyExists(INFO_KEY)) {
        Aws::Map<Aws::String, const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue>> infoMap;
        Aws::Utils::Json::JsonView infoView = jsonView.GetObject(INFO_KEY);
        if (infoView.KeyExists(RATING_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetN(infoView.GetDouble(RATING_KEY));
            infoMap.emplace(std::make_pair(RATING_KEY, attributeValue));
        }
        if (infoView.KeyExists(PLOT_KEY)) {
```

```
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetS(infoView.GetString(PLOT_KEY));
            infoMap.emplace(std::make_pair(PLOT_KEY, attributeValue));
        }

        result[INFO_KEY].SetM(infoMap);
    }

    return result;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);
```

```
        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
```

```
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
```

```
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para C++.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query

- Scan
- UpdateItem

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute um cenário interativo para criar a tabela e executar ações nela.

```
import (
 "context"
 "fmt"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunMovieScenario is an interactive example that shows you how to use the AWS
 SDK for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//  1. Create a table that can hold movie data.
//  2. Put, get, and update a single movie in the table.
//  3. Write movie data to the table from a sample JSON file.
//  4. Query for movies that were released in a given year.
//  5. Scan for movies that were released in a range of years.
//  6. Delete a movie from the table.
//  7. Delete the table.
//
```

```
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is
 loaded
// into the named table.
func RunMovieScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner,
 tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB getting started demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{TableName: tableName,
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
```

```
var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 demotools.NotEmpty{})
customMovie.Year = questioner.AskInt("What year was it released?",
 demotools.NotEmpty{}, demotools.InIntRange{Lower: 1900, Upper: 2030})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 demotools.NotEmpty{})
err = tableBasics.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
 "What new rating would you give it?",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 demotools.NotEmpty{})
attributes, err := tableBasics.UpdateMovie(ctx, customMovie)
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
   log.Printf("\t%v: %v\n", valKey, val)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(ctx, movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
```

```
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
 demotools.InIntRange{Lower: 1, Upper: show},
)
movie, err := tableBasics.GetMovie(ctx, movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018},
)
releases, err := tableBasics.Query(ctx, releaseYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
endYear := questioner.AskInt("Enter another year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
releases, err = tableBasics.Scan(ctx, startYear, endYear)
if err == nil {
```

```
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n",
startYear, endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables(ctx)
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
   log.Printf("\t%v", table)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
 err = tableBasics.DeleteMovie(ctx, customMovie)
}
if err == nil {
 log.Printf("Deleted %v.\n", customMovie.Title)
}

if questioner.AskBool("Delete the table, too? (y/n)", "y") {
 err = tableBasics.DeleteTable(ctx)
} else {
 log.Println("Don't forget to delete the table when you're done or you might " +
  "incur charges on your account.")
}
if err == nil {
 log.Printf("Deleted table %v.\n", tableBasics.TableName)
}
```

```
 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
```

```
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Crie uma estrutura e métodos que chamem ações do DynamoDB.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
```

```
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
```

```
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
```

```
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
```

```
   }
  }
 }
 return attributeMap, err
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
```

```
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
```

```
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
```

```
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
```

```
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Go.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Crie uma tabela do DynamoDB.

```
// Create a table with a Sort key.
public static void createTable(DynamoDbClient ddb, String tableName) {
    DynamoDbWaiter dbWaiter = ddb.waiter();
    ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

    // Define attributes.
    attributeDefinitions.add(AttributeDefinition.builder()
        .attributeName("year")
        .attributeType("N")
        .build());

    attributeDefinitions.add(AttributeDefinition.builder()
        .attributeName("title")
        .attributeType("S")
        .build());

    ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
    KeySchemaElement key = KeySchemaElement.builder()
        .attributeName("year")
        .keyType(KeyType.HASH)
        .build();

    KeySchemaElement key2 = KeySchemaElement.builder()
        .attributeName("title")
        .keyType(KeyType.RANGE)
        .build();

    // Add KeySchemaElement objects to the list.
    tableKey.add(key);
    tableKey.add(key2);
```

```
        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

Crie uma função auxiliar para baixar e extrair o arquivo JSON de exemplo.

```
    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
            .dynamoDbClient(ddb)
            .build();

        DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
```

```
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {
            // Only add 200 Movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }
```

Obtenha um item de uma tabela.

```
    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
```

```
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

Exemplo completo.

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Creates the Amazon DynamoDB Movie table with partition and sort key.
 * 2. Puts data into the Amazon DynamoDB table from a JSON document using the
 * Enhanced client.
 * 3. Gets data from the Movie table.
 * 4. Adds a new item.
 * 5. Updates an item.
 * 6. Uses a Scan to query items using the Enhanced client.
```

```
 * 7. Queries all items where the year is 2013 using the Enhanced Client.
 * 8. Deletes the table.
 */

public class Scenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        String tableName = "Movies";
        String fileName = "../../../resources/sample_files/movies.json";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon DynamoDB example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
            "1. Creating an Amazon DynamoDB table named Movies with a key named
 year and a sort key named title.");
        createTable(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Loading data into the Amazon DynamoDB table.");
        loadData(ddb, tableName, fileName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Getting data from the Movie table.");
        getItem(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Putting a record into the Amazon DynamoDB
 table.");
        putRecord(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("5. Updating a record.");
        updateTableItem(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Scanning the Amazon DynamoDB table.");
        scanMovies(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Querying the Movies released in 2013.");
        queryTable(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        System.out.println(DASHES);

        ddb.close();
    }

    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();
```

```
        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Query the table.
    public static void queryTable(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
```

```
            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            QueryConditional queryConditional = QueryConditional
                .keyEqualTo(Key.builder()
                    .partitionValue(2013)
                    .build());

            // Get items in the table and write out the ID value.
            Iterator<Movies> results =
custTable.query(queryConditional).items().iterator();
            String result = "";

            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The title of the movie is " +
rec.getTitle());
                System.out.println("The movie information  is " + rec.getInfo());
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Scan the table.
    public static void scanMovies(DynamoDbClient ddb, String tableName) {
        System.out.println("******* Scanning all movies.\n");
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            Iterator<Movies> results = custTable.scan().items().iterator();
            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The movie title is " + rec.getTitle());
                System.out.println("The movie year is " + rec.getYear());
            }

        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
            .dynamoDbClient(ddb)
            .build();

        DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {
            // Only add 200 Movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }

    // Update the record to include show only directors.
    public static void updateTableItem(DynamoDbClient ddb, String tableName) {
        HashMap<String, AttributeValue> itemKey = new HashMap<>();
```

```
        itemKey.put("year", AttributeValue.builder().n("1933").build());
        itemKey.put("title", AttributeValue.builder().s("King Kong").build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("info", AttributeValueUpdate.builder()
            .value(AttributeValue.builder().s("{\"directors\":[\"Merian C. Cooper
\",\"Ernest B. Schoedsack\"]")
                .build())
            .action(AttributeAction.PUT)
            .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(itemKey)
            .attributeUpdates(updatedValues)
            .build();

        try {
            ddb.updateItem(request);
        } catch (ResourceNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        System.out.println("Item was updated!");
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
```

```
    }

    public static void putRecord(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> table = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));

            // Populate the Table.
            Movies record = new Movies();
            record.setYear(2020);
            record.setTitle("My Movie2");
            record.setInfo("no info");
            table.putItem(record);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Added a new movie to the table.");
    }

    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
```

```
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";
```

```
const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });
```

```
  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so
'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
```

```
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/
latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */

  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
```

```
  await client.send(deleteCommand);
  log("Movie deleted.");

  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
```

```
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression.
 Scan will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
```

```
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma tabela do DynamoDB.

```
suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
```

```
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

Crie uma função auxiliar para baixar e extrair o arquivo JSON de exemplo.

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}
```

```
suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

Obtenha um item de uma tabela.

```
suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }
```

```
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

Exemplo completo.

```
suspend fun main() {
    val tableName = "Movies"
    val fileName = "../../../resources/sample_files/movies.json"
    val partitionAlias = "#a"

    println("Creating an Amazon DynamoDB table named Movies with a key named id
 and a sort key named title.")
    createScenarioTable(tableName, "year")
    loadData(tableName, fileName)
    getMovie(tableName, "year", "1933")
    scanMovies(tableName)
    val count = queryMovieTable(tableName, "year", partitionAlias)
    println("There are $count Movies released in 2013.")
    deletIssuesTable(tableName)
}

suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
```

```
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}

// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
```

```
    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}

suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
```

```
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}

suspend fun deletIssuesTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun queryMovieTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = "year"

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.N("2013")
```

```
    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}

suspend fun scanMovies(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Kotlin.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Query
- Scan
- UpdateItem

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB getting started demo using PHP!
\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDBService();

        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
```

```
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
$tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
|| $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
```

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
        $attributes = ["rating" =>
            [
                'AttributeName' => 'rating',
                'AttributeType' => 'N',
                'Value' => $rating,
            ],
            'plot' => [
                'AttributeName' => 'plot',
                'AttributeType' => 'S',
                'Value' => $plot,
            ]
        ];
        $service->updateItemAttributesByKey($tableName, $key, $attributes);
        echo "Movie added and updated.";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);


        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
 {$movie['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

        $movie = $service->getItemByKey($tableName, $key);
```

```
        echo "Ok, you have rated {$movie['Item']['title']['S']} as a
{$movie['Item']['rating']['N']}\n";

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
```

```
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para PHP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma classe que encapsule uma tabela do DynamoDB.

```
from decimal import Decimal
from io import BytesIO
import json
import logging
import os
from pprint import pprint
import requests
from zipfile import ZipFile
import boto3
from boto3.dynamodb.conditions import Key
from botocore.exceptions import ClientError
from question import Question

logger = logging.getLogger(__name__)

class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            self.table = table
        return exists


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
```

```
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
```

```
            return tables


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
```

```
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.
```

```
        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return response["Items"]


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies


    def delete_movie(self, title, year):
        """
```

```
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Crie uma função auxiliar para baixar e extrair o arquivo JSON de exemplo.

```
def get_sample_movie_data(movie_file_name):
    """
    Gets sample movie data, either from a local file or by first downloading it
 from
    the Amazon DynamoDB developer guide.
```

```
    :param movie_file_name: The local file name where the movie data is stored in
 JSON format.
    :return: The movie data as a dict.
    """
    if not os.path.isfile(movie_file_name):
        print(f"Downloading {movie_file_name}...")
        movie_content = requests.get(
            "https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip"
        )
        movie_zip = ZipFile(BytesIO(movie_content.content))
        movie_zip.extractall()

    try:
        with open(movie_file_name) as movie_file:
            movie_data = json.load(movie_file, parse_float=Decimal)
    except FileNotFoundError:
        print(
            f"File {movie_file_name} not found. You must first download the file
 to "
            "run this demo. See the README for instructions."
        )
        raise
    else:
        # The sample file lists over 4000 movies, return only the first 250.
        return movie_data[:250]
```

Execute um cenário interativo para criar a tabela e executar ações nela.

```
def run_scenario(table_name, movie_file_name, dyn_resource):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB getting started demo.")
    print("-" * 88)

    movies = Movies(dyn_resource)
    movies_exists = movies.exists(table_name)
    if not movies_exists:
```

```
        print(f"\nCreating table {table_name}...")
        movies.create_table(table_name)
        print(f"\nCreated table {movies.table.name}.")

    my_movie = Question.ask_questions(
        [
            Question(
                "title", "Enter the title of a movie you want to add to the
table: "
            ),
            Question("year", "What year was it released? ", Question.is_int),
            Question(
                "rating",
                "On a scale of 1 - 10, how do you rate it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question("plot", "Summarize the plot for me: "),
        ]
    )
    movies.add_movie(**my_movie)
    print(f"\nAdded '{my_movie['title']}' to '{movies.table.name}'.")
    print("-" * 88)

    movie_update = Question.ask_questions(
        [
            Question(
                "rating",
                f"\nLet's update your movie.\nYou rated it {my_movie['rating']},
what new "
                f"rating would you give it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question(
                "plot",
                f"You summarized the plot as '{my_movie['plot']}'.\nWhat would
you say now? ",
            ),
        ]
    )
    my_movie.update(movie_update)
    updated = movies.update_movie(**my_movie)
    print(f"\nUpdated '{my_movie['title']}' with new attributes:")
```

```
    pprint(updated)
    print("-" * 88)

    if not movies_exists:
        movie_data = get_sample_movie_data(movie_file_name)
        print(f"\nReading data from '{movie_file_name}' into your table.")
        movies.write_batch(movie_data)
        print(f"\nWrote {len(movie_data)} movies into {movies.table.name}.")
    print("-" * 88)

    title = "The Lord of the Rings: The Fellowship of the Ring"
    if Question.ask_question(
        f"Let's move on...do you want to get info about '{title}'? (y/n) ",
        Question.is_yesno,
    ):
        movie = movies.get_movie(title, 2001)
        print("\nHere's what I found:")
        pprint(movie)
    print("-" * 88)

    ask_for_year = True
    while ask_for_year:
        release_year = Question.ask_question(
            f"\nLet's get a list of movies released in a given year. Enter a year
between "
            f"1972 and 2018: ",
            Question.is_int,
            Question.in_range(1972, 2018),
        )
        releases = movies.query_movies(release_year)
        if releases:
            print(f"There were {len(releases)} movies released in
{release_year}:")
            for release in releases:
                print(f"\t{release['title']}")
            ask_for_year = False
        else:
            print(f"I don't know about any movies released in {release_year}!")
            ask_for_year = Question.ask_question(
                "Try another year? (y/n) ", Question.is_yesno
            )
    print("-" * 88)

    years = Question.ask_questions(
```

```
        [
            Question(
                "first",
                f"\nNow let's scan for movies released in a range of years. Enter
a year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
            Question(
                "second",
                "Now enter another year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
        ]
    )
    releases = movies.scan_movies(years)
    if releases:
        count = Question.ask_question(
            f"\nFound {len(releases)} movies. How many do you want to see? ",
            Question.is_int,
            Question.in_range(1, len(releases)),
        )
        print(f"\nHere are your {count} movies:\n")
        pprint(releases[:count])
    else:
        print(
            f"I don't know about any movies released between {years['first']} "
            f"and {years['second']}."
        )
    print("-" * 88)

    if Question.ask_question(
        f"\nLet's remove your movie from the table. Do you want to remove "
        f"'{my_movie['title']}'? (y/n)",
        Question.is_yesno,
    ):
        movies.delete_movie(my_movie["title"], my_movie["year"])
        print(f"\nRemoved '{my_movie['title']}' from the table.")
    print("-" * 88)

    if Question.ask_question(f"\nDelete the table? (y/n) ", Question.is_yesno):
        movies.delete_table()
        print(f"Deleted {table_name}.")
```

```
    else:
        print(
            "Don't forget to delete the table when you're done or you might incur
 "
            "charges on your account."
        )

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        run_scenario(
            "doc-example-table-movies", "moviedata.json",
 boto3.resource("dynamodb")
        )
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

Esse cenário usa a classe auxiliar a seguir para fazer perguntas em um prompt de comando.

```
class Question:
    """
    A helper class to ask questions at a command prompt and validate and convert
    the answers.
    """

    def __init__(self, key, question, *validators):
        """
        :param key: The key that is used for storing the answer in a dict, when
                    multiple questions are asked in a set.
        :param question: The question to ask.
        :param validators: The answer is passed through the list of validators
 until
                           one fails or they all pass. Validators may also
 convert the
                           answer to another form, such as from a str to an int.
        """
        self.key = key
        self.question = question
        self.validators = Question.non_empty, *validators
```

```
    @staticmethod
    def ask_questions(questions):
        """
        Asks a set of questions and stores the answers in a dict.

        :param questions: The list of questions to ask.
        :return: A dict of answers.
        """
        answers = {}
        for question in questions:
            answers[question.key] = Question.ask_question(
                question.question, *question.validators
            )
        return answers

    @staticmethod
    def ask_question(question, *validators):
        """
        Asks a single question and validates it against a list of validators.
        When an answer fails validation, the complaint is printed and the
question
        is asked again.

        :param question: The question to ask.
        :param validators: The list of validators that the answer must pass.
        :return: The answer, converted to its final form by the validators.
        """
        answer = None
        while answer is None:
            answer = input(question)
            for validator in validators:
                answer, complaint = validator(answer)
                if answer is None:
                    print(complaint)
                    break
        return answer

    @staticmethod
    def non_empty(answer):
        """
        Validates that the answer is not empty.
        :return: The non-empty answer, or None.
        """
```

```
        return answer if answer != "" else None, "I need an answer. Please?"

    @staticmethod
    def is_yesno(answer):
        """
        Validates a yes/no answer.
        :return: True when the answer is 'y'; otherwise, False.
        """
        return answer.lower() == "y", ""

    @staticmethod
    def is_int(answer):
        """
        Validates that the answer can be converted to an int.
        :return: The int answer; otherwise, None.
        """
        try:
            int_answer = int(answer)
        except ValueError:
            int_answer = None
        return int_answer, f"{answer} must be a valid integer."

    @staticmethod
    def is_letter(answer):
        """
        Validates that the answer is a letter.
        :return The letter answer, converted to uppercase; otherwise, None.
        """
        return (
            answer.upper() if answer.isalpha() else None,
            f"{answer} must be a single letter.",
        )

    @staticmethod
    def is_float(answer):
        """
        Validate that the answer can be converted to a float.
        :return The float answer; otherwise, None.
        """
        try:
            float_answer = float(answer)
        except ValueError:
            float_answer = None
        return float_answer, f"{answer} must be a valid float."
```

```
    @staticmethod
    def in_range(lower, upper):
        """
        Validate that the answer is within a range. The answer must be of a type
that can
        be compared to the lower and upper bounds.
        :return: The answer, if it is within the range; otherwise, None.
        """

        def _validate(answer):
            return (
                answer if lower <= answer <= upper else None,
                f"{answer} must be between {lower} and {upper}.",
            )

        return _validate
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Python (Boto3).
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma classe que encapsule uma tabela do DynamoDB.

```
# Creates an Amazon DynamoDB table that can be used to store movie data.
# The table uses the release year of the movie as the partition key and the
# title as the sort key.
#
# @param table_name [String] The name of the table to create.
# @return [Aws::DynamoDB::Table] The newly created table.
def create_table(table_name)
  @table = @dynamo_resource.create_table(
    table_name: table_name,
    key_schema: [
      { attribute_name: 'year', key_type: 'HASH' }, # Partition key
      { attribute_name: 'title', key_type: 'RANGE' } # Sort key
    ],
    attribute_definitions: [
      { attribute_name: 'year', attribute_type: 'N' },
      { attribute_name: 'title', attribute_type: 'S' }
    ],
    billing_mode: 'PAY_PER_REQUEST'
  )
  @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
  @table
rescue Aws::DynamoDB::Errors::ServiceError => e
  @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
  raise
end
```

Crie uma função auxiliar para baixar e extrair o arquivo JSON de exemplo.

```
  # Gets sample movie data, either from a local file or by first downloading it
 from
  # the Amazon DynamoDB Developer Guide.
  #
  # @param movie_file_name [String] The local file name where the movie data is
 stored in JSON format.
  # @return [Hash] The movie data as a Hash.
  def fetch_movie_data(movie_file_name)
    if !File.file?(movie_file_name)
      @logger.debug("Downloading #{movie_file_name}...")
      movie_content = URI.open(
        'https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip'
      )
      movie_json = ''
      Zip::File.open_buffer(movie_content) do |zip|
        zip.each do |entry|
          movie_json = entry.get_input_stream.read
        end
      end
    else
      movie_json = File.read(movie_file_name)
    end
    movie_data = JSON.parse(movie_json)
    # The sample file lists over 4000 movies. This returns only the first 250.
    movie_data.slice(0, 250)
  rescue StandardError => e
    puts("Failure downloading movie data:\n#{e}")
    raise
  end
```

Execute um cenário interativo para criar a tabela e executar ações nela.

```
  table_name = "doc-example-table-movies-#{rand(10**4)}"
  scaffold = Scaffold.new(table_name)
  dynamodb_wrapper = DynamoDBBasics.new(table_name)

  new_step(1, 'Create a new DynamoDB table if none already exists.')
  unless scaffold.exists?(table_name)
    puts("\nNo such table: #{table_name}. Creating it...")
    scaffold.create_table(table_name)
    print "Done!\n".green
```

```
 end

 new_step(2, 'Add a new record to the DynamoDB table.')
 my_movie = {}
 my_movie[:title] = CLI::UI::Prompt.ask('Enter the title of a movie to add to
the table. E.g. The Matrix')
 my_movie[:year] = CLI::UI::Prompt.ask('What year was it released? E.g.
1989').to_i
 my_movie[:rating] = CLI::UI::Prompt.ask('On a scale of 1 - 10, how do you rate
it? E.g. 7').to_i
 my_movie[:plot] = CLI::UI::Prompt.ask('Enter a brief summary of the plot. E.g.
A man awakens to a new reality.')
 dynamodb_wrapper.add_item(my_movie)
 puts("\nNew record added:")
 puts JSON.pretty_generate(my_movie).green
 print "Done!\n".green

 new_step(3, 'Update a record in the DynamoDB table.')
 my_movie[:rating] = CLI::UI::Prompt.ask("Let's update the movie you added with
a new rating, e.g. 3:").to_i
 response = dynamodb_wrapper.update_item(my_movie)
 puts("Updated '#{my_movie[:title]}' with new attributes:")
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(4, 'Get a record from the DynamoDB table.')
 puts("Searching for #{my_movie[:title]} (#{my_movie[:year]})...")
 response = dynamodb_wrapper.get_item(my_movie[:title], my_movie[:year])
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(5, 'Write a batch of items into the DynamoDB table.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
 print "Done!\n".green

 new_step(5, 'Query for a batch of items by key.')
 loop do
   release_year = CLI::UI::Prompt.ask('Enter a year between 1972 and 2018, e.g.
1999:').to_i
```

```
    results = dynamodb_wrapper.query_items(release_year)
    if results.any?
      puts("There were #{results.length} movies released in #{release_year}:")
      results.each do |movie|
        print "\t #{movie['title']}".green
      end
      break
    else
      continue = CLI::UI::Prompt.ask("Found no movies released in
 #{release_year}! Try another year? (y/n)")
      break unless continue.eql?('y')
    end
  end
  print "\nDone!\n".green

  new_step(6, 'Scan for a batch of items using a filter expression.')
  years = {}
  years[:start] = CLI::UI::Prompt.ask('Enter a starting year between 1972 and
 2018:')
  years[:end] = CLI::UI::Prompt.ask('Enter an ending year between 1972 and
 2018:')
  releases = dynamodb_wrapper.scan_items(years)
  if !releases.empty?
    puts("Found #{releases.length} movies.")
    count = Question.ask(
      'How many do you want to see? ', method(:is_int), in_range(1,
 releases.length)
    )
    puts("Here are your #{count} movies:")
    releases.take(count).each do |release|
      puts("\t#{release['title']}")
    end
  else
    puts("I don't know about any movies released between #{years[:start]} "\
         "and #{years[:end]}.")
  end
  print "\nDone!\n".green

  new_step(7, 'Delete an item from the DynamoDB table.')
  answer = CLI::UI::Prompt.ask("Do you want to remove '#{my_movie[:title]}'? (y/
n) ")
  if answer.eql?('y')
    dynamodb_wrapper.delete_item(my_movie[:title], my_movie[:year])
    puts("Removed '#{my_movie[:title]}' from the table.")
```

```
    print "\nDone!\n".green
  end

  new_step(8, 'Delete the DynamoDB table.')
  answer = CLI::UI::Prompt.ask('Delete the table? (y/n)')
  if answer.eql?('y')
    scaffold.delete_table
    puts("Deleted #{table_name}.")
  else
    puts("Don't forget to delete the table when you're done!")
  end
  print "\nThanks for watching!\n".green
rescue Aws::Errors::ServiceError
  puts('Something went wrong with the demo.')
rescue Errno::ENOENT
  true
end
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Ruby.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    " Create an Amazon Dynamo DB table.

    TRY.
        DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
        DATA(lo_dyn) = /aws1/cl_dyn_factory=>create( lo_session ).
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        DATA(oo_result) = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
```

```
      " It throws exception if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.

    " Describe table
    TRY.
        DATA(lo_table) = lo_dyn->describetable( iv_tablename = iv_table_name ).
        DATA(lv_tablename) = lo_table->get_table( )->ask_tablename( ).
        MESSAGE 'The table name is ' && lv_tablename TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table does not exist' TYPE 'E'.
    ENDTRY.

    " Put items into the table.
    TRY.
        DATA(lo_resp_putitem) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1975' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.5' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s = 'Star
 Wars' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1978' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
```

```
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '8.1' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Speed' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1994' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.9' }| ) ) )
          ) ).
        " TYPE REF TO ZCL_AWS1_dyn_PUT_ITEM_OUTPUT
        MESSAGE '3 rows inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Get item from table.
    TRY.
        DATA(lo_resp_getitem) = lo_dyn->getitem(
          iv_tablename               = iv_table_name
          it_key                     = VALUE /aws1/cl_dynattributevalue=>tt_key(
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ) ).
        DATA(lt_attr) = lo_resp_getitem->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'year' ]-value.
```

```
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Query item from table.
    TRY.
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
             ( NEW /aws1/cl_dynattributevalue( iv_n = '1975' ) ) ).
        DATA(lt_keyconditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
            it_attributevaluelist = lt_attributelist
            iv_comparisonoperator = |EQ|
          ) ) ) ).
        DATA(lo_query_result) = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_keyconditions ).
        DATA(lt_items) = lo_query_result->get_items( ).
        READ TABLE lo_query_result->get_items( ) INTO DATA(lt_item) INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Scan items from table.
    TRY.
        DATA(lo_scan_result) = lo_dyn->scan( iv_tablename = iv_table_name ).
        lt_items = lo_scan_result->get_items( ).
        " Read the first item and display the attributes.
        READ TABLE lo_query_result->get_items( ) INTO lt_item INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
```

```
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Update items from table.
    TRY.
        DATA(lt_attributeupdates) = VALUE /aws1/
cl_dynattrvalueupdate=>tt_attributeupdates(
          ( VALUE /aws1/cl_dynattrvalueupdate=>ts_attributeupdates_maprow(
          key = 'rating' value = NEW /aws1/cl_dynattrvalueupdate(
            io_value  = NEW /aws1/cl_dynattributevalue( iv_n = '7.6' )
            iv_action = |PUT| ) ) ) ).
        DATA(lt_key) = VALUE /aws1/cl_dynattributevalue=>tt_key(
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 '1980' ) ) ) ).
        DATA(lo_resp) = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = lt_key
          it_attributeupdates = lt_attributeupdates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Delete table.
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
```

```
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para SAP ABAP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Uma classe Swift que gerencia chamadas do DynamoDB para o SDK para Swift.

```
import AWSDynamoDB
import Foundation

/// An enumeration of error codes representing issues that can arise when using
/// the `MovieTable` class.
enum MoviesError: Error {
    /// The specified table wasn't found or couldn't be created.
```

```
    case TableNotFound
    /// The specified item wasn't found or couldn't be created.
    case ItemNotFound
    /// The Amazon DynamoDB client is not properly initialized.
    case UninitializedClient
    /// The table status reported by Amazon DynamoDB is not recognized.
    case StatusUnknown
    /// One or more specified attribute values are invalid or missing.
    case InvalidAttributes
}

/// A class representing an Amazon DynamoDB table containing movie
/// information.
public class MovieTable {
    var ddbClient: DynamoDBClient?
    let tableName: String

    /// Create an object representing a movie table in an Amazon DynamoDB
    /// database.
    ///
    /// - Parameters:
    ///   - region: The optional Amazon Region to create the database in.
    ///   - tableName: The name to assign to the table. If not specified, a
    ///     random table name is generated automatically.
    ///
    /// > Note: The table is not necessarily available when this function
    /// returns. Use `tableExists()` to check for its availability, or
    /// `awaitTableActive()` to wait until the table's status is reported as
    /// ready to use by Amazon DynamoDB.
    ///
    init(region: String? = nil, tableName: String) async throws {
        do {
            let config = try await DynamoDBClient.DynamoDBClientConfiguration()
            if let region = region {
                config.region = region
            }

            self.ddbClient = DynamoDBClient(config: config)
            self.tableName = tableName

            try await self.createTable()
        } catch {
            print("ERROR: ", dump(error, name: "Initializing Amazon
 DynamoDBClient client"))
```

```
            throw error
        }
    }

    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }


    /// Check to see if the table exists online yet.
    ///
    /// - Returns: `true` if the table exists, or `false` if not.
    ///
```

```
    func tableExists() async throws -> Bool {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }

            return description.tableName == self.tableName
        } catch {
            print("ERROR: tableExists:", dump(error))
            throw error
        }
    }


    ///
    /// Waits for the table to exist and for its status to be active.
    ///
    func awaitTableActive() async throws {
        while try (await self.tableExists() == false) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
        }

        while try (await self.getTableStatus() != .active) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
```

```
        }
    }


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }


    /// Get the table's status.
    ///
    /// - Returns: The table status, as defined by the
    ///   `DynamoDBClientTypes.TableStatus` enum.
    ///
    func getTableStatus() async throws -> DynamoDBClientTypes.TableStatus {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: self.tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }
            guard let status = description.tableStatus else {
                throw MoviesError.StatusUnknown
```

```
            }
            return status
        } catch {
            print("ERROR: getTableStatus:", dump(error))
            throw error
        }
    }


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
jsonData)

            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }
```

```
            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
    }


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
```

```
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    /// Given a movie's details, add a movie to the Amazon DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title as a `String`.
    ///   - year: The release year of the movie (`Int`).
    ///   - rating: The movie's rating if available (`Double`; default is
    ///     `nil`).
    ///   - plot: A summary of the movie's plot (`String`; default is `nil`,
    ///     indicating no plot summary is available).
    ///
    func add(title: String, year: Int, rating: Double? = nil,
             plot: String? = nil) async throws
    {
        do {
            let movie = Movie(title: title, year: year, rating: rating, plot:
plot)
            try await self.add(movie: movie)
        } catch {
            print("ERROR: add with fields:", dump(error))
            throw error
        }
    }


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
```

```
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }
```

```
            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
```

```
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
```

```
                }

                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
```

```
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }


    /// Delete a movie, given its title and release year.
    ///
```

```
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
}
```

As estruturas usadas pela classe MovieTable para representar filmes.

```
import Foundation
import AWSDynamoDB

/// The optional details about a movie.
public struct Details: Codable {
    /// The movie's rating, if available.
    var rating: Double?
    /// The movie's plot, if available.
    var plot: String?
}

/// A structure describing a movie. The `year` and `title` properties are
/// required and are used as the key for Amazon DynamoDB operations. The
/// `info` sub-structure's two properties, `rating` and `plot`, are optional.
```

```
public struct Movie: Codable {
    /// The year in which the movie was released.
    var year: Int
    /// The movie's title.
    var title: String
    /// A `Details` object providing the optional movie rating and plot
    /// information.
    var info: Details

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - rating: The movie's rating (optional `Double`).
    ///   - plot: The movie's plot (optional `String`)
    init(title: String, year: Int, rating: Double? = nil, plot: String? = nil) {
        self.title = title
        self.year = year

        self.info = Details(rating: rating, plot: plot)
    }

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - info: The optional rating and plot information for the movie in a
    ///     `Details` object.
    init(title: String, year: Int, info: Details?){
        self.title = title
        self.year = year

        if info != nil {
            self.info = info!
        } else {
            self.info = Details(rating: nil, plot: nil)
        }
    }

    ///
```

```
    /// Return a new `MovieTable` object, given an array mapping string to Amazon
    /// DynamoDB attribute values.
    ///
    /// - Parameter item: The item information provided to the form used by
    ///   DynamoDB. This is an array of strings mapped to
    ///   `DynamoDBClientTypes.AttributeValue` values.
    init(withItem item: [Swift.String:DynamoDBClientTypes.AttributeValue]) throws
{
        // Read the attributes.

        guard let titleAttr = item["title"],
              let yearAttr = item["year"] else {
            throw MoviesError.ItemNotFound
        }
        let infoAttr = item["info"] ?? nil

        // Extract the values of the title and year attributes.

        if case .s(let titleVal) = titleAttr {
            self.title = titleVal
        } else {
            throw MoviesError.InvalidAttributes
        }

        if case .n(let yearVal) = yearAttr {
            self.year = Int(yearVal)!
        } else {
            throw MoviesError.InvalidAttributes
        }

        // Extract the rating and/or plot from the `info` attribute, if
        // they're present.

        var rating: Double? = nil
        var plot: String? = nil

        if infoAttr != nil, case .m(let infoVal) = infoAttr {
            let ratingAttr = infoVal["rating"] ?? nil
            let plotAttr = infoVal["plot"] ?? nil

            if ratingAttr != nil, case .n(let ratingVal) = ratingAttr {
                rating = Double(ratingVal) ?? nil
            }
            if plotAttr != nil, case .s(let plotVal) = plotAttr {
```

```
                plot = plotVal
            }
        }

        self.info = Details(rating: rating, plot: plot)
    }

    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
}
```

Um programa que usa a classe MovieTable para acessar um banco de dados do DynamoDB.

```
import ArgumentParser
import ClientRuntime
import Foundation

import AWSDynamoDB


@testable import MovieList

extension String {
    // Get the directory if the string is a file path.
    func directory() -> String {
        guard let lastIndex = lastIndex(of: "/") else {
            print("Error: String directory separator not found.")
            return ""
        }
        return String(self[...lastIndex])
    }
}

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The path of the sample movie data JSON file.")
    var jsonPath: String = #file.directory() + "../../../../../resources/
sample_files/movies.json"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion: String?

    @Option(
        help: ArgumentHelp("The level of logging for the Swift SDK to perform."),
        completion: .list([
            "critical",
            "debug",
            "error",
            "info",
            "notice",
            "trace",
            "warning"
        ])
    )
    var logLevel: String = "error"
```

```
    /// Configuration details for the command.
    static var configuration = CommandConfiguration(
        commandName: "basics",
        abstract: "A basic scenario demonstrating the usage of Amazon DynamoDB.",
        discussion: """
        An example showing how to use Amazon DynamoDB to perform a series of
        common database activities on a simple movie database.
        """
    )

    /// Called by ``main()`` to asynchronously run the AWS example.
    func runAsync() async throws {
        print("Welcome to the AWS SDK for Swift basic scenario for Amazon
DynamoDB!")

        //========================================================================
        // 1. Create the table. The Amazon DynamoDB table is represented by
        //    the `MovieTable` class.
        //========================================================================

        let tableName = "ddb-movies-sample-\(Int.random(in: 1 ... Int.max))"

        print("Creating table \"\(tableName)\"...")

        let movieDatabase = try await MovieTable(region: awsRegion,
                                                 tableName: tableName)

        print("\nWaiting for table to be ready to use...")
        try await movieDatabase.awaitTableActive()

        //========================================================================
        // 2. Add a movie to the table.
        //========================================================================

        print("\nAdding a movie...")
        try await movieDatabase.add(title: "Avatar: The Way of Water", year:
2022)
        try await movieDatabase.add(title: "Not a Real Movie", year: 2023)

        //========================================================================
        // 3. Update the plot and rating of the movie using an update
        //    expression.
        //========================================================================
```

```
        print("\nAdding details to the added movie...")
        _ = try await movieDatabase.update(title: "Avatar: The Way of Water",
year: 2022,
                                           rating: 9.2, plot: "It's a sequel.")

        //===========================================================================
        // 4. Populate the table from the JSON file.
        //===========================================================================

        print("\nPopulating the movie database from JSON...")
        try await movieDatabase.populate(jsonPath: jsonPath)

        //===========================================================================
        // 5. Get a specific movie by key. In this example, the key is a
        //    combination of `title` and `year`.
        //===========================================================================

        print("\nLooking for a movie in the table...")
        let gotMovie = try await movieDatabase.get(title: "This Is the End",
year: 2013)

        print("Found the movie \"\(gotMovie.title)\", released in
\(gotMovie.year).")
        print("Rating: \(gotMovie.info.rating ?? 0.0).")
        print("Plot summary: \(gotMovie.info.plot ?? "None.")")

        //===========================================================================
        // 6. Delete a movie.
        //===========================================================================

        print("\nDeleting the added movie...")
        try await movieDatabase.delete(title: "Avatar: The Way of Water", year:
2022)

        //===========================================================================
        // 7. Use a query with a key condition expression to return all movies
        //    released in a given year.
        //===========================================================================

        print("\nGetting movies released in 1994...")
        let movieList = try await movieDatabase.getMovies(fromYear: 1994)
        for movie in movieList {
            print("    \(movie.title)")
        }
```

```
        //=========================================================================
        // 8. Use `scan()` to return movies released in a range of years.
        //=========================================================================

        print("\nGetting movies released between 1993 and 1997...")
        let scannedMovies = try await movieDatabase.getMovies(firstYear: 1993,
 lastYear: 1997)
        for movie in scannedMovies {
            print("    \(movie.title) (\(movie.year))")
        }

        //=========================================================================
        // 9. Delete the table.
        //=========================================================================

        print("\nDeleting the table...")
        try await movieDatabase.deleteTable()
    }
}

@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Swift.
  - BatchWriteItem
  - CreateTable
  - DeleteItem

- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Ações do DynamoDB usando AWS SDKs

Os exemplos de código a seguir demonstram como realizar ações específicas do DynamoDB com AWS SDKs. Cada exemplo inclui um link para o GitHub, em que é possível encontrar instruções para configurar e executar o código.

Esses trechos chamam a API do DynamoDB e são trechos de código de programas maiores que devem ser executados no contexto. É possível ver as ações em contexto em Cenários do DynamoDB usando AWS SDKs .

Os exemplos a seguir incluem apenas as ações mais utilizadas. Para obter uma lista completa, consulte a Referência de API do Amazon DynamoDB.

Exemplos

- Usar BatchExecuteStatement com um SDK da AWS
- Usar BatchGetItem com o AWS SDK ou a CLI
- Usar BatchWriteItem com o AWS SDK ou a CLI
- Usar CreateTable com o AWS SDK ou a CLI
- Usar DeleteItem com o AWS SDK ou a CLI
- Usar DeleteTable com o AWS SDK ou a CLI
- Usar DescribeTable com o AWS SDK ou a CLI
- Usar DescribeTimeToLive com o AWS SDK ou a CLI
- Usar ExecuteStatement com um SDK da AWS

- Usar GetItem com o AWS SDK ou a CLI
- Usar ListTables com o AWS SDK ou a CLI
- Usar PutItem com o AWS SDK ou a CLI
- Usar Query com o AWS SDK ou a CLI
- Usar Scan com o AWS SDK ou a CLI
- Usar UpdateItem com o AWS SDK ou a CLI
- Usar UpdateTable com o AWS SDK ou a CLI
- Usar UpdateTimeToLive com o AWS SDK ou a CLI

Usar **`BatchExecuteStatement`** com um SDK da AWS

Os exemplos de código a seguir mostram como usar o `BatchExecuteStatement`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Excluir dados usando DELETE do PartiQL
- Inserir dados usando INSERT do PartiQL
- Consultar uma tabela usando lotes de instruções PartiQL
- Consultar dados usando SELECT do PartiQL
- Atualizar dados usando UPDATE do PartiQL

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Use lotes de instruções INSERT para adicionar itens.

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
```

```
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }
```

```
                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
```

```
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }
```

Use lotes de instruções SELECT para obter itens.

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },
```

```
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

Use lotes de instruções UPDATE para atualizar itens.

```
        /// <summary>
        /// Updates information for multiple movies.
```

```
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
```

```
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Use lotes de instruções DELETE para excluir itens.

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
```

```
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Use lotes de instruções INSERT para adicionar itens.

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
```

```
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
```

```
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

Use lotes de instruções SELECT para obter itens.

```
    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();
```

```
            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Use lotes de instruções UPDATE para atualizar itens.

```
    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";
```

```
        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Use lotes de instruções DELETE para excluir itens.

```
    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());
```

```
        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Defina uma estrutura de receptor de função para o exemplo.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Use lotes de instruções INSERT para adicionar itens.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
```

```
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}
```

Use lotes de instruções SELECT para obter itens.

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
```

```
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}
```

Use lotes de instruções UPDATE para atualizar itens.

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
```

```
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Use lotes de instruções DELETE para excluir itens.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Crie um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
```

```
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtenha um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Atualize um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Exclua um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
```

```
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para JavaScript.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }
```

```
        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function insertItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para PHP.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
 with
```

```
                            each statement. This list must be in the same order as
the
                            statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Leia um lote de itens usando o PartiQL.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Selects a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_select(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "SELECT * FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({ statements: request_items })
  end
```

Exclua um lote de itens usando o PartiQL.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
```

```
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamodb = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamodb.table(table_name)
end

# Deletes a batch of items from a table using PartiQL
#
# @param batch_titles [Array] Collection of movie titles
# @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
def batch_execute_write(batch_titles)
  request_items = batch_titles.map do |title, year|
    {
      statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
      parameters: [title, year]
    }
  end
  @dynamodb.client.batch_execute_statement({ statements: request_items })
end
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API do AWS SDK para Ruby.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`BatchGetItem`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `BatchGetItem`.

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void
 RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { _table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                        } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
                        } }
                          }
                      }
                  }},
                {
                    _table2Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
```

```
                        {
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 1"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = await client.BatchGetItemAsync(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the
response.
```

```
                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue>
attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
```

```
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }

 Console.WriteLine("*************************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- Para obter detalhes da API, consulte BatchGetItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_batch_get_item
#
# This function gets a batch of items from a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the keys of the items to get.
#
#  Returns:
#       The items as json output.
```

```
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_get_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_get_item"
    echo "Get a batch of items from a DynamoDB table."
    echo " -i item  -- Path to json file containing the keys of the items to
 get."
    echo ""
  }

  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  response=$(aws dynamodb batch-get-item \
    --request-items file://"$item")
  local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-get-item operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
```

```
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte BatchGetItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Batch get items from different Amazon DynamoDB tables.
/*!
  \sa batchGetItem()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::batchGetItem(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    Aws::DynamoDB::Model::BatchGetItemRequest request;

    // Table1: Forum.
    Aws::String table1Name = "Forum";
    Aws::DynamoDB::Model::KeysAndAttributes table1KeysAndAttributes;

    // Table1: Projection expression.
    table1KeysAndAttributes.SetProjectionExpression("#n, Category, Messages,
#v");

    // Table1: Expression attribute names.
    Aws::Http::HeaderValueCollection headerValueCollection;
    headerValueCollection.emplace("#n", "Name");
    headerValueCollection.emplace("#v", "Views");
    table1KeysAndAttributes.SetExpressionAttributeNames(headerValueCollection);

    // Table1: Set key name, type, and value to search.
    std::vector<Aws::String> nameValues = {"Amazon DynamoDB", "Amazon S3"};
    for (const Aws::String &name: nameValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetS(name);
        keys.emplace("Name", key);
        table1KeysAndAttributes.AddKeys(keys);
    }

    Aws::Map<Aws::String, Aws::DynamoDB::Model::KeysAndAttributes> requestItems;
    requestItems.emplace(table1Name, table1KeysAndAttributes);

    // Table2: ProductCatalog.
    Aws::String table2Name = "ProductCatalog";
    Aws::DynamoDB::Model::KeysAndAttributes table2KeysAndAttributes;
    table2KeysAndAttributes.SetProjectionExpression("Title, Price, Color");

    // Table2: Set key name, type, and value to search.
    std::vector<Aws::String> idValues = {"102", "103", "201"};
    for (const Aws::String &id: idValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetN(id);
        keys.emplace("Id", key);
        table2KeysAndAttributes.AddKeys(keys);
    }
```

```
   requestItems.emplace(table2Name, table2KeysAndAttributes);

   bool result = true;
   do {  // Use a do loop to handle pagination.
       request.SetRequestItems(requestItems);
       const Aws::DynamoDB::Model::BatchGetItemOutcome &outcome =
dynamoClient.BatchGetItem(
               request);

       if (outcome.IsSuccess()) {
           for (const auto &responsesMapEntry:
outcome.GetResult().GetResponses()) {
               Aws::String tableName = responsesMapEntry.first;
               const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &tableResults = responsesMapEntry.second;
               std::cout << "Retrieved " << tableResults.size()
                         << " responses for table '" << tableName << "'.\n"
                         << std::endl;
               if (tableName == "Forum") {

                   std::cout << "Name | Category | Message | Views" <<
std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Name").GetS() << " | ";
                       std::cout << item.at("Category").GetS() << " | ";
                       std::cout << (item.count("Message") == 0 ? "" : item.at(
                               "Messages").GetN()) << " | ";
                       std::cout << (item.count("Views") == 0 ? "" : item.at(
                               "Views").GetN()) << std::endl;
                   }
               }
               else {
                   std::cout << "Title | Price | Color" << std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Title").GetS() << " | ";
                       std::cout << (item.count("Price") == 0 ? "" : item.at(
                               "Price").GetN());
                       if (item.count("Color")) {
                           std::cout << " | ";
                           for (const
std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> &listItem: item.at(
```

```
                                    "Color").GetL())
                                std::cout << listItem->GetS() << " ";
                        }
                        std::cout << std::endl;
                    }
                }
                std::cout << std::endl;
            }

            // If necessary, repeat request for remaining items.
            requestItems = outcome.GetResult().GetUnprocessedKeys();
        }
        else {
            std::cerr << "Batch get item failed: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
            break;
        }
    } while (!requestItems.empty());

    return result;
}
```

- Para obter detalhes da API, consulte BatchGetItem na Referência da API AWS SDK para C++.

CLI

AWS CLI

Como recuperar vários itens de uma tabela

O exemplo de `batch-get-items` a seguir lê vários itens da tabela `MusicCollection` usando um lote de três solicitações `GetItem` e solicita o número de unidades de capacidade de leitura consumidas pela operação. O comando retorna somente o atributo `AlbumTitle`.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

Conteúdo de `request-items.json`:

```
{
    "MusicCollection": {
        "Keys": [
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Call Me Today"}
            },
            {
                "Artist": {"S": "Acme Band"},
                "SongTitle": {"S": "Happy Day"}
            },
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Scared of My Shadow"}
            }
        ],
        "ProjectionExpression":"AlbumTitle"
    }
}
```

Saída:

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
```

```
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

Para obter mais informações, consulte Operações em lote no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte BatchGetItem na Referência de comandos da AWS CLI.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Mostra como receber itens em lote usando o cliente de serviço.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchReadItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItems(dynamoDbClient, tableName);
    }

    public static void getBatchItems(DynamoDbClient dynamoDbClient, String
 tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();
```

```
        // Make the batchGetItem request.
        BatchGetItemResponse batchGetItemResponse =
 dynamoDbClient.batchGetItem(batchGetItemRequest);

        // Extract and print the retrieved items.
        Map<String, List<Map<String, AttributeValue>>> responses =
 batchGetItemResponse.responses();
        if (responses.containsKey(tableName)) {
            List<Map<String, AttributeValue>> musicItems =
 responses.get(tableName);
            for (Map<String, AttributeValue> item : musicItems) {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            }
        } else {
            System.out.println("No items retrieved.");
        }
    }
}
```

Mostra como receber itens em lote usando o cliente de serviço e um paginador.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BatchGetItemsPaginator {

    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
```

```
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """";

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItemsPaginator(dynamoDbClient, tableName) ;
    }

    public static void getBatchItemsPaginator(DynamoDbClient dynamoDbClient,
 String tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Use batchGetItemPaginator for paginated requests.
        dynamoDbClient.batchGetItemPaginator(batchGetItemRequest).stream()
            .flatMap(response -> response.responses().getOrDefault(tableName,
 Collections.emptyList()).stream())
            .forEach(item -> {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            });
    }
}
```

- Para obter detalhes da API, consulte BatchGetItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte BatchGet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
```

```
  });

  const response = await docClient.send(command);
  console.log(response.Responses.Books);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte BatchGetItem na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};
```

```
ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte BatchGetItem na Referência da API AWS SDK para JavaScript.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: recebe o item com o SongTitle “Somewhere Down The Road” das tabelas “Music” e “Songs” do DynamoDB.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
```

```
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Saída:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Para ver detalhes da API, consulte BatchGetItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: recebe o item com o SongTitle “Somewhere Down The Road” das tabelas “Music” e “Songs” do DynamoDB.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
```

```
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Saída:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Para ver detalhes da API, consulte BatchGetItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import decimal
import json
import logging
import os
import pprint
```

```
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
dynamodb = boto3.resource("dynamodb")

MAX_GET_SIZE = 100  # Amazon DynamoDB rejects a get batch larger than 100 items.


def do_batch_get(batch_keys):
    """
    Gets a batch of items from Amazon DynamoDB. Batches can contain keys from
    more than one table.

    When Amazon DynamoDB cannot process all items in a batch, a set of
 unprocessed
    keys is returned. This function uses an exponential backoff algorithm to
 retry
    getting the unprocessed keys until all are retrieved or the specified
    number of tries is reached.

    :param batch_keys: The set of keys to retrieve. A batch can contain at most
 100
                       keys. Otherwise, Amazon DynamoDB returns an error.
    :return: The dictionary of retrieved items grouped under their respective
             table names.
    """
    tries = 0
    max_tries = 5
    sleepy_time = 1  # Start with 1 second of sleep, then exponentially increase.
    retrieved = {key: [] for key in batch_keys}
    while tries < max_tries:
        response = dynamodb.batch_get_item(RequestItems=batch_keys)
        # Collect any retrieved items and retry unprocessed keys.
        for key in response.get("Responses", []):
            retrieved[key] += response["Responses"][key]
        unprocessed = response["UnprocessedKeys"]
        if len(unprocessed) > 0:
            batch_keys = unprocessed
            unprocessed_count = sum(
                [len(batch_key["Keys"]) for batch_key in batch_keys.values()]
            )
            logger.info(
```

```
                "%s unprocessed keys returned. Sleep, then retry.",
 unprocessed_count
            )
            tries += 1
            if tries < max_tries:
                logger.info("Sleeping for %s seconds.", sleepy_time)
                time.sleep(sleepy_time)
                sleepy_time = min(sleepy_time * 2, 32)
        else:
            break

    return retrieved
```

- Para obter detalhes da API, consulte BatchGetItem na Referência da API AWS SDK para Python (Boto3).

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Gets an array of `Movie` objects describing all the movies in the
    /// specified list. Any movies that aren't found in the list have no
    /// corresponding entry in the resulting array.
    ///
    /// - Parameters
    ///     - keys: An array of tuples, each of which specifies the title and
    ///       release year of a movie to fetch from the table.
    ///
    /// - Returns:
```

```
    ///     - An array of `Movie` objects describing each match found in the
    ///     table.
    ///
    /// - Throws:
    ///     - `MovieError.ClientUninitialized` if the DynamoDB client has not
    ///     been initialized.
    ///     - DynamoDB errors are thrown without change.
    func batchGet(keys: [(title: String, year: Int)]) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MovieError.ClientUninitialized
            }

            var movieList: [Movie] = []
            var keyItems: [[Swift.String: DynamoDBClientTypes.AttributeValue]] =
[]

            // Convert the list of keys into the form used by DynamoDB.

            for key in keys {
                let item: [Swift.String: DynamoDBClientTypes.AttributeValue] = [
                    "title": .s(key.title),
                    "year": .n(String(key.year))
                ]
                keyItems.append(item)
            }

            // Create the input record for `batchGetItem()`. The list of
requested
            // items is in the `requestItems` property. This array contains one
            // entry for each table from which items are to be fetched. In this
            // example, there's only one table containing the movie data.
            //
            // If we wanted this program to also support searching for matches
            // in a table of book data, we could add a second `requestItem`
            // mapping the name of the book table to the list of items we want to
            // find in it.
            let input = BatchGetItemInput(
                requestItems: [
                    self.tableName: .init(
                        consistentRead: true,
                        keys: keyItems
                    )
                ]
```

```
            )

            // Fetch the matching movies from the table.

            let output = try await client.batchGetItem(input: input)

            // Get the set of responses. If there aren't any, return the empty
            // movie list.

            guard let responses = output.responses else {
                return movieList
            }

            // Get the list of matching items for the table with the name
            // `tableName`.

            guard let responseList = responses[self.tableName] else {
                return movieList
            }

            // Create `Movie` items for each of the matching movies in the table
            // and add them to the `MovieList` array.

            for response in responseList {
                try movieList.append(Movie(withItem: response))
            }

            return movieList
        } catch {
            print("ERROR: batchGet", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte BatchGetItem na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`BatchWriteItem`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `BatchWriteItem`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Conheça os conceitos básicos

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Grava um lote de itens na tabela de filmes.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
```

```
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);
```

```
            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
##############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte BatchWriteItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
//! Batch write items from a JSON file.
/*!
  \sa batchWriteItem()
  \param jsonFilePath: JSON file path.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

/*
 * The input for this routine is a JSON file that you can download from the
 following URL:
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
SampleData.html.
 *
 * The JSON data uses the BatchWriteItem API request syntax. The JSON strings are
 * converted to AttributeValue objects. These AttributeValue objects will then
 generate
 * JSON strings when constructing the BatchWriteItem request, essentially
 outputting
 * their input.
 *
 * This is perhaps an artificial example, but it demonstrates the APIs.
 */

bool AwsDoc::DynamoDB::batchWriteItem(const Aws::String &jsonFilePath,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    std::ifstream fileStream(jsonFilePath);
```

```
   if (!fileStream) {
       std::cerr << "Error: could not open file '" << jsonFilePath << "'."
                 << std::endl;
   }

   std::stringstream stringStream;
   stringStream << fileStream.rdbuf();
   Aws::Utils::Json::JsonValue jsonValue(stringStream);

   Aws::DynamoDB::Model::BatchWriteItemRequest batchWriteItemRequest;
   Aws::Map<Aws::String, Aws::Utils::Json::JsonView> level1Map =
jsonValue.View().GetAllObjects();
   for (const auto &level1Entry: level1Map) {
       const Aws::Utils::Json::JsonView &entriesView = level1Entry.second;
       const Aws::String &tableName = level1Entry.first;
       // The JSON entries at this level are as follows:
       //  key - table name
       //  value - list of request objects
       if (!entriesView.IsListType()) {
           std::cerr << "Error: JSON file entry '"
                     << tableName << "' is not a list." << std::endl;
           continue;
       }

       Aws::Utils::Array<Aws::Utils::Json::JsonView> entries =
entriesView.AsArray();

       Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;
       if (AwsDoc::DynamoDB::addWriteRequests(tableName, entries,
                                              writeRequests)) {
           batchWriteItemRequest.AddRequestItems(tableName, writeRequests);
       }
   }

   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   Aws::DynamoDB::Model::BatchWriteItemOutcome outcome =
dynamoClient.BatchWriteItem(
           batchWriteItemRequest);

   if (outcome.IsSuccess()) {
       std::cout << "DynamoDB::BatchWriteItem was successful." << std::endl;
   }
```

```
    else {
        std::cerr << "Error with DynamoDB::BatchWriteItem. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return outcome.IsSuccess();
}

//! Convert requests in JSON format to a vector of WriteRequest objects.
/*!
  \sa addWriteRequests()
  \param tableName: Name of the table for the write operations.
  \param requestsJson: Request data in JSON format.
  \param writeRequests: Vector to receive the WriteRequest objects.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::addWriteRequests(const Aws::String &tableName,
                                        const
 Aws::Utils::Array<Aws::Utils::Json::JsonView> &requestsJson,

 Aws::Vector<Aws::DynamoDB::Model::WriteRequest> &writeRequests) {
    for (size_t i = 0; i < requestsJson.GetLength(); ++i) {
        const Aws::Utils::Json::JsonView &requestsEntry = requestsJson[i];
        if (!requestsEntry.IsObject()) {
            std::cerr << "Error: incorrect requestsEntry type "
                      << requestsEntry.WriteReadable() << std::endl;
            return false;
        }

        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> requestsMap =
 requestsEntry.GetAllObjects();

        for (const auto &request: requestsMap) {
            const Aws::String &requestType = request.first;
            const Aws::Utils::Json::JsonView &requestJsonView = request.second;

            if (requestType == "PutRequest") {
                if (!requestJsonView.ValueExists("Item")) {
                    std::cerr << "Error: item key missing for requests "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }
```

```
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 attributes;
                if (!getAttributeObjectsMap(requestJsonView.GetObject("Item"),
                                            attributes)) {
                    std::cerr << "Error getting attributes "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }

                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(attributes);
                writeRequests.push_back(
                        Aws::DynamoDB::Model::WriteRequest().WithPutRequest(
                                putRequest));
            }
            else {
                std::cerr << "Error: unimplemented request type '" << requestType
                          << "'." << std::endl;
            }
        }
    }

    return true;
}

//! Generate a map of AttributeValue objects from JSON records.
/*!
  \sa getAttributeObjectsMap()
  \param jsonView: JSONView of attribute records.
  \param writeRequests: Map to receive the AttributeValue objects.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::getAttributeObjectsMap(const Aws::Utils::Json::JsonView
 &jsonView,
                                         Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &attributes) {
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> objectsMap =
 jsonView.GetAllObjects();
    for (const auto &entry: objectsMap) {
        const Aws::String &attributeKey = entry.first;
        const Aws::Utils::Json::JsonView &attributeJsonView = entry.second;

        if (!attributeJsonView.IsObject()) {
```

```
            std::cerr << "Error: attribute not an object "
                      << attributeJsonView.WriteReadable() << std::endl;
            return false;
        }

        attributes.emplace(attributeKey,

 Aws::DynamoDB::Model::AttributeValue(attributeJsonView));
    }

    return true;
}
```

- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK para C++.

CLI

AWS CLI

Como adicionar vários itens a uma tabela

O exemplo de `batch-write-item` a seguir adiciona três novos itens à tabela `MusicCollection` usando um lote de três solicitações `PutItem`. Ele também solicita informações sobre o número de unidades de capacidade de gravação consumidas pela operação e quaisquer coleções de itens modificadas pela operação.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

Conteúdo de `request-items.json`:

```
{
    "MusicCollection": [
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
```

```
                    "SongTitle": {"S": "Call Me Today"},
                    "AlbumTitle": {"S": "Somewhat Famous"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "Acme Band"},
                    "SongTitle": {"S": "Happy Day"},
                    "AlbumTitle": {"S": "Songs About Life"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Scared of My Shadow"},
                    "AlbumTitle": {"S": "Blue Sky Blues"}
                }
            }
        }
    ]
}
```

Saída:

```
{
    "UnprocessedItems": {},
    "ItemCollectionMetrics": {
        "MusicCollection": [
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "No One You Know"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            },
```

```
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "Acme Band"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            }
        ]
    },
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 6.0,
            "Table": {
                "CapacityUnits": 3.0
            },
            "LocalSecondaryIndexes": {
                "AlbumTitleIndex": {
                    "CapacityUnits": 3.0
                }
            }
        }
    ]
}
```

Para obter mais informações, consulte Operações em lote no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte BatchWriteItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
```

```
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
```

```
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Insere vários itens em uma tabela usando o cliente de serviço.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutRequest;
import software.amazon.awssdk.services.dynamodb.model.WriteRequest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchWriteItems {
    public static void main(String[] args){
        final String usage = """
```

```
                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        addBatchItems(dynamoDbClient, tableName);
    }

    public static void addBatchItems(DynamoDbClient dynamoDbClient, String
tableName) {
        // Specify the updates you want to perform.
        List<WriteRequest> writeRequests = new ArrayList<>();

        // Set item 1.
        Map<String, AttributeValue> item1Attributes = new HashMap<>();
        item1Attributes.put("Artist",
AttributeValue.builder().s("Artist1").build());
        item1Attributes.put("Rating", AttributeValue.builder().s("5").build());
        item1Attributes.put("Comments", AttributeValue.builder().s("Great
song!").build());
        item1Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle1").build());

writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item1Attri

        // Set item 2.
        Map<String, AttributeValue> item2Attributes = new HashMap<>();
        item2Attributes.put("Artist",
AttributeValue.builder().s("Artist2").build());
        item2Attributes.put("Rating", AttributeValue.builder().s("4").build());
        item2Attributes.put("Comments", AttributeValue.builder().s("Nice
melody.").build());
        item2Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle2").build());
```

```
 writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item2Attri

        try {
            // Create the BatchWriteItemRequest.
            BatchWriteItemRequest batchWriteItemRequest =
 BatchWriteItemRequest.builder()
                .requestItems(Map.of(tableName, writeRequests))
                .build();

            // Execute the BatchWriteItem operation.
            BatchWriteItemResponse batchWriteItemResponse =
 dynamoDbClient.batchWriteItem(batchWriteItemRequest);

            // Process the response.
            System.out.println("Batch write successful: " +
 batchWriteItemResponse);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Insira vários itens em uma tabela usando o cliente aprimorado.

```
import com.example.dynamodb.Customer;
import com.example.dynamodb.Music;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import
 software.amazon.awssdk.enhanced.dynamodb.model.BatchWriteItemEnhancedRequest;
import software.amazon.awssdk.enhanced.dynamodb.model.WriteBatch;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
```

```
import java.time.ZoneOffset;

/*
 * Before running this code example, create an Amazon DynamoDB table named
 Customer with these columns:
 *   - id - the id of the record that is the key
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnhancedBatchWriteItems {
        public static void main(String[] args) {
                Region region = Region.US_EAST_1;
                DynamoDbClient ddb = DynamoDbClient.builder()
                                .region(region)
                                .build();
                DynamoDbEnhancedClient enhancedClient =
 DynamoDbEnhancedClient.builder()
                                .dynamoDbClient(ddb)
                                .build();
                putBatchRecords(enhancedClient);
                ddb.close();
        }

        public static void putBatchRecords(DynamoDbEnhancedClient enhancedClient)
 {
                try {
                        DynamoDbTable<Customer> customerMappedTable =
 enhancedClient.table("Customer",
                                        TableSchema.fromBean(Customer.class));
                        DynamoDbTable<Music> musicMappedTable =
 enhancedClient.table("Music",
                                        TableSchema.fromBean(Music.class));
                        LocalDate localDate = LocalDate.parse("2020-04-07");
                        LocalDateTime localDateTime = localDate.atStartOfDay();
```

```
                        Instant instant =
localDateTime.toInstant(ZoneOffset.UTC);

                        Customer record2 = new Customer();
                        record2.setCustName("Fred Pink");
                        record2.setId("id110");
                        record2.setEmail("fredp@noserver.com");
                        record2.setRegistrationDate(instant);

                        Customer record3 = new Customer();
                        record3.setCustName("Susan Pink");
                        record3.setId("id120");
                        record3.setEmail("spink@noserver.com");
                        record3.setRegistrationDate(instant);

                        Customer record4 = new Customer();
                        record4.setCustName("Jerry orange");
                        record4.setId("id101");
                        record4.setEmail("jorange@noserver.com");
                        record4.setRegistrationDate(instant);

                        BatchWriteItemEnhancedRequest
batchWriteItemEnhancedRequest = BatchWriteItemEnhancedRequest
                                        .builder()
                                        .writeBatches(

WriteBatch.builder(Customer.class) // add items to the Customer

         // table

.mappedTableResource(customerMappedTable)

.addPutItem(builder -> builder.item(record2))

.addPutItem(builder -> builder.item(record3))

.addPutItem(builder -> builder.item(record4))
                                                                        .build(),

WriteBatch.builder(Music.class) // delete an item from the Music

      // table

.mappedTableResource(musicMappedTable)
```

```
 .addDeleteItem(builder -> builder.key(

       Key.builder().partitionValue(

                        "Famous Band")

                        .build()))
                                                                    .build())
                                      .build();

                         // Add three items to the Customer table and delete one
 item from the Music
                         // table.

 enhancedClient.batchWriteItem(batchWriteItemEnhancedRequest);
                         System.out.println("done");

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
        }
}
```

- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte BatchWrite.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year'
 is recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
```

```
      },
    });

    await docClient.send(command);
  }
};
```

- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
```

```
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK para JavaScript.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
    public function writeBatch(string $TableName, array $Batch, int $depth = 2)
    {
        if (--$depth <= 0) {
            throw new Exception("Max depth exceeded. Please try with fewer batch
 items or increase depth.");
        }

        $marshal = new Marshaler();
```

```
        $total = 0;
        foreach (array_chunk($Batch, 25) as $Items) {
            foreach ($Items as $Item) {
                $BatchWrite['RequestItems'][$TableName][] = ['PutRequest' =>
 ['Item' => $marshal->marshalItem($Item)]];
            }
            try {
                echo "Batching another " . count($Items) . " for a total of " .
 ($total += count($Items)) . " items!\n";
                $response = $this->dynamoDbClient->batchWriteItem($BatchWrite);
                $BatchWrite = [];
            } catch (Exception $e) {
                echo "uh oh...";
                echo $e->getMessage();
                die();
            }
            if ($total >= 250) {
                echo "250 movies is probably enough. Right? We can stop there.
\n";
                break;
            }
        }
    }
```

- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: cria um item ou substitui um item por um novo item nas tabelas Music e Songs do DynamoDB.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
```

```
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

Saída:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Para ver detalhes da API, consulte BatchWriteItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: cria um item ou substitui um item por um novo item nas tabelas Music e Songs do DynamoDB.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

Saída:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Para ver detalhes da API, consulte BatchWriteItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obter detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Fills an Amazon DynamoDB table with the specified data. Items are sent in
  # batches of 25 until all items are written.
  #
  # @param movies [Enumerable] The data to put in the table. Each item must
 contain at least
  #                            the keys required by the schema that was specified
 when the
  #                            table was created.
  def write_batch(movies)
    index = 0
    slice_size = 25
    while index < movies.length
      movie_items = []
      movies[index, slice_size].each do |movie|
        movie_items.append({ put_request: { item: movie } })
      end
      @dynamo_resource.client.batch_write_item({ request_items: { @table.name =>
 movie_items } })
      index += slice_size
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts(
      "Couldn't load data into table #{@table.name}. Here's why:"
```

```
    )
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obter mais detalhes da API, consulte BatchWriteItem na Referência da API AWS SDK para Ruby.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
 jsonData)
```

```
            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }

            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
```

```
    }
```

- Para obter detalhes da API, consulte BatchWriteItem na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`CreateTable`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `CreateTable`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Acelerar leituras com o DAX
- Cenários avançados do índice secundário global
- Criar uma tabela com um índice secundário global
- Criar uma tabela com o throughput a quente habilitado
- Crie e gerencie tabelas globais demonstrando o MREC
- Criar e gerenciar tabelas globais com MRSC
- Gerenciar índices secundários globais
- Gerenciar políticas baseadas em recursos
- Configuração do controle de acesso por atributo
- Trabalhar com tabelas globais e a consistência final de replicação mutirregional (MREC)
- Trabalhar com índices secundários locais
- Trabalhar com a marcação de recursos
- Trabalho com os fluxos e com a vida útil
- Trabalhar com a criptografia de tabelas

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
 -a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
#############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
```

```
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte CreateTable na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
```

```
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como criar uma tabela com tags

O exemplo de `create-table` a seguir usa os atributos especificados e o esquema de chaves para criar uma tabela chamada `MusicCollection`. Essa tabela usa um throughput provisionado e é criptografada em repouso usando a CMK de propriedade padrão da AWS. O comando também aplica uma tag à tabela, com uma chave `Owner` e valor de `blueTeam`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
```

```
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como criar uma tabela no modo sob demanda

O exemplo a seguir cria uma tabela chamada `MusicCollection` usando o modo sob demanda, em vez do modo de throughput provisionado. Esse método é útil para tabelas com workloads imprevisíveis.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 3: como criar uma tabela e criptografá-la com uma CMK gerenciada pelo cliente

O exemplo a seguir cria uma tabela chamada `MusicCollection` e a criptografa usando uma CMK gerenciada pelo cliente.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
```

```
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 4: como criar uma tabela com um índice secundário local

O exemplo a seguir usa os atributos especificados e o esquema de chaves para criar uma tabela chamada `MusicCollection` com um índice secundário local chamado `AlbumTitleIndex`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
```

```
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
```

```
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 5: como criar uma tabela com um índice secundário global

O exemplo a seguir cria uma tabela chamada `GameScores` com um índice secundário global chamado `GameTitleIndex`. A tabela-base tem uma chave de partição `UserId` e uma chave de classificação `GameTitle`, permitindo que você encontre a melhor pontuação de

um usuário individual para um jogo específico de forma eficiente, enquanto o GSI tem uma chave de partição `GameTitle` e uma chave de classificação `TopScore`, permitindo que você encontre rapidamente a pontuação mais alta geral para um jogo específico.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
```

```
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
```

```
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 6: como criar uma tabela com vários índices secundários globais ao mesmo tempo

O exemplo a seguir cria uma tabela chamada `GameScores` com dos índices secundários globais. Os esquemas do GSI são passados por meio de um arquivo, e não pela linha de comando.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Conteúdo de `gsi.json`:

```
[
    {
        "IndexName": "GameTitleIndex",
```

```
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Saída:

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
```

```
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
```

```
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 7: como criar uma tabela com o Streams habilitado

O exemplo a seguir cria uma tabela chamada `GameScores` com o DynamoDB Streams habilitado. Imagens novas e antigas de cada item serão gravadas no fluxo.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
```

```
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

Para obter mais informações, consulte Operações básicas nas tabelas no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 8: como criar uma tabela com o fluxo somente de chaves habilitado

O exemplo a seguir cria uma tabela chamada `GameScores` com o DynamoDB Streams habilitado. Somente os atributos-chave dos itens modificados são gravados no fluxo.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Para obter mais informações, consulte Captura de dados de alterações com o Amazon DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 9: como criar uma tabela com a classe Standard-Infrequent Access

O exemplo a seguir cria uma tabela chamada `GameScores` e atribui a classe de tabela Standard-Infrequent Access (DynamoDB Standard-IA). Essa classe de tabela é otimizada para que o armazenamento seja o custo dominante.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
```

```
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

Para obter mais informações, consulte Classes de tabela no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 10: como criar uma tabela com a proteção contra exclusão habilitada

O exemplo a seguir cria uma tabela chamada `GameScores` e habilita a proteção contra exclusão.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Para obter mais informações, consulte Usar a proteção contra exclusão no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte CreateTable na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
```

```
   public static void main(String[] args) {
       final String usage = """

           Usage:
               <tableName> <key>

           Where:
               tableName - The Amazon DynamoDB table to create (for example,
Music3).
               key - The key for the Amazon DynamoDB table (for example,
Artist).
           """;

       if (args.length != 2) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       System.out.println("Creating an Amazon DynamoDB table " + tableName + "
with a simple primary key: " + key);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
           .region(region)
           .build();

       String result = createTable(ddb, tableName, key);
       System.out.println("New table is " + result);
       ddb.close();
   }

   public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
       DynamoDbWaiter dbWaiter = ddb.waiter();
       CreateTableRequest request = CreateTableRequest.builder()
           .attributeDefinitions(AttributeDefinition.builder()
               .attributeName(key)
               .attributeType(ScalarAttributeType.S)
               .build())
           .keySchema(KeySchemaElement.builder()
               .attributeName(key)
               .keyType(KeyType.HASH)
               .build())
```

```
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
```

```
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
```

```
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma tabela.

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: este exemplo cria uma tabela chamada Thread que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave).

O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemplo 2: este exemplo cria uma tabela chamada “Thread” que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave). Um índice secundário local também é definido. A chave do índice secundário local será definida automaticamente por meio da chave de hash primária na tabela (ForumName). O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
```

```
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemplo 3: este exemplo mostra como usar um único pipeline para criar uma tabela chamada “Thread” que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave) e um índice secundário local. O Add-DDBKeySchema e o Add-DDBIndexSchema criam um objeto TableSchema para você se um não for fornecido pelo pipeline ou pelo parâmetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para ver detalhes da API, consulte CreateTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: este exemplo cria uma tabela chamada Thread que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave). O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Exemplo 2: este exemplo cria uma tabela chamada “Thread” que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave). Um índice secundário local também é definido. A chave do índice secundário local será definida automaticamente por meio da chave de hash primária na tabela (ForumName). O esquema usado para construir a tabela pode ser canalizado para cada cmdlet conforme mostrado ou especificado usando o parâmetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
```

```
LocalSecondaryIndexes  : {LastPostIndex}
```

Exemplo 3: este exemplo mostra como usar um único pipeline para criar uma tabela chamada “Thread” que tem uma chave primária que consiste em “ForumName” (hash do tipo de chave) e “Subject” (intervalo de tipos de chave) e um índice secundário local. O Add-DDBKeySchema e o Add-DDBIndexSchema criam um objeto TableSchema para você se um não for fornecido pelo pipeline ou pelo parâmetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Saída:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para ver detalhes da API, consulte CreateTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma tabela para armazenar dados de filmes.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
```

```
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
```

```
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
```

```
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
      " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
```

```
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DeleteItem`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DeleteItem`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Usar operações condicionais

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };
```

```
            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_delete_item
#
```

```
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
```

```
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
```

```
    echo "$@"
  fi
}

############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
```

```
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte DeleteItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Delete an item from an Amazon DynamoDB table.
/*!
  \sa deleteItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::deleteItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteItemRequest request;

    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));
    request.SetTableName(tableName);
```

```
    const Aws::DynamoDB::Model::DeleteItemOutcome &outcome =
 dynamoClient.DeleteItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item \"" << partitionValue << "\" deleted!" << std::endl;
    }
    else {
        std::cerr << "Failed to delete item: " << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();
```

```
            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como excluir um item

O exemplo de `delete-item` a seguir exclui um item da tabela `MusicCollection` e solicita detalhes sobre o item excluído e a capacidade usada pela solicitação.

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

Saída:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Blue Sky Blues"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Scared of My Shadow"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 2.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como excluir um item de forma condicional

O exemplo a seguir exclui um item da tabela `ProductCatalog` somente se a `ProductCategory` for `Sporting Goods` ou `Gardening Supplies` e o preço estiver entre 500 e 600. Ele retorna detalhes sobre o item que foi excluído.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
```

```
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Conteúdo de `names.json`:

```
{
    "#P": "Price"
}
```

Conteúdo de `values.json`:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

Saída:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte DeleteItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
```

```
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
```

```
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyval>

                Where:
                    tableName - The Amazon DynamoDB table to delete the item from
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to delete
 (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Deleting item \"%s\" from %s\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());
```

```
        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();

        try {
            ddb.deleteItem(deleteReq);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte DeleteCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
```

```
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Exclua um item de uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Exclua um item de uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun deleteDynamoDBItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        DeleteItemRequest {
            tableName = tableNameVal
            key = keyToGet
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
because of your rating...harsh.\n";

    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: remove o item do DynamoDB que corresponde à chave fornecida.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Para ver detalhes da API, consulte DeleteItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: remove o item do DynamoDB que corresponde à chave fornecida.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Para ver detalhes da API, consulte DeleteItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
```

```
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

É possível especificar uma condição para que um item seja excluído somente quando ele atender a determinados critérios.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def delete_underrated_movie(self, title, year, rating):
        """
        Deletes a movie only if it is rated below a specified value. By using a
        condition expression in a delete operation, you can specify that an item
 is
        deleted only when it meets certain criteria.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        :param rating: The rating threshold to check before deleting the movie.
        """
        try:
            self.table.delete_item(
                Key={"year": year, "title": title},
                ConditionExpression="info.rating <= :val",
                ExpressionAttributeValues={":val": Decimal(str(rating))},
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't delete %s because its rating is greater than %s.",
                    title,
                    rating,
                )
            else:
                logger.error(
                    "Couldn't delete movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Deletes a movie from the table.
  #
  # @param title [String] The title of the movie to delete.
  # @param year [Integer] The release year of the movie to delete.
  def delete_item(title, year)
    @table.delete_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete movie #{title}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        DATA(lo_resp) = lo_dyn->deleteitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key_input ).
        MESSAGE 'Deleted one item.' TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte DeleteItem na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB
```

```
    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte DeleteItem na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DeleteTable`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DeleteTable`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Acelerar leituras com o DAX
- Criar e gerenciar tabelas globais com MRSC
- Gerenciar índices secundários globais
- Trabalhar com tabelas globais e a consistência final de replicação mutirregional (MREC)

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
```

```
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para C++.

CLI

AWS CLI

Como excluir uma tabela

O exemplo de `delete-table` a seguir exclui a tabela `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Saída:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Para obter mais informações, consulte Excluir uma tabela no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte DeleteTable na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
```

```
 return err
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).
```

```
                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
```

```
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: exclui a tabela especificada. A confirmação será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable"
```

Exemplo 2: exclui a tabela especificada. A confirmação não será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para ver detalhes da API, consulte DeleteTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: exclui a tabela especificada. A confirmação será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable"
```

Exemplo 2: exclui a tabela especificada. A confirmação não será solicitada antes que a operação continue.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para ver detalhes da API, consulte DeleteTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
```

```
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
```

```
        " Wait till the table is actually deleted.
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte DeleteTable na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
```

```
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte DeleteTable na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DescribeTable`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DescribeTable`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Crie e gerencie tabelas globais demonstrando o MREC
- Criar e gerenciar tabelas globais com MRSC
- Trabalhar com tabelas globais e a consistência final de replicação mutirregional (MREC)
- Trabalho com os fluxos e com a vida útil
- Trabalhar com a criptografia de tabelas

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    private static async Task GetTableInformation()
    {
        Console.WriteLine("\n*** Retrieving table information ***");

        var response = await Client.DescribeTableAsync(new DescribeTableRequest
        {
            TableName = ExampleTableName
        });

        var table = response.Table;
        Console.WriteLine($"Name: {table.TableName}");
        Console.WriteLine($"# of items: {table.ItemCount}");

    }
```

- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
```

```
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
```

```
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte DescribeTable na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Describe an Amazon DynamoDB table.
/*!
  \sa describeTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::describeTable(const Aws::String &tableName,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   Aws::DynamoDB::Model::DescribeTableRequest request;
   request.SetTableName(tableName);

   const Aws::DynamoDB::Model::DescribeTableOutcome &outcome =
 dynamoClient.DescribeTable(
           request);

   if (outcome.IsSuccess()) {
       const Aws::DynamoDB::Model::TableDescription &td =
 outcome.GetResult().GetTable();
       std::cout << "Table name  : " << td.GetTableName() << std::endl;
       std::cout << "Table ARN   : " << td.GetTableArn() << std::endl;
       std::cout << "Status      : "
                 <<
 Aws::DynamoDB::Model::TableStatusMapper::GetNameForTableStatus(
                        td.GetTableStatus()) << std::endl;
       std::cout << "Item count  : " << td.GetItemCount() << std::endl;
       std::cout << "Size (bytes): " << td.GetTableSizeBytes() << std::endl;

       const Aws::DynamoDB::Model::ProvisionedThroughputDescription &ptd =
 td.GetProvisionedThroughput();
       std::cout << "Throughput" << std::endl;
       std::cout << "  Read Capacity : " << ptd.GetReadCapacityUnits() <<
 std::endl;
       std::cout << "  Write Capacity: " << ptd.GetWriteCapacityUnits() <<
 std::endl;

       const Aws::Vector<Aws::DynamoDB::Model::AttributeDefinition> &ad =
 td.GetAttributeDefinitions();
       std::cout << "Attributes" << std::endl;
       for (const auto &a: ad)
           std::cout << "  " << a.GetAttributeName() << " (" <<

 Aws::DynamoDB::Model::ScalarAttributeTypeMapper::GetNameForScalarAttributeType(
                            a.GetAttributeType()) <<
                    ")" << std::endl;
   }
```

```
    else {
        std::cerr << "Failed to describe table: " <<
 outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK para C++.

CLI

AWS CLI

Como descrever uma tabela

O exemplo a seguir de `describe-table` descreve a tabela `MusicCollection`.

```
aws dynamodb describe-table \
    --table-name MusicCollection
```

Saída:

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
```

```
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "ACTIVE",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1421866952.062
    }
}
```

Para obter mais informações, consulte Descrever uma tabela no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte DescribeTable na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import
 software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughputDescription;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information
 about (for example, Music3).
                """;

        if (args.length != 1) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Getting description for %s\n\n", tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        describeDymamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void describeDymamoDBTable(DynamoDbClient ddb, String
tableName) {
        DescribeTableRequest request = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            TableDescription tableInfo = ddb.describeTable(request).table();
            if (tableInfo != null) {
                System.out.format("Table name  : %s\n", tableInfo.tableName());
                System.out.format("Table ARN   : %s\n", tableInfo.tableArn());
                System.out.format("Status      : %s\n", tableInfo.tableStatus());
                System.out.format("Item count  : %d\n", tableInfo.itemCount());
                System.out.format("Size (bytes): %d\n",
tableInfo.tableSizeBytes());

                ProvisionedThroughputDescription throughputInfo =
tableInfo.provisionedThroughput();
                System.out.println("Throughput");
                System.out.format("  Read Capacity : %d\n",
throughputInfo.readCapacityUnits());
                System.out.format("  Write Capacity: %d\n",
throughputInfo.writeCapacityUnits());

                List<AttributeDefinition> attributes =
tableInfo.attributeDefinitions();
                System.out.println("Attributes");
                for (AttributeDefinition a : attributes) {
```

```
                    System.out.format("  %s (%s)\n", a.attributeName(),
 a.attributeType());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("\nDone!");
    }
}
```

- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK para JavaScript.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: exibe detalhes da tabela especificada.

```
Get-DDBTable -TableName "myTable"
```

- Para ver detalhes da API, consulte DescribeTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: exibe detalhes da tabela especificada.

```
Get-DDBTable -TableName "myTable"
```

- Para ver detalhes da API, consulte DescribeTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
```

```
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
```

```
                raise
        else:
            self.table = table
        return exists
```

- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
```

```
rescue Aws::DynamoDB::Errors::ResourceNotFoundException
  @logger.debug("Table #{table_name} doesn't exist")
  false
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't check for existence of #{table_name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- Para obter detalhes da API, consulte DescribeTable na Referência da API AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
TRY.
    oo_result = lo_dyn->describetable( iv_tablename = iv_table_name ).
    DATA(lv_tablename) = oo_result->get_table( )->ask_tablename( ).
    DATA(lv_tablearn) = oo_result->get_table( )->ask_tablearn( ).
    DATA(lv_tablestatus) = oo_result->get_table( )->ask_tablestatus( ).
    DATA(lv_itemcount) = oo_result->get_table( )->ask_itemcount( ).
    MESSAGE 'The table name is ' && lv_tablename
        && '. The table ARN is ' && lv_tablearn
        && '. The tablestatus is ' && lv_tablestatus
        && '. Item count is ' && lv_itemcount TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && lv_tablename && ' does not exist' TYPE 'E'.
ENDTRY.
```

- Para obter detalhes da API, consulte DescribeTable na Referência da API do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DescribeTimeToLive`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DescribeTimeToLive`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Trabalho com os fluxos e com a vida útil

CLI

AWS CLI

Como ver as configurações de vida útil de uma tabela

O exemplo `describe-time-to-live` a seguir exibe as configurações de vida útil da tabela `MusicCollection`.

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

Saída:

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
}
```

Para obter mais informações, consulte Vida útil no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte DescribeTimeToLive na Referência de comandos da AWS CLI.

Java

SDK para Java 2.x

Descreva a configuração de TTL em uma tabela existente do DynamoDB usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.logging.Level;
import java.util.logging.Logger;

    public DescribeTimeToLiveResponse describeTTL(final String tableName, final
 Region region) {
        final DescribeTimeToLiveRequest request =
            DescribeTimeToLiveRequest.builder().tableName(tableName).build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.describeTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte DescribeTimeToLive na Referência de API do AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Descreva a configuração de TTL em uma tabela existente do DynamoDB usando o AWS SDK para JavaScript.

```
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const describeTTL = async (tableName, region) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));

        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED')
 {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- Para obter detalhes da API, consulte DescribeTimeToLive na Referência de API do AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Descreva a configuração de TTL em uma tabela existente do DynamoDB usando o AWS SDK para Python (Boto3).

```
import boto3


def describe_ttl(table_name, region):
    """
    Describes TTL on an existing table, as well as a region.

    :param table_name: String representing the name of the table
    :param region: AWS Region of the table - example `us-east-1`
    :return: Time to live description.
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        ttl_description = dynamodb.describe_time_to_live(TableName=table_name)
        print(
            f"TimeToLive for table {table_name} is status
 {ttl_description['TimeToLiveDescription']['TimeToLiveStatus']}"
        )

        return ttl_description
    except Exception as e:
        print(f"Error describing table: {e}")
        raise


# Enter your own table name and AWS region
describe_ttl("your-table-name", "us-east-1")
```

- Para obter detalhes da API, consulte DescribeTimeToLive na Referência de API do AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`ExecuteStatement`** com um SDK da AWS

Os exemplos de código a seguir mostram como usar o `ExecuteStatement`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Excluir dados usando DELETE do PartiQL
- Inserir dados usando INSERT do PartiQL
- Consultar uma tabela usando o PartiQL
- Consultar dados usando SELECT do PartiQL
- Atualizar dados usando UPDATE do PartiQL

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Use uma instrução INSERT para adicionar um item.

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Use uma instrução SELECT para obter um item.

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });
```

```
            return response.Items;
        }
```

Use uma instrução SELECT para obter uma lista de itens.

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Use uma instrução UPDATE para atualizar um item.

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
```

```
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Use uma instrução DELETE para excluir um único filme.

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Use uma instrução INSERT para adicionar um item.

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
```

```
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        request.SetStatement(sqlStream.str());

        // Create the parameter attributes.
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
```

```
        }
    }
```

Use uma instrução SELECT para obter um item.

```
   //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to retrieve movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
       else {
           // Print the retrieved movie information.
           const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

           if (items.size() == 1) {
               printMovieInfo(items[0]);
           }
           else {
```

```
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }
```

Use uma instrução UPDATE para atualizar um item.

```
    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
        }
```

```
    }
```

Use uma instrução DELETE para excluir um item.

```
    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Defina uma estrutura de receptor de função para o exemplo.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Use uma instrução INSERT para adicionar um item.

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
```

```
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Use uma instrução SELECT para obter um item.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Use uma instrução SELECT para obter uma lista de itens e projetar os resultados.

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

Use uma instrução UPDATE para atualizar um item.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

Use uma instrução DELETE para excluir um item.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
```

```
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
```

```
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Crie um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
```

```
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtenha um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Atualize um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
```

```
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Exclua um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para JavaScript.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
```

```
        ]);
    }
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para PHP.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
```

```
                    statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Selecione um único item usando o PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Gets a single record from a table using PartiQL.
  # Note: To perform more fine-grained selects,
  # use the Client.query instance method instead.
  #
  # @param title [String] The title of the movie to search.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def select_item_by_title(title)
    request = {
      statement: "SELECT * FROM \"#{@table.name}\" WHERE title=?",
      parameters: [title]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Atualize um único item usando o PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Updates a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def update_rating_by_title(title, year, rating)
    request = {
```

```
      statement: "UPDATE \"#{@table.name}\" SET info.rating=? WHERE title=? and
 year=?",
      parameters: [{ "N": rating }, title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Adicione um único item usando o PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Adds a single record to a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param plot [String] The plot of the movie.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def insert_item(title, year, plot, rating)
    request = {
      statement: "INSERT INTO \"#{@table.name}\" VALUE {'title': ?, 'year': ?,
 'info': ?}",
      parameters: [title, year, { 'plot': plot, 'rating': rating }]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Exclua um único item usando o PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
```

```
  @dynamodb = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamodb.table(table_name)
end

# Deletes a single record from a table using PartiQL.
#
# @param title [String] The title of the movie to update.
# @param year [Integer] The year the movie was released.
# @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
def delete_item_by_title(title, year)
  request = {
    statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
    parameters: [title, year]
  }
  @dynamodb.client.execute_statement(request)
end
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para Ruby.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`GetItem`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetItem`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Acelerar leituras com o DAX
- Criar e gerenciar tabelas globais com MRSC
- Trabalhar com tabelas globais e a consistência final de replicação mutirregional (MREC)

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
```

```
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- Para obter detalhes da API, consulte GetItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
```

```
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como ler um item em uma tabela

O exemplo de `get-item` a seguir recupera um item da tabela `MusicCollection`. A tabela tem uma chave primária de hash e intervalo (`Artist` e `SongTitle`), portanto, você deve especificar esses dois atributos. O comando também solicita informações sobre a capacidade de leitura consumida pela operação.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Saída:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obter mais informações, consulte Ler um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como ler um item usando uma leitura consistente

O exemplo a seguir recupera um item da tabela `MusicCollection` usando leituras altamente consistentes.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Saída:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obter mais informações, consulte Ler um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 3: como recuperar atributos específicos de um item

O exemplo a seguir usa uma expressão de projeção para recuperar apenas três atributos do item desejado.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Conteúdo de `names.json`:

```
{
```

```
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Saída:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Para obter mais informações, consulte Ler um item no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte GetItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
```

```
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
```

```
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Obtenha um item de uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
```

```
   public static void main(String[] args) {
       final String usage = """

               Usage:
                   <tableName> <key> <keyVal>

               Where:
                   tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                   key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                   keyval - The key value that represents the item to get (for
example, Famous Band).
               """;

       if (args.length != 3) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       String keyVal = args[2];
       System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
               .region(region)
               .build();

       getDynamoDBItem(ddb, tableName, key, keyVal);
       ddb.close();
   }

   public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
       HashMap<String, AttributeValue> keyToGet = new HashMap<>();
       keyToGet.put(key, AttributeValue.builder()
               .s(keyVal)
               .build());

       GetItemRequest request = GetItemRequest.builder()
               .key(keyToGet)
               .tableName(tableName)
```

```
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Obtenha um item de uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
```

```
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtenha um item de uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: exibe o item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Para ver detalhes da API, consulte GetItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: exibe o item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Para ver detalhes da API, consulte GetItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table
```

```
def initialize(table_name)
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamo_resource.table(table_name)
end

# Gets movie data from the table for a specific movie.
#
# @param title [String] The title of the movie.
# @param year [Integer] The release year of the movie.
# @return [Hash] The data about the requested movie.
def get_item(title, year)
  @table.get_item(key: { 'year' => year, 'title' => title })
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename               = iv_table_name
      it_key                     = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
    MESSAGE 'Movie name is: ' && lo_title->get_s( )
      && 'Movie year is: ' && lo_year->get_n( )
```

```
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte GetItem na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
```

```
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte GetItem na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`ListTables`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `ListTables`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Configuração do controle de acesso por atributo

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
private static async Task ListMyTables()
{
    Console.WriteLine("\n*** Listing tables ***");

    string lastTableNameEvaluated = null;
    do
    {
        var response = await Client.ListTablesAsync(new ListTablesRequest
        {
            Limit = 2,
            ExclusiveStartTableName = lastTableNameEvaluated
        });

        foreach (var name in response.TableNames)
        {
            Console.WriteLine(name);
        }

        lastTableNameEvaluated = response.LastEvaluatedTableName;
    } while (lastTableNameEvaluated != null);
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
##############################################################################
# function dynamodb_list_tables
#
# This function lists all the tables in a DynamoDB.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_list_tables() {
  response=$(aws dynamodb list-tables \
    --output text \
    --query "TableNames")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  echo "$response" | tr -s "[:space:]" "\n"

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
##############################################################################
# function errecho
```

```
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte ListTable na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! List the Amazon DynamoDB tables for the current AWS account.
/*!
  \sa listTables()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::listTables(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
    listTablesRequest.SetLimit(50);
    do {
        const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamoClient.ListTables(
                listTablesRequest);
        if (!outcome.IsSuccess()) {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }

        for (const auto &tableName: outcome.GetResult().GetTableNames())
            std::cout << tableName << std::endl;
        listTablesRequest.SetExclusiveStartTableName(
                outcome.GetResult().GetLastEvaluatedTableName());
```

```
    } while (!listTablesRequest.GetExclusiveStartTableName().empty());

    return true;
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como listar tabelas

O exemplo de `list-tables` a seguir lista todas as tabelas associadas à conta e região atuais da AWS.

```
aws dynamodb list-tables
```

Saída:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Para obter mais informações, consulte Listar nomes de tabela no Guia do desenvolvedor Amazon DynamoDB.

Exemplo 2: como limitar o tamanho da página

O exemplo a seguir retorna uma lista de todas as tabelas existentes, mas recupera apenas um item em cada chamada. Pode ser necessário realizar várias chamadas para obter a lista completa. Limitar o tamanho da página é útil ao executar os comandos da lista em um grande número de recursos, o que pode resultar em um erro de “tempo limite” ao usar o tamanho de página padrão de 1.000.

```
aws dynamodb list-tables \
    --page-size 1
```

Saída:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Para obter mais informações, consulte Listar nomes de tabela no Guia do desenvolvedor Amazon DynamoDB.

Exemplo 3: como limitar o número de itens retornados

O exemplo a seguir limita o número de itens retornados para dois. A resposta inclui um valor `NextToken` a ser usado para recuperar a próxima página de resultados.

```
aws dynamodb list-tables \
    --max-items 2
```

Saída:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

Para obter mais informações, consulte Listar nomes de tabela no Guia do desenvolvedor Amazon DynamoDB.

Exemplo 4: como recuperar a próxima página de resultados

O comando a seguir usa o valor `NextToken` de uma chamada anterior feita ao comando `list-tables` para recuperar outra página de resultados. Como a resposta nesse caso não inclui um valor para `NextToken`, sabemos que chegamos ao final dos resultados.

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

Saída:

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

Para obter mais informações, consulte Listar nomes de tabela no Guia do desenvolvedor Amazon DynamoDB.

- Para obter detalhes da API, consulte ListTable na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
```

```
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun listAllTables() {
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
public function listTables($exclusiveStartTableName = "", $limit = 100)
{
    $this->dynamoDbClient->listTables([
        'ExclusiveStartTableName' => $exclusiveStartTableName,
        'Limit' => $limit,
    ]);
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: exibe detalhes de todas as tabelas, iterando automaticamente até que o serviço indique que não existem mais tabelas.

```
Get-DDBTableList
```

- Para ver detalhes da API, consulte ListTables na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: exibe detalhes de todas as tabelas, iterando automaticamente até que o serviço indique que não existem mais tabelas.

```
Get-DDBTableList
```

- Para ver detalhes da API, consulte ListTables na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Determine se uma tabela existe.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;

    println!("Tables:");

    for name in &table_names {
        println!("  {}", name);
    }

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

Determine se a tabela existe.

```
pub async fn table_exists(client: &Client, table: &str) -> Result<bool, Error> {
    debug!("Checking for table: {table}");
    let table_list = client.list_tables().send().await;

    match table_list {
        Ok(list) => Ok(list.table_names().contains(&table.into())),
        Err(e) => Err(e.into()),
    }
}
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
```

```
        oo_result = lo_dyn->listtables( ).
        " You can loop over the oo_result to get table properties like this.
        LOOP AT oo_result->get_tablenames( ) INTO DATA(lo_table_name).
          DATA(lv_tablename) = lo_table_name->get_value( ).
        ENDLOOP.
        DATA(lv_tablecount) = lines( oo_result->get_tablenames( ) ).
        MESSAGE 'Found ' && lv_tablecount && ' tables' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte ListTables na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Get a list of the DynamoDB tables available in the specified Region.
    ///
    /// - Returns: An array of strings listing all of the tables available
    ///   in the Region specified when the session was created.
    public func getTableList() async throws -> [String] {
        let input = ListTablesInput(
        )
        return try await session.listTables(input: input)
    }
```

- Para obter detalhes da API, consulte ListTables na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`PutItem`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `PutItem`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Acelerar leituras com o DAX
- Cenários avançados do índice secundário global
- Criar um item com TTL
- Crie e gerencie tabelas globais demonstrando o MREC
- Criar e gerenciar tabelas globais com MRSC
- Usar operações condicionais
- Trabalhar com tabelas globais e a consistência final de replicação mutirregional (MREC)
- Trabalho com os fluxos e com a vida útil

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
/// <summary>
/// Adds a new item to the table.
```

```
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }
```

```
while getopts "n:i:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    i) item="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$item" ]]; then
  errecho "ERROR: You must provide an item with the -i parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    item:   $item"
iecho ""
iecho ""

response=$(aws dynamodb put-item \
  --table-name "$table_name" \
  --item file://"$item")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
```

```
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte PutItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
```

```
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
```

```
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
```

```
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como adicionar um item a uma tabela

O exemplo de `put-item` a seguir adiciona um novo item à tabela MusicCollection.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Conteúdo de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Saída:

```
{
```

```
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como substituir condicionalmente um item em uma tabela

O exemplo de `put-item` a seguir substitui um item existente na tabela `MusicCollection` somente se o item existente tiver um atributo `AlbumTitle` com o valor `Greatest Hits`. O comando retorna o valor anterior do item.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Conteúdo de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Conteúdo de `names.json`:

```
{
    "#A": "AlbumTitle"
}
```

Conteúdo de `values.json`:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Saída:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

Se a chave já existir, você verá a seguinte saída:

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte PutItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
```

```
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
```

```
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Coloca um item em uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is
 placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
 Happy Day).
                **Warning** This program will  place an item that you specify
 into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
```

```
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();
```

```
        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Coloque um item em uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};
```

```
// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Coloque um item em uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: cria um item ou substitui um item por um novo item.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Para ver detalhes da API, consulte PutItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: cria um item ou substitui um item por um novo item.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Para ver detalhes da API, consulte PutItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
```

```
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB
```

```
    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
```

```
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Para obter detalhes da API, consulte PutItem na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`Query`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `Query`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Acelerar leituras com o DAX
- Cenários avançados do índice secundário global
- Comparar vários valores com um único atributo
- Gerenciar índices secundários globais

- Executar operações de consulta avançada
- Consultar uma tabela usando uma condição begins_with
- Consulte uma tabela usando um intervalo de datas
- Consultar uma tabela usando um índice secundário global
- Consultar uma tabela com uma expressão de filtro complexa
- Consultar uma tabela com uma expressão de filtro dinâmica
- Consultar uma tabela com uma expressão de filtro e limite
- Consultar uma tabela com atributos aninhados
- Consultar uma tabela com paginação
- Consultar uma tabela com leituras altamente consistentes
- Consultar itens com TTL
- Consultar tabelas usando padrões de data e hora
- Usar nomes de atributo de expressão
- Trabalhar com índices secundários locais

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
```

```
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
```

```
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
```

```
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi
```

```
  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
```

```
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte Query na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::QueryRequest request;

    request.SetTableName(tableName);

    if (!projectionExpression.empty()) {
        request.SetProjectionExpression(projectionExpression);
    }

    // Set query key condition expression.
    request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

    // Set Expression AttributeValues.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
    attributeValues.emplace(":valueToMatch", partitionValue);

    request.SetExpressionAttributeValues(attributeValues);

    bool result = true;

    // "exclusiveStartKey" is used for pagination.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 exclusiveStartKey;
    do {
        if (!exclusiveStartKey.empty()) {
            request.SetExclusiveStartKey(exclusiveStartKey);
            exclusiveStartKey.clear();
```

```
        }
        // Perform Query operation.
        const Aws::DynamoDB::Model::QueryOutcome &outcome =
 dynamoClient.Query(request);
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "Number of items retrieved from Query: " <<
 items.size()
                          << std::endl;
                // Iterate each item and print.
                for (const auto &item: items) {
                    std::cout
                            <<
 "*************************************************"
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como consultar uma tabela

O exemplo da `query` a seguir consulta itens da tabela `MusicCollection`. A tabela tem uma chave primária de hash e intervalo (`Artist` e `SongTitle`), mas essa consulta especifica apenas o valor da chave de hash. Ela retorna nomes de músicas do artista “No One You Know”.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Conteúdo de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
```

```
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como consultar uma tabela usando leituras altamente consistentes e percorrer o índice em ordem decrescente

O exemplo a seguir executa a mesma consulta do primeiro exemplo, mas retorna os resultados na ordem inversa e usa leituras altamente consistentes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Conteúdo de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
```

```
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 3: como filtrar resultados específicos

O exemplo a seguir consulta o `MusicCollection`, mas exclui os resultados com valores específicos no atributo `AlbumTitle`. Observe que isso não afeta `ScannedCount` ou `ConsumedCapacity` já que o filtro é aplicado após a leitura dos itens.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Conteúdo de `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Conteúdo de `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Saída:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 4: como recuperar somente uma contagem de itens

O exemplo a seguir recupera uma contagem de itens que correspondem à consulta, mas não recupera os itens em si.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Conteúdo de `expression-attributes.json`:

```
{
```

```
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 5: como consultar um índice

O exemplo a seguir consulta o índice secundário local `AlbumTitleIndex`. A consulta retorna todos os atributos da tabela base projetados no índice secundário local. Ao consultar um índice secundário local ou global, você deve fornecer o nome da tabela base usando o parâmetro `table-name`.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Conteúdo de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Items": [
```

```
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Para obter mais informações, consulte Operações de consulta no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte Query na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
```

```
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
```

```
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte Query na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Consulta uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
```

```
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
```

```
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Consulta uma tabela usando o `DynamoDbClient` e um índice secundário.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
```

```
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
```

```
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
```

```
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
```

```
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: invoca uma consulta que exibe itens do DynamoDB com SongTitle e Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
```

```
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
```

- Para ver detalhes da API, consulte Query na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: invoca uma consulta que exibe itens do DynamoDB com SongTitle e Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Saída:

```
Name                            Value
----                            -----
Genre                           Country
Artist                          No One You Know
Price                           1.94
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
```

- Para ver detalhes da API, consulte Query na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Consulte itens usando uma expressão de condição de chave.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

Consulte itens e projete-os para retornar um subconjunto de dados.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
```

```
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Encontre os filmes feitos no ano especificado.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
```

```
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obter os detalhes da API, consulte Query na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
```

```
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
```

```
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **Scan** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `Scan`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Acelerar leituras com o DAX
- Comparar vários valores com um único atributo

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
```

```
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
 info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
 LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
```

```
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
```

```
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
```

```
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
```

```
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
```

```
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte Scan na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
//! Scan an Amazon DynamoDB table.
/*!
  \sa scanTable()
  \param tableName: Name for the DynamoDB table.
```

```
 \param projectionExpression: An optional projection expression, ignored if
empty.
 \param clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
*/

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                 const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
&clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
   Aws::DynamoDB::Model::ScanRequest request;
   request.SetTableName(tableName);

   if (!projectionExpression.empty())
       request.SetProjectionExpression(projectionExpression);

   Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
all_items;
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
last_evaluated_key; // Used for pagination;
   do {
       if (!last_evaluated_key.empty()) {
           request.SetExclusiveStartKey(last_evaluated_key);
       }
       const Aws::DynamoDB::Model::ScanOutcome &outcome =
dynamoClient.Scan(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           all_items.insert(all_items.end(), items.begin(), items.end());

           last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
       }
       else {
           std::cerr << "Failed to Scan items: " <<
outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }

   } while (!last_evaluated_key.empty());
```

```
    if (!all_items.empty()) {
        std::cout << "Number of items retrieved from scan: " << all_items.size()
                  << std::endl;
        // Iterate each item and print.
        for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 &itemMap: all_items) {
            std::cout << "*****************************************************"
                      << std::endl;
            // Output each retrieved field and its value.
            for (const auto &itemEntry: itemMap)
                std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
                          << std::endl;
        }
    }

    else {
        std::cout << "No items found in table: " << tableName << std::endl;
    }

    return true;
}
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para C++.

CLI

AWS CLI

Como verificar uma tabela

O exemplo de `scan` faz uma varredura da tabela `MusicCollection` e restringe os resultados a músicas do artista “No One You Know”. Em cada item, somente o nome do álbum e da música são retornados.

```
aws dynamodb scan \
    --table-name MusicCollection \
    --filter-expression "Artist = :a" \
    --projection-expression "#ST, #AT" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json
```

Conteúdo de `expression-attribute-names.json`:

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

Conteúdo de `expression-attribute-values.json`:

```
{
    ":a": {"S": "No One You Know"}
}
```

Saída:

```
{
    "Count": 2,
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "AlbumTitle": {
                "S": "Somewhat Famous"
            }
        },
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            },
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            }
        }
    ],
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

Para obter mais informações, consulte Trabalhar com verificações no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte Scan na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
```

```
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte Scan na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Verifica uma tabela do Amazon DynamoDB usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To scan items from an Amazon DynamoDB table using the AWS SDK for Java V2,
```

```
 * its better practice to use the
 * Enhanced Client, See the EnhancedScanRecords example.
 */

public class DynamoDBScanItems {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information from
 (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        scanItems(ddb, tableName);
        ddb.close();
    }

    public static void scanItems(DynamoDbClient ddb, String tableName) {
        try {
            ScanRequest scanRequest = ScanRequest.builder()
                    .tableName(tableName)
                    .build();

            ScanResponse response = ddb.scan(scanRequest);
            for (Map<String, AttributeValue> item : response.items()) {
                Set<String> keys = item.keySet();
                for (String key : keys) {
                    System.out.println("The key name is " + key + "\n");
                    System.out.println("The value is " + item.get(key).s());
```

```
                }
            }

        } catch (DynamoDbException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte ScanCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
```

```
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values
 you want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK para JavaScript.
- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun scanItems(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
```

```
}
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

    public function scan(string $tableName, array $key, string $filters)
    {
        $query = [
            'ExpressionAttributeNames' => ['#year' => 'year'],
            'ExpressionAttributeValues' => [
                ":min" => ['N' => '1990'],
                ":max" => ['N' => '1999'],
            ],
            'FilterExpression' => "#year between :min and :max",
```

```
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->scan($query);
    }
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: exibe todos os itens da tabela Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Exemplo 2: exibe itens na tabela Music com CriticRating maior ou igual a nove.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
```

```
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Para ver detalhes da API, consulte Scan na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: exibe todos os itens da tabela Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Exemplo 2: exibe itens na tabela Music com CriticRating maior ou igual a nove.

```
$scanFilter = @{
```

```
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Para ver detalhes da API, consulte Scan na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
```

```
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
```

```
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Scans for movies that were released in a range of years.
```

```
  # Uses a projection expression to return a subset of data for each movie.
  #
  # @param year_range [Hash] The range of years to retrieve.
  # @return [Array] The list of movies released in the specified years.
  def scan_items(year_range)
    movies = []
    scan_hash = {
      filter_expression: '#yr between :start_yr and :end_yr',
      projection_expression: '#yr, title, info.rating',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: {
        ':start_yr' => year_range[:start], ':end_yr' => year_range[:end]
      }
    }
    done = false
    start_key = nil
    until done
      scan_hash[:exclusive_start_key] = start_key unless start_key.nil?
      response = @table.scan(scan_hash)
      movies.concat(response.items) unless response.items.empty?
      start_key = response.last_evaluated_key
      done = start_key.nil?
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't scan for movies. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    movies
  end
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
 Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("   {:?}", item);
    }

    Ok(())
}
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        " Scan movies for rating greater than or equal to the rating specified
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_rating }| ) ) ).
        DATA(lt_filter_conditions) = VALUE /aws1/
cl_dyncondition=>tt_filterconditionmap(
```

```
          ( VALUE /aws1/cl_dyncondition=>ts_filterconditionmap_maprow(
          key = 'rating'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |GE|
          ) ) ) ).
        oo_scan_result = lo_dyn->scan( iv_tablename = iv_table_name
          it_scanfilter = lt_filter_conditions ).
        DATA(lt_items) = oo_scan_result->get_items( ).
        LOOP AT lt_items INTO DATA(lo_item).
          " You can loop over to get individual attributes.
          DATA(lo_title) = lo_item[ key = 'title' ]-value.
          DATA(lo_year) = lo_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_scan_result->get_count( ).
        MESSAGE 'Found ' && lv_count && ' items' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obter os detalhes da API, consulte Scan na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
    /// - Parameters:
```

```
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }
```

```
                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }
```

- Para obter detalhes da API, consulte Scan na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`UpdateItem`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `UpdateItem`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Conheça os conceitos básicos
- Atualizar condicionalmente a TTL de um item
- Contar operadores de expressão
- Criar e gerenciar tabelas globais com MRSC
- Executar operações de lista
- Executar operações de mapa
- Executar operações de conjunto
- Como funciona a ordem da expressão de atualização

- Atualiza a TTL de um item
- Usar operações de contador atômico
- Usar operações condicionais
- Usar nomes de atributo de expressão

.NET

SDK para .NET (v4)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
```

```
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para .NET.

Bash

AWS CLI com script Bash

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
##############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
```

```
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
item to update."
    echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
 }

 while getopts "n:k:e:v:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     e) update_expression="${OPTARG}" ;;
     v) values="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
   return 1
 fi
 if [[ -z "$update_expression" ]]; then
   errecho "ERROR: You must provide an update expression with the -e parameter."
   usage
```

```
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

As funções utilitárias usadas neste exemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
```

```
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
```

```
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência de comandos da AWS CLI.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
```

```
                                    const Aws::String &attributeKey,
                                    const Aws::String &attributeValue,
                                    const Aws::Client::ClientConfiguration
&clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   // *** Define UpdateItem request arguments.
   // Define TableName argument.
   Aws::DynamoDB::Model::UpdateItemRequest request;
   request.SetTableName(tableName);

   // Define KeyName argument.
   Aws::DynamoDB::Model::AttributeValue attribValue;
   attribValue.SetS(partitionValue);
   request.AddKey(partitionKey, attribValue);

   // Construct the SET update expression argument.
   Aws::String update_expression("SET #a = :valueA");
   request.SetUpdateExpression(update_expression);

   // Construct attribute name argument.
   Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
   expressionAttributeNames["#a"] = attributeKey;
   request.SetExpressionAttributeNames(expressionAttributeNames);

   // Construct attribute value argument.
   Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
   attributeUpdatedValue.SetS(attributeValue);
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
expressionAttributeValues;
   expressionAttributeValues[":valueA"] = attributeUpdatedValue;
   request.SetExpressionAttributeValues(expressionAttributeValues);

   // Update the item.
   const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
dynamoClient.UpdateItem(
           request);
   if (outcome.IsSuccess()) {
       std::cout << "Item was updated" << std::endl;
   } else {
       std::cerr << outcome.GetError().GetMessage() << std::endl;
       return false;
   }
```

```
    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
```

```
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API do AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como atualizar um item em uma tabela

O exemplo da `update-item` a seguir atualiza um item da tabela `MusicCollection`. Ele adiciona um novo atributo (`Year`) e modifica o atributo `AlbumTitle`. Todos os atributos no item, conforme aparecem após a atualização, são retornados na resposta.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Conteúdo de `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
```

```
}
```

Conteúdo de `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Saída:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
```

```
    }
}
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como atualizar um item de forma condicional

O exemplo a seguir atualiza um item na tabela `MusicCollection`, mas somente se o item existente ainda não tiver um atributo `Year`.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Conteúdo de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Conteúdo de `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Conteúdo de `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Se o item já tiver um atributo `Year`, o DynamoDB retornará saída a seguir.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Para obter mais informações, consulte Gravar um item no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte UpdateItem na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
```

```
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API do AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Atualiza um item em uma tabela usando o DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
```

```
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {
```

```
        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para PHP.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: define o atributo de gênero como “Rap” no item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para ver detalhes da API, consulte UpdateItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: define o atributo de gênero como “Rap” no item do DynamoDB com a chave de partição SongTitle e a chave de classificação Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
```

```
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Saída:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para ver detalhes da API, consulte UpdateItem na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Atualize um item usando uma expressão de atualização.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
```

```
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
```

```
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Atualize um item usando uma expressão de atualização que inclui uma operação aritmética.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Atualize um item somente quando ele atender a determinadas condições.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
```

```
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Attributes"]
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
```

```
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
```

```
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obter os detalhes da API, consulte UpdateItem na Referência da API AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
 nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
```

```
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
```

```
            throw error
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência de API do AWS SDK para Swift.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`UpdateTable`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `UpdateTable`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto nos seguintes exemplos de código:

- Crie e gerencie tabelas globais demonstrando o MREC
- Criar e gerenciar tabelas globais com MRSC
- Gerenciar índices secundários globais
- Atualizar a configuração de throughput a quente de uma tabela
- Trabalhar com tabelas globais e a consistência final de replicação mutirregional (MREC)
- Trabalhar com a criptografia de tabelas

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
//! Update a DynamoDB table.
```

```
/*!
  \sa updateTable()
  \param tableName: Name for the DynamoDB table.
  \param readCapacity: Provisioned read capacity.
  \param writeCapacity: Provisioned write capacity.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::updateTable(const Aws::String &tableName,
                                   long long readCapacity, long long
 writeCapacity,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Updating " << tableName << " with new provisioned throughput
 values"
              << std::endl;
    std::cout << "Read capacity : " << readCapacity << std::endl;
    std::cout << "Write capacity: " << writeCapacity << std::endl;

    Aws::DynamoDB::Model::UpdateTableRequest request;
    Aws::DynamoDB::Model::ProvisionedThroughput provisionedThroughput;

 provisionedThroughput.WithReadCapacityUnits(readCapacity).WithWriteCapacityUnits(
            writeCapacity);

 request.WithProvisionedThroughput(provisionedThroughput).WithTableName(tableName);

    const Aws::DynamoDB::Model::UpdateTableOutcome &outcome =
 dynamoClient.UpdateTable(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated the table." << std::endl;
    } else {
        const Aws::DynamoDB::DynamoDBError &error = outcome.GetError();
        if (error.GetErrorType() == Aws::DynamoDB::DynamoDBErrors::VALIDATION &&
            error.GetMessage().find("The provisioned throughput for the table
 will not change") != std::string::npos) {
            std::cout << "The provisioned throughput for the table will not
 change." << std::endl;
        } else {
            std::cerr << outcome.GetError().GetMessage() << std::endl;
            return false;
```

```
        }
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que aguarda a tabela se tornar ativa.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
```

```
                        << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte UpdateTable na Referência da API AWS SDK para C++.

CLI

AWS CLI

Exemplo 1: como modificar o modo de faturamento de uma tabela

O exemplo `update-table` a seguir aumenta a capacidade de leitura e gravação provisionada na tabela `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
```

```
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T13:18:18.921000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        }
    }
}
```

Para ter mais informações, consulte Updating a Table no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 2: como criar um índice secundário global

O exemplo a seguir adiciona um índice secundário global à tabela `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
```

```
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

Conteúdo de `gsi-updates.json`:

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
```

```
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "Backfilling": false,
```

```
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ]
    }
}
```

Para ter mais informações, consulte Updating a Table no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 3: como habilitar o DynamoDB Streams em uma tabela

O comando a seguir habilita o DynamoDB Streams na tabela `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
```

```
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
```

```
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112"
    }
```

```
}
```

Para ter mais informações, consulte Updating a Table no Guia do desenvolvedor do Amazon DynamoDB.

Exemplo 4: como habilitar a criptografia do lado do servidor

O exemplo a seguir habilita a criptografia do lado do servidor na tabela `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

Saída:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
```

```
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
```

```
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112",
        "SSEDescription": {
            "Status": "UPDATING"
        }
    }
}
```

Para ter mais informações, consulte Updating a Table no Guia do desenvolvedor do Amazon DynamoDB.

- Para ter detalhes da API, consulte UpdateTable na Referência de comandos da AWS CLI.

PowerShell

Ferramentas para PowerShell V4

Exemplo 1: atualiza os valores de throughput provisionado da tabela especificada.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Para ver detalhes da API, consulte UpdateTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V4).

Ferramentas para PowerShell V5

Exemplo 1: atualiza os valores de throughput provisionado da tabela especificada.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Para ver detalhes da API, consulte UpdateTable na Ferramentas da AWS para PowerShell Cmdlet Reference (V5).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`UpdateTimeToLive`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `UpdateTimeToLive`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação em contexto no seguinte exemplo de código:

- Trabalho com os fluxos e com a vida útil

CLI

AWS CLI

Como atualizar as configurações de vida útil de uma tabela

O exemplo `update-time-to-live` a seguir habilita a vida útil na tabela especificada.

```
aws dynamodb update-time-to-live \
```

```
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

Saída:

```
{
    "TimeToLiveSpecification": {
        "Enabled": true,
        "AttributeName": "ttl"
    }
}
```

Para obter mais informações, consulte Vida útil no Guia do desenvolvedor do Amazon DynamoDB.

- Para obter detalhes da API, consulte UpdateTimeToLive na Referência de comandos da AWS CLI.

Java

SDK para Java 2.x

Habilite o TTL em uma tabela existente do DynamoDB usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse enableTTL(final String tableName, final
 String attributeName, final Region region) {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(true)
            .build();
```

```
        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

Desabilite o TTL em uma tabela existente do DynamoDB. usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse disableTTL(
        final String tableName, final String attributeName, final Region region)
 {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(false)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
```

```
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte UpdateTimeToLive na Referência de API do AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Habilite a TTL em uma tabela existente do DynamoDB.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') =>
 {

    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };
```

```
    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Desabilite a TTL em uma tabela existente do DynamoDB.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1')
 => {

    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
```

```
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Para obter detalhes da API, consulte UpdateTimeToLive na Referência de API do AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Habilite a TTL em uma tabela existente do DynamoDB.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
```

```
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": True, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(
                f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}"
            )
        return response
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
enable_ttl("your-table-name", "expireAt")
```

Desabilite a TTL em uma tabela existente do DynamoDB.

```
import boto3


def disable_ttl(table_name, ttl_attribute_name):
    """
    Disables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table being modified
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")
```

```
        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": False, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been disabled successfully.")
        else:
            print(
                f"Failed to disable TTL, status code
 {response['ResponseMetadata']['HTTPStatusCode']}"
            )
    except Exception as ex:
        print("Couldn't disable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
disable_ttl("your-table-name", "expireAt")
```

- Para obter detalhes da API, consulte UpdateTimeToLive na Referência de API do AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Cenários do DynamoDB usando AWS SDKs

Os exemplos de código a seguir mostram como implementar cenários comuns no DynamoDB com AWS SDKs. Estes cenários mostram como realizar tarefas específicas chamando várias funções dentro do DynamoDB ou combinadas com outros Serviços da AWS. Cada cenário inclui um link para o código-fonte completo, onde podem ser encontradas instruções sobre como configurar e executar o código.

Os cenários têm como alvo um nível intermediário de experiência para ajudar você a compreender ações de serviço em contexto.

Exemplos

- Acelerar as leituras do DynamoDB com o DAX usando um AWS SDK
- Trabalhar com cenários avançados do índice secundário global do DynamoDB usando a AWS Command Line Interface v2
- Criar uma aplicação para enviar dados para uma tabela do DynamoDB
- Comparar vários valores com um único atributo no DynamoDB com um SDK da AWS
- Atualizar condicionalmente um item do DynamoDB com TTL usando um SDK da AWS
- Conecte-se a uma instância local do DynamoDB usando um SDK da AWS
- Contar operadores de expressão no DynamoDB com um SDK da AWS
- Criar uma API REST do API Gateway para monitorar dados da COVID-19
- Criar uma aplicação de mensageiro com o Step Functions
- Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos
- Crie uma tabela do DynamoDB com um índice secundário global usando o SDK da AWS.
- Criar uma tabela do DynamoDB com uma configuração de throughput a quente usando um AWS SDK
- Criar uma aplicação Web para monitorar dados do DynamoDB
- Criar uma aplicação de chat websocket com o API Gateway
- Criar um item do DynamoDB com TTL usando um SDK da AWS
- Crie e gerencie tabelas globais do DynamoDB com consistência forte multirregional usando um SDK da AWS.
- Crie e gerencie tabelas globais do DynamoDB demonstrando o MREC usando um SDK da AWS
- Excluir dados do DynamoDB usando declarações DELETE do PartiQL com um SDK da AWS
- Detectar EPI em imagens com o Amazon Rekognition usando um AWS SDK
- Inserir dados do DynamoDB usando declarações INSERT do PartiQL com um SDK da AWS
- Chamar uma função do Lambda em um navegador
- Gerenciar índices secundários globais no DynamoDB usando a AWS Command Line Interface v2
- Gerenciar políticas baseadas em recursos do DynamoDB usando a AWS Command Line Interface v2

- Monitorar a performance do Amazon DynamoDB usando um SDK da AWS
- Executar operações de consulta avançada no DynamoDB usando um SDK da AWS
- Executar operações de lista no DynamoDB com um SDK da AWS
- Executar operações de mapa no DynamoDB com um SDK da AWS
- Executar operações de conjunto no DynamoDB com um SDK da AWS
- Consultar uma tabela do DynamoDB usando lotes de instruções PartiQL e um AWS SDK
- Consultar uma tabela do DynamoDB usando o PartiQL e um AWS SDK
- Consultar uma tabela do DynamoDB usando um índice secundário global por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB usando uma condição begins_with por meio de um SDK da AWS
- Consulte uma tabela do DynamoDB usando um intervalo de datas na chave de classificação por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com uma expressão de filtro complexa por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com uma expressão de filtro dinâmica por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com uma expressão de filtro e limite por meio de um SDK da AWS
- Consultar uma tabela do DynamoDB com atributos aninhados por meio de um SDK da AWS
- Consultar uma tabela no DynamoDB com paginação usando um SDK da AWS
- Consultar uma tabela do DynamoDB com leituras altamente consistentes usando um SDK da AWS
- Consultar dados do DynamoDB usando declarações SELECT do PartiQL com um SDK da AWS
- Consultar itens com TTL em uma tabela do DynamoDB usando um SDK da AWS
- Consultar tabelas do DynamoDB usando padrões de data e hora com um SDK da AWS
- Salvar o EXIF e outras informações de imagem usando um AWS SDK
- Configurar o controle de acesso por atributo para o DynamoDB usando a AWS Command Line Interface v2
- Com funciona a ordem da expressão de atualização no DynamoDB com um SDK da AWS
- Atualizar uma configuração de tabela do DynamoDB com throughput a quente usando um AWS SDK
- Atualizar um item do DynamoDB com TTL usando um SDK da AWS

- Atualizar dados do DynamoDB usando declarações UPDATE do PartiQL com um SDK da AWS
- Usar o API Gateway para invocar uma função do Lambda
- Usar Step Functions para invocar funções do Lambda
- Usar um modelo de documento para o DynamoDB usando um AWS SDK
- Usar um modelo de persistência de objetos de alto nível para o DynamoDB usando um AWS SDK
- Usar operações de contador atômico no DynamoDB com um SDK da AWS
- Usar operações condicionais no DynamoDB com um SDK da AWS
- Usar nomes de atributo de expressão no DynamoDB com um SDK da AWS
- Usar eventos programados para invocar uma função do Lambda
- Trabalhar com índices secundários locais do DynamoDB usando a AWS Command Line Interface v2
- Trabalhar com fluxos e vida útil do DynamoDB usando a AWS Command Line Interface v2
- Trabalhar com tabelas globais do DynamoDB e a consistência final de replicação mutirregional (MREC) usando AWS Command Line Interface v2
- Trabalhar com a marcação de recursos do DynamoDB usando a AWS Command Line Interface v2
- Trabalhar com a criptografia de tabelas do DynamoDB usando a AWS Command Line Interface v2

Acelerar as leituras do DynamoDB com o DAX usando um AWS SDK

O código de exemplo a seguir mostra como:

- Criar e gravar dados em uma tabela com os clientes DAX e SDK.
- Obter, consultar e verificar a tabela com ambos os clientes e comparar a respectiva performance.

Para obter mais informações, consulte Desenvolver com o cliente do DynamoDB Accelerator.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma tabela com o cliente do DAX ou Boto3.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

Grave dados de teste na tabela.

```
import boto3
```

```
def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate
 the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

Obtenha itens para várias iterações associadas ao cliente do DAX e ao cliente do Boto3 e relate o tempo gasto para cada um.

```
import argparse
```

```
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each
 iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(
                    Key={"partition_key": partition_key, "sort_key": sort_key}
                )
                print(".", end="")
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
```

```
    )
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the DAX client."
        )
        # Use a with statement so the DAX client closes the cluster after
completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax
            )
    else:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the Boto3 client."
        )
        test_start, test_end = get_item_test(test_key_count, test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/ test_iterations}."
    )
```

Consulte a tabela para várias iterações associadas ao cliente do DAX e ao cliente do Boto3 e relate o tempo gasto para cada um.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
```

```
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this
 value.
    :param sort_keys: The range of sort key values for the query. The query
 returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
```

```
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX
client.")
        # Use a with statement so the DAX client closes the cluster after
completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations,
dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3
client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Verifique a tabela para várias iterações associadas ao cliente do DAX e ao cliente do Boto3 e relate o tempo gasto para cada um.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
```

```
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Exclua a tabela.

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Python (Boto3).
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Query
  - Scan

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Trabalhar com cenários avançados do índice secundário global do DynamoDB usando a AWS Command Line Interface v2

O exemplo de código a seguir mostra como trabalhar com configurações de índice secundário global (GSI).

- Crie uma tabela com vários GSIs.
- Crie uma tabela com capacidade e GSI sob demanda.
- Insira itens em uma tabela com vários GSIs.
- Consulte vários GSIs com condições diferentes.

Bash

AWS CLI com script Bash

Crie uma tabela com vários GSIs.

```
# Create a table with multiple GSIs
aws dynamodb create-table \
    --table-name MusicLibrary \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
        AttributeName=Year,AttributeType=N \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumIndex\",
                \"KeySchema\": [{\"AttributeName\":\"AlbumTitle\",\"KeyType\":
\"HASH\"}],
```

```
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            },
            {
                \"IndexName\": \"GenreYearIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Year\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Artist\",\"SongTitle\"]}
            }
        ]"
```

Crie uma tabela com capacidade e GSI sob demanda.

```
# Create a table with on-demand capacity and GSI
aws dynamodb create-table \
    --table-name MusicOnDemand \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GenreIndex\",
                \"KeySchema\": [{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH
\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            }
        ]"
```

Insira itens em uma tabela com vários GSIs.

```
# Add items to MusicLibrary table
aws dynamodb put-item \
    --table-name MusicLibrary \
```

```
    --item '{
        "Artist": {"S": "The Beatles"},
        "SongTitle": {"S": "Hey Jude"},
        "AlbumTitle": {"S": "Past Masters"},
        "Genre": {"S": "Rock"},
        "Year": {"N": "1968"}
    }'

aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "Miles Davis"},
        "SongTitle": {"S": "So What"},
        "AlbumTitle": {"S": "Kind of Blue"},
        "Genre": {"S": "Jazz"},
        "Year": {"N": "1959"}
    }'
```

Consulte os itens de uma tabela com vários GSIs.

```
# Query the AlbumIndex GSI
echo "Querying AlbumIndex GSI:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Kind of Blue"}}'

# Query the GenreYearIndex GSI with a range condition
echo "Querying GenreYearIndex GSI with range condition:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name GenreYearIndex \
    --key-condition-expression "Genre = :genre AND #yr > :year" \
    --expression-attribute-names '{"#yr": "Year"}' \
    --expression-attribute-values '{":genre":{"S":"Rock"},":year":{"N":"1965"}}'
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable

- PutItem
- Query

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação para enviar dados para uma tabela do DynamoDB

Os exemplos de código a seguir mostram como criar uma aplicação que envia dados para uma tabela do Amazon DynamoDB e notifica você quando um usuário atualiza a tabela.

Java

SDK para Java 2.x

Mostra como criar uma aplicação Web dinâmica que envia dados usando a API Java do Amazon DynamoDB e envia uma mensagem de texto usando a API Java do Amazon Simple Notification Service.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Este exemplo mostra como criar uma aplicação que permite que os usuários enviem dados para uma tabela do Amazon DynamoDB e enviem uma mensagem de texto ao administrador usando o Amazon Simple Notification Service (Amazon SNS).

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK para JavaScript v3.

Serviços usados neste exemplo

- DynamoDB
- Amazon SNS

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Comparar vários valores com um único atributo no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como comparar vários valores com um único atributo no DynamoDB.

- Use o operador IN para comparar vários valores com um único atributo.
- Compare o operador IN com várias condições OR.
- Saiba quais são os benefícios de desempenho e complexidade de expressão oferecidos pelo IN.

Java

SDK para Java 2.x

Compare vários valores com um único atributo no DynamoDB usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Locale;
import java.util.Map;

    /**
     * Queries a table using the IN operator to compare multiple values with a
 single attribute.
     *
     * <p>This method demonstrates how to use the IN operator in a filter
 expression
     * to match an attribute against multiple values.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareMultipleValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
```

```
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Queries a table using multiple OR conditions to compare multiple values
with a single attribute.
     *
     * <p>This method demonstrates the alternative approach to using the IN
operator,
     * by using multiple OR conditions.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareWithOrConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {
```

```
        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for OR conditions
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the OR conditions
        StringBuilder orConditions = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                orConditions.append(" OR ");
            }
            orConditions.append("#attrName = :val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression(orConditions.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Compares the performance of using the IN operator versus multiple OR
conditions.
     *
     * <p>This method demonstrates the performance difference between using the
IN operator
     * and using multiple OR conditions.
```

```
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return Map containing the performance comparison results
     */
    public static Map<String, Object> comparePerformance(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Measure performance of IN operator
            long inStartTime = System.nanoTime();
            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long inEndTime = System.nanoTime();
            long inDuration = inEndTime - inStartTime;

            // Measure performance of OR conditions
            long orStartTime = System.nanoTime();
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long orEndTime = System.nanoTime();
            long orDuration = orEndTime - orStartTime;

            // Record results
            results.put("inOperatorDuration", inDuration);
            results.put("orConditionsDuration", orDuration);
            results.put("inOperatorItems", inResponse.count());
            results.put("orConditionsItems", orResponse.count());
            results.put("inOperatorExpression", "IN operator with " +
valuesList.size() + " values");
```

```
            results.put("orConditionsExpression", valuesList.size() + " OR
conditions");
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Scans a table using the IN operator with a large number of values.
     *
     * <p>This method demonstrates how to use the IN operator with a large number
of values,
     * which can help stay within the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The scan response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithLargeInClause(
        DynamoDbClient dynamoDbClient, String tableName, String attributeName,
List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
```

```
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }

    /**
     * Generates a list of sample values for testing.
     *
     * <p>Helper method to generate a list of sample values for testing.
     *
     * @param valueType The type of values to generate (string, number, or
boolean)
     * @param count The number of values to generate
     * @return List of generated attribute values
     */
    public static List<AttributeValue> generateSampleValues(String valueType, int
count) {
        List<AttributeValue> values = new ArrayList<>();

        for (int i = 0; i < count; i++) {
            AttributeValue value;

            switch (valueType.toLowerCase(Locale.ROOT)) {
                case "string":
                    value = AttributeValue.builder().s("Value" + i).build();
                    break;
                case "number":
                    value =
AttributeValue.builder().n(String.valueOf(i)).build();
                    break;
```

```
                case "boolean":
                    value = AttributeValue.builder().bool(i % 2 == 0).build();
                    break;
                default:
                    throw new IllegalArgumentException("Unsupported value type: "
+ valueType);
            }

            values.add(value);
        }

        return values;
    }
```

Exemplo de uso da comparação de vários valores com o AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        System.out.println("Demonstrating how to compare multiple values with a
single attribute in DynamoDB");

        try {
            // Example 1: Using the IN operator
            System.out.println("\nExample 1: Using the IN operator");
            List<AttributeValue> categories = List.of(
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Computers").build(),
                AttributeValue.builder().s("Accessories").build());

            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            System.out.println("Found " + inResponse.count() + " items using IN
operator");
            System.out.println("Items: " + inResponse.items());

            // Example 2: Using multiple OR conditions
```

```
            System.out.println("\nExample 2: Using multiple OR conditions");
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            System.out.println("Found " + orResponse.count() + " items using OR
conditions");
            System.out.println("Items: " + orResponse.items());

            // Example 3: Performance comparison
            System.out.println("\nExample 3: Performance comparison");
            Map<String, Object> perfComparison = comparePerformance(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            if ((boolean) perfComparison.get("success")) {
                System.out.println("IN operator duration: " +
perfComparison.get("inOperatorDuration") + " ns");
                System.out.println("OR conditions duration: " +
perfComparison.get("orConditionsDuration") + " ns");
                System.out.println("IN operator found " +
perfComparison.get("inOperatorItems") + " items");
                System.out.println("OR conditions found " +
perfComparison.get("orConditionsItems") + " items");
                System.out.println("Expression complexity comparison:");
                System.out.println("  IN operator: " +
perfComparison.get("inOperatorExpression"));
                System.out.println("  OR conditions: " +
perfComparison.get("orConditionsExpression"));
            } else {
                System.out.println("Performance comparison failed: " +
perfComparison.get("error"));
            }

            // Example 4: Using IN with a large number of values
```

```
            System.out.println("\nExample 4: Using IN with a large number of
values");
            List<AttributeValue> productIds = generateSampleValues("string", 20);

            ScanResponse largeInResponse = scanWithLargeInClause(dynamoDbClient,
tableName, "ProductId", productIds);

            System.out.println(
                "Found " + largeInResponse.count() + " items using IN with " +
productIds.size() + " values");

            // Explain the benefits of using IN
            System.out.println("\nKey points about using the IN operator in
DynamoDB:");
            System.out.println("1. The IN operator allows comparing a single
attribute against multiple values");
            System.out.println("2. IN is more concise than using multiple OR
conditions");
            System.out.println("3. IN counts as only 1 operator regardless of the
number of values");
            System.out.println("4. Multiple OR conditions count as 1 operator per
condition plus 1 per OR");
            System.out.println("5. Using IN helps stay within the 300 operator
limit for complex expressions");
            System.out.println("6. IN can be used in filter expressions and
condition expressions");
            System.out.println("7. The IN operator supports up to 100 comparison
values");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.
  - Query
  - Scan

JavaScript

SDK para JavaScript (v3)

Compare vários valores com um único atributo usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 attribute
 * against multiple possible values, which is more efficient than using multiple
 OR conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
```

```
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
`:val${index}`).join(', ')})`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
```

```
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
```

```
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
```

```
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val
${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);
```

```
    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}
```

Exemplo de uso da comparação de vários valores com o AWS SDK para JavaScript.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
  const valuesList = ["Electronics", "Computers", "Accessories"];

  console.log(`Searching for products in any of these categories:
 ${valuesList.join(', ')}`);

  try {
    // Using the IN operator (recommended approach)
    console.log("\nApproach 1: Using the IN operator");
    const response = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response.Count} products in the specified categories`);

    // Using multiple OR conditions (alternative approach)
    console.log("\nApproach 2: Using multiple OR conditions");
    const response2 = await compareWithOrConditions(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response2.Count} products in the specified categories`);

    // Example with a query operation
    console.log("\nQuerying a specific manufacturer's products in multiple
 categories");
```

```
    const partitionKeyName = "Manufacturer";
    const partitionKeyValue = "Acme";

    const response3 = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList,
      partitionKeyName,
      partitionKeyValue
    );

    console.log(`Found ${response3.Count} Acme products in the specified
 categories`);

    // Explain the benefits of using the IN operator
    console.log("\nBenefits of using the IN operator:");
    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - Query
  - Scan

Python

SDK para Python (Boto3)

Compare vários valores com um único atributo usando o AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key
```

```
from typing import Any, Dict, List, Optional


def compare_multiple_values(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.

    This function demonstrates the use of the IN operator to compare a single
 attribute
    against multiple possible values, which is more efficient than using multiple
 OR conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        attribute_name (str): The name of the attribute to compare against the
 values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
 attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
 query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create the filter expression using the is_in method
    filter_expression = Attr(attribute_name).is_in(values_list)

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
```

```
            KeyConditionExpression=key_condition,
 FilterExpression=filter_expression
        )
    else:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}


def compare_with_or_conditions(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Alternative implementation using multiple OR conditions instead of the IN
 operator.

    This function is provided for comparison to show why using the IN operator is
 preferable.
    With many values, this approach becomes verbose and less efficient.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        attribute_name (str): The name of the attribute to compare against the
values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create a filter expression with multiple OR conditions
    filter_expression = None
    for value in values_list:
        condition = Attr(attribute_name).eq(value)
        if filter_expression is None:
            filter_expression = condition
        else:
            filter_expression = filter_expression | condition

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value and filter_expression:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
            KeyConditionExpression=key_condition,
FilterExpression=filter_expression
        )
    elif filter_expression:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)
    else:
        # Return empty response if no values provided
        return {"Items": [], "Count": 0}

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
```

```
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}
```

Exemplo de uso da comparação de vários valores com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use the compare_multiple_values function."""
    # Example parameters
    table_name = "Products"
    attribute_name = "Category"
    values_list = ["Electronics", "Computers", "Accessories"]

    print(f"Searching for products in any of these categories: {values_list}")

    # Using the IN operator (recommended approach)
    print("\nApproach 1: Using the IN operator")
    response = compare_multiple_values(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )

    print(f"Found {response['Count']} products in the specified categories")

    # Using multiple OR conditions (alternative approach)
    print("\nApproach 2: Using multiple OR conditions")
    response2 = compare_with_or_conditions(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )
```

```
    print(f"Found {response2['Count']} products in the specified categories")

    # Example with a query operation
    print("\nQuerying a specific manufacturer's products in multiple categories")
    partition_key_name = "Manufacturer"
    partition_key_value = "Acme"

    response3 = compare_multiple_values(
        table_name=table_name,
        attribute_name=attribute_name,
        values_list=values_list,
        partition_key_name=partition_key_name,
        partition_key_value=partition_key_value,
    )

    print(f"Found {response3['Count']} Acme products in the specified
categories")

    # Explain the benefits of using the IN operator
    print("\nBenefits of using the IN operator:")
    print("1. More concise expression compared to multiple OR conditions")
    print("2. Better readability and maintainability")
    print("3. Potentially better performance with large value lists")
    print("4. Simpler code that's less prone to errors")
    print("5. Easier to modify when adding or removing values")
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de API do AWS SDK para Python (Boto3).
  - Query
  - Scan

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Atualizar condicionalmente um item do DynamoDB com TTL usando um SDK da AWS

Os exemplos de código a seguir mostram como atualizar condicionalmente a TTL de um item.

Java

SDK para Java 2.x

Atualize a TTL em um item do DynamoDB existente em uma tabela, com uma condição.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
```

```
                sortKey - The name of the sort key. Also known as the range
attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String UPDATED_AT_ATTR = "updatedAt";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
    private static final String CONDITION_EXPRESSION = "attribute_exists(" +
PRIMARY_KEY_ATTR + ")";
    private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
    private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs an UpdateTTLConditional with a default DynamoDB client.
     */
    public UpdateTTLConditional() {
        this.dynamoDbClient = null;
    }

    /**
     * Constructs an UpdateTTLConditional with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
```

```
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());
```

```
        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Atualize a TTL em um item do DynamoDB existente em uma tabela, com uma condição.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
```

```
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Atualize a TTL em um item do DynamoDB existente em uma tabela, com uma condição.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"
```

```
        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Conecte-se a uma instância local do DynamoDB usando um SDK da AWS

O exemplo de código a seguir mostra como substituir uma URL de endpoint para se conectar a uma implantação de desenvolvimento local do DynamoDB e de um SDK da AWS.

Para obter mais informações, consulte DynamoDB Local.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
/// Lists your tables from a local DynamoDB instance by setting the SDK Config's
/// endpoint_url and test_credentials.
#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();

    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .test_credentials()
        // DynamoDB run locally uses port 8000 by default.
        .endpoint_url("http://localhost:8000")
        .load()
        .await;
    let dynamodb_local_config =
 aws_sdk_dynamodb::config::Builder::from(&config).build();

    let client = aws_sdk_dynamodb::Client::from_conf(dynamodb_local_config);

    let list_resp = client.list_tables().send().await;
    match list_resp {
        Ok(resp) => {
            println!("Found {} tables", resp.table_names().len());
            for name in resp.table_names() {
                println!("  {}", name);
            }
```

```
        }
        Err(err) => eprintln!("Failed to list local dynamodb tables: {err:?}"),
    }
}
```

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Contar operadores de expressão no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como contar operadores de expressão no DynamoDB.

- Entenda o limite de 300 operadores do DynamoDB.
- Conte operadores em expressões complexas.
- Otimize as expressões para permanecer dentro dos limites.

Java

SDK para Java 2.x

Demonstre a contagem de operadores de expressão usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

    /**
     * Creates a complex filter expression with a specified number of conditions.
     *
     * <p>This method demonstrates how to generate a complex expression with
     * a specific number of operators to test the 300 operator limit.
```

```
    *
    * @param conditionsCount Number of conditions to include
    * @param useAnd Whether to use AND (true) or OR (false) between conditions
    * @return Map containing the filter expression, attribute values, and
operator count
    */
   public static Map<String, Object> createComplexFilterExpression(int
conditionsCount, boolean useAnd) {
       // Initialize the expression parts and attribute values
       StringBuilder filterExpression = new StringBuilder();
       Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

       // Generate the specified number of conditions
       for (int i = 0; i < conditionsCount; i++) {
           // Add the operator between conditions (except for the first one)
           if (i > 0) {
               filterExpression.append(useAnd ? " AND " : " OR ");
           }

           // Alternate between different comparison operators for variety
           String valueKey = ":val" + i;

           switch (i % 5) {
               case 0:
                   filterExpression.append("attribute").append(i).append(" =
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey, AttributeValue.builder().s("value" +
i).build());
                   break;
               case 1:
                   filterExpression.append("attribute").append(i).append(" >
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                   break;
               case 2:
                   filterExpression.append("attribute").append(i).append(" <
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey,
                       AttributeValue.builder().n(String.valueOf(i *
10)).build());
```

```
                    break;
                case 3:
                    filterExpression
                        .append("contains(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("substring" +
i).build());
                    break;
                case 4:
                    filterExpression
                        .append("attribute_exists(attribute")
                        .append(i)
                        .append(")");
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each condition has 1 operator (=, >, <, contains, attribute_exists)
        // Each AND or OR between conditions is 1 operator
        int operatorCount = conditionsCount + (conditionsCount > 0 ?
conditionsCount - 1 : 0);

        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("filterExpression", filterExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Creates a complex update expression with a specified number of operations.
     *
```

```
    * <p>This method demonstrates how to generate a complex update expression
with
    * a specific number of operators to test the 300 operator limit.
    *
    * @param operationsCount Number of operations to include
    * @return Map containing the update expression, attribute values, and
operator count
    */
   public static Map<String, Object> createComplexUpdateExpression(int
operationsCount) {
       // Initialize the expression parts and attribute values
       StringBuilder updateExpression = new StringBuilder("SET ");
       Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

       // Generate the specified number of SET operations
       for (int i = 0; i < operationsCount; i++) {
           // Add comma between operations (except for the first one)
           if (i > 0) {
               updateExpression.append(", ");
           }

           // Alternate between different types of SET operations
           String valueKey = ":val" + i;

           switch (i % 3) {
               case 0:
                   // Simple assignment (1 operator: =)
                   updateExpression.append("attribute").append(i).append(" =
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey, AttributeValue.builder().s("value" +
i).build());
                   break;
               case 1:
                   // Addition (2 operators: = and +)
                   updateExpression
                       .append("attribute")
                       .append(i)
                       .append(" = attribute")
                       .append(i)
                       .append(" + ")
                       .append(valueKey);
                   expressionAttributeValues.put(
```

```
                        valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                    break;
                case 2:
                    // Conditional assignment with if_not_exists (2 operators: =
and if_not_exists)
                    updateExpression
                        .append("attribute")
                        .append(i)
                        .append(" = if_not_exists(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each operation has 1-2 operators as noted above
        int operatorCount = 0;
        for (int i = 0; i < operationsCount; i++) {
            operatorCount += (i % 3 == 0) ? 1 : 2;
        }

        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("updateExpression", updateExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
```

```
    * Test the operator limit by attempting an operation with a complex
expression.
    *
    * <p>This method demonstrates what happens when an expression approaches or
    * exceeds the 300 operator limit.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param key The key of the item to update
    * @param operatorCount Target number of operators to include
    * @return Map containing the result of the operation attempt
    */
   public static Map<String, Object> testOperatorLimit(
       DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int operatorCount) {

       // Create a complex update expression with the specified operator count
       Map<String, Object> expressionData =
           createComplexUpdateExpression((int) Math.ceil(operatorCount /
1.5)); // Adjust to get close to target count

       String updateExpression = (String)
expressionData.get("updateExpression");
       @SuppressWarnings("unchecked")
       Map<String, AttributeValue> expressionAttributeValues =
           (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
       int actualCount = (int) expressionData.get("operatorCount");

       System.out.println("Generated update expression with approximately " +
actualCount + " operators");

       // Define the update parameters
       UpdateItemRequest request = UpdateItemRequest.builder()
           .tableName(tableName)
           .key(key)
           .updateExpression(updateExpression)
           .expressionAttributeValues(expressionAttributeValues)
           .returnValues("UPDATED_NEW")
           .build();

       try {
           // Attempt the update operation
           UpdateItemResponse response = dynamoDbClient.updateItem(request);
```

```
            Map<String, Object> result = new HashMap<>();
            result.put("success", true);
            result.put("message", "Operation succeeded with " + actualCount + "
operators");
            result.put("data", response);
            return result;

        } catch (DynamoDbException e) {
            // Check if the error is due to exceeding the operator limit
            if (e.getMessage().contains("too many operators")) {
                Map<String, Object> result = new HashMap<>();
                result.put("success", false);
                result.put("message", "Operation failed: " + e.getMessage());
                result.put("operatorCount", actualCount);
                return result;
            }

            // Return other errors
            Map<String, Object> result = new HashMap<>();
            result.put("success", false);
            result.put("message", "Operation failed: " + e.getMessage());
            result.put("error", e);
            return result;
        }
    }

    /**
     * Break down a complex expression into multiple simpler operations.
     *
     * <p>This method demonstrates how to handle expressions that would exceed
     * the 300 operator limit by breaking them into multiple operations.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param totalOperations Total number of operations to perform
     * @return Map containing the results of the operations
     */
    public static Map<String, Object> breakDownComplexExpression(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int totalOperations) {

        // Calculate how many operations we can safely include in each batch
```

```
        // Using 150 as a conservative limit (well below 300)
        final int operationsPerBatch = 100;
        final int batchCount = (int) Math.ceil((double) totalOperations /
operationsPerBatch);

        System.out.println("Breaking down " + totalOperations + " operations into
" + batchCount + " batches");

        Map<String, Object> results = new HashMap<>();
        results.put("totalBatches", batchCount);

        Map<Integer, Map<String, Object>> batchResults = new HashMap<>();

        // Process each batch
        for (int batch = 0; batch < batchCount; batch++) {
            // Calculate the operations for this batch
            int batchStart = batch * operationsPerBatch;
            int batchEnd = Math.min(batchStart + operationsPerBatch,
totalOperations);
            int batchSize = batchEnd - batchStart;

            System.out.println(
                "Processing batch " + (batch + 1) + "/" + batchCount + " with " +
batchSize + " operations");

            // Create an update expression for this batch
            Map<String, Object> expressionData =
createComplexUpdateExpression(batchSize);

            String updateExpression = (String)
expressionData.get("updateExpression");
            @SuppressWarnings("unchecked")
            Map<String, AttributeValue> expressionAttributeValues =
                (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
            int operatorCount = (int) expressionData.get("operatorCount");

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression(updateExpression)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues("UPDATED_NEW")
```

```
                .build();

            try {
                // Perform the update operation for this batch
                UpdateItemResponse response = dynamoDbClient.updateItem(request);

                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", true);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("attributes", response.attributes());

                batchResults.put(batch, batchResult);

            } catch (DynamoDbException e) {
                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", false);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("error", e.getMessage());

                batchResults.put(batch, batchResult);

                // Continue with next batch instead of breaking
                continue;
            }
        }

        results.put("results", batchResults);
        return results;
    }

    /**
     * Count operators in a DynamoDB expression based on the rules in the
documentation.
     *
     * <p>This method demonstrates how operators are counted according to the
     * DynamoDB documentation.
     *
     * @param expression The DynamoDB expression to analyze
     * @return Map containing the breakdown of operator counts
     */
    public static Map<String, Integer> countOperatorsInExpression(String
expression) {
```

```
        // Initialize counters for different operator types
        Map<String, Integer> counts = new HashMap<>();
        counts.put("comparisonOperators", 0);
        counts.put("logicalOperators", 0);
        counts.put("functions", 0);
        counts.put("arithmeticOperators", 0);
        counts.put("specialOperators", 0);
        counts.put("total", 0);

        // Count comparison operators (=, <>, <, <=, >, >=)
        // This is a simplified approach and may not catch all cases
        int comparisonCount = 0;
        Pattern comparisonPattern = Pattern.compile("(=|<>|<=|>=|<|>)");
        Matcher comparisonMatcher = comparisonPattern.matcher(expression);
        while (comparisonMatcher.find()) {
            comparisonCount++;
        }
        counts.put("comparisonOperators", comparisonCount);

        // Count logical operators (AND, OR, NOT)
        int andCount = countOccurrences(expression, "\\bAND\\b");
        int orCount = countOccurrences(expression, "\\bOR\\b");
        int notCount = countOccurrences(expression, "\\bNOT\\b");
        counts.put("logicalOperators", andCount + orCount + notCount);

        // Count functions (attribute_exists, attribute_not_exists,
 attribute_type, begins_with, contains, size)
        int functionCount = countOccurrences(
            expression,
            "\\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\\(");
        counts.put("functions", functionCount);

        // Count arithmetic operators (+ and -)
        // This is a simplified approach and may not catch all cases
        int arithmeticCount = 0;
        Pattern arithmeticPattern = Pattern.compile("[a-zA-Z0-9_)\\]]\\s*[\\+\
\-]\\s*[a-zA-Z0-9_:(]");
        Matcher arithmeticMatcher = arithmeticPattern.matcher(expression);
        while (arithmeticMatcher.find()) {
            arithmeticCount++;
        }
        counts.put("arithmeticOperators", arithmeticCount);
```

```
        // Count special operators (BETWEEN, IN)
        int betweenCount = countOccurrences(expression, "\\bBETWEEN\\b");
        int inCount = countOccurrences(expression, "\\bIN\\b");
        counts.put("specialOperators", betweenCount + inCount);

        // Add extra operators for BETWEEN (each BETWEEN includes an AND)
        int currentLogicalOps = counts.getOrDefault("logicalOperators", 0);
        counts.put("logicalOperators", currentLogicalOps + betweenCount);

        // Calculate total
        int total = counts.getOrDefault("comparisonOperators", 0)
            + counts.getOrDefault("logicalOperators", 0)
            + counts.getOrDefault("functions", 0)
            + counts.getOrDefault("arithmeticOperators", 0)
            + counts.getOrDefault("specialOperators", 0);
        counts.put("total", total);

        return counts;
    }

    /**
     * Helper method to count occurrences of a pattern in a string.
     *
     * @param text The text to search in
     * @param regex The regular expression pattern to search for
     * @return The number of occurrences
     */
    private static int countOccurrences(String text, String regex) {
        final Pattern pattern = Pattern.compile(regex);
        final Matcher matcher = pattern.matcher(text);
        int count = 0;
        while (matcher.find()) {
            count++;
        }
        return count;
    }
```

Exemplo de uso da contagem de operadores de expressão com o AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
```

```
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating DynamoDB expression operator counting
and the 300 operator limit");

        try {
            // Example 1: Analyze a simple expression
            System.out.println("\nExample 1: Analyzing a simple expression");
            String simpleExpression = "Price = :price AND Rating > :rating AND
Category IN (:cat1, :cat2, :cat3)";
            Map<String, Integer> simpleCount =
countOperatorsInExpression(simpleExpression);

            System.out.println("Expression: " + simpleExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
simpleCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
simpleCount.get("logicalOperators"));
            System.out.println("- Functions: " + simpleCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
simpleCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
simpleCount.get("specialOperators"));
            System.out.println("- Total operators: " + simpleCount.get("total"));

            // Example 2: Analyze a complex expression
            System.out.println("\nExample 2: Analyzing a complex expression");
            String complexExpression = "(attribute_exists(Category) AND Size
BETWEEN :min AND :max) OR "
                + "(Price > :price AND contains(Description, :keyword) AND "
                + "(Rating >= :minRating OR Reviews > :minReviews))";
            Map<String, Integer> complexCount =
countOperatorsInExpression(complexExpression);

            System.out.println("Expression: " + complexExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
complexCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
complexCount.get("logicalOperators"));
            System.out.println("- Functions: " + complexCount.get("functions"));
```

```
            System.out.println("- Arithmetic operators: " +
complexCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
complexCount.get("specialOperators"));
            System.out.println("- Total operators: " +
complexCount.get("total"));

            // Example 3: Test approaching the operator limit
            System.out.println("\nExample 3: Testing an expression approaching
the operator limit");
            Map<String, Object> approachingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 290);
            System.out.println(approachingLimit.get("message"));

            // Example 4: Test exceeding the operator limit
            System.out.println("\nExample 4: Testing an expression exceeding the
operator limit");
            Map<String, Object> exceedingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 310);
            System.out.println(exceedingLimit.get("message"));

            // Example 5: Breaking down a complex expression
            System.out.println("\nExample 5: Breaking down a complex expression
into multiple operations");
            Map<String, Object> breakdownResult =
breakDownComplexExpression(dynamoDbClient, tableName, key, 500);
            @SuppressWarnings("unchecked")
            Map<Integer, Map<String, Object>> results =
                (Map<Integer, Map<String, Object>>)
breakdownResult.get("results");
            System.out.println(
                "Processed " + results.size() + " of " +
breakdownResult.get("totalBatches") + " batches");

            // Explain the operator counting rules
            System.out.println("\nKey points about DynamoDB expression operator
counting:");
            System.out.println("1. The maximum number of operators in any
expression is 300");
            System.out.println("2. Each comparison operator (=, <>, <, <=, >, >=)
counts as 1 operator");
            System.out.println("3. Each logical operator (AND, OR, NOT) counts as
1 operator");
```

```
            System.out.println("4. Each function call (attribute_exists,
 contains, etc.) counts as 1 operator");
            System.out.println("5. Each arithmetic operator (+ or -) counts as 1
 operator");
            System.out.println("6. BETWEEN counts as 2 operators (BETWEEN itself
 and the AND within it)");
            System.out.println("7. IN counts as 1 operator regardless of the
 number of values");
            System.out.println("8. Parentheses for grouping and attribute paths
 don't count as operators");
            System.out.println("9. When you exceed the limit, the error always
 reports '301 operators'");
            System.out.println("10. For complex operations, break them into
 multiple smaller operations");

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demonstre a contagem de operadores de expressão usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
```

```
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 conditions
 * @returns {Object} - Object containing the filter expression and attribute
 values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
      case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
      case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
  }

  // Join the conditions with AND or OR
```

```
  const operator = useAnd ? " AND " : " OR ";
  const filterExpression = conditions.join(operator);

  // Calculate the operator count
  // Each condition has 1 operator (=, >, <, contains, attribute_exists)
  // Each AND or OR between conditions is 1 operator
  const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount
 - 1 : 0);

  return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute
 values
 */
function createComplexUpdateExpression(operationsCount) {
  // Initialize the expression parts and attribute values
  const setOperations = [];
  const expressionAttributeValues = {};

  // Generate the specified number of SET operations
  for (let i = 0; i < operationsCount; i++) {
    // Alternate between different types of SET operations
    let operation;
    const valueKey = `:val${i}`;

    switch (i % 3) {
      case 0:
        // Simple assignment (1 operator: =)
        operation = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
```

```
        // Addition (2 operators: = and +)
        operation = `attribute${i} = attribute${i} + ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        // Conditional assignment with if_not_exists (2 operators: = and
 if_not_exists)
        operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    }

    setOperations.push(operation);
  }

  // Create the update expression
  const updateExpression = `SET ${setOperations.join(", ")}`;

  // Calculate the operator count
  // Each operation has 1-2 operators as noted above
  let operatorCount = 0;
  for (let i = 0; i < operationsCount; i++) {
    operatorCount += (i % 3 === 0) ? 1 : 2;
  }

  return {
    updateExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
```

```
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
  const { updateExpression, expressionAttributeValues, operatorCount:
 actualCount } =
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to
 get close to target count

  console.log(`Generated update expression with approximately ${actualCount}
 operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
```

```
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}

/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,
  key,
  totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
  // Using 150 as a conservative limit (well below 300)
  const operationsPerBatch = 100;
  const batchCount = Math.ceil(totalOperations / operationsPerBatch);

  console.log(`Breaking down ${totalOperations} operations into ${batchCount}
 batches`);

  const results = [];
```

```
 // Process each batch
 for (let batch = 0; batch < batchCount; batch++) {
   // Calculate the operations for this batch
   const batchStart = batch * operationsPerBatch;
   const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
   const batchSize = batchEnd - batchStart;

   console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
operations`);

   // Create an update expression for this batch
   const { updateExpression, expressionAttributeValues, operatorCount } =
     createComplexUpdateExpression(batchSize);

   // Define the update parameters
   const params = {
     TableName: tableName,
     Key: key,
     UpdateExpression: updateExpression,
     ExpressionAttributeValues: expressionAttributeValues,
     ReturnValues: "UPDATED_NEW"
   };

   try {
     // Perform the update operation for this batch
     const response = await docClient.send(new UpdateCommand(params));

     results.push({
       batch: batch + 1,
       success: true,
       operatorCount,
       attributes: response.Attributes
     });
   } catch (error) {
     results.push({
       batch: batch + 1,
       success: false,
       operatorCount,
       error: error.message
     });

     // Stop processing if an error occurs
     break;
   }
```

```
  }

  return {
    totalBatches: batchCount,
    results
  };
}

/**
 * Count operators in a DynamoDB expression based on the rules in the
 documentation.
 *
 * This function demonstrates how operators are counted according to the
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
  // Initialize counters for different operator types
  const counts = {
    comparisonOperators: 0,
    logicalOperators: 0,
    functions: 0,
    arithmeticOperators: 0,
    specialOperators: 0,
    total: 0
  };

  // Count comparison operators (=, <>, <, <=, >, >=)
  const comparisonRegex = /[^<>]=[^=]|<>|<=|>=|[^<]>[^=]|[^>]<[^=]/g;
  const comparisonMatches = expression.match(comparisonRegex) || [];
  counts.comparisonOperators = comparisonMatches.length;

  // Count logical operators (AND, OR, NOT)
  const andMatches = expression.match(/\bAND\b/g) || [];
  const orMatches = expression.match(/\bOR\b/g) || [];
  const notMatches = expression.match(/\bNOT\b/g) || [];
  counts.logicalOperators = andMatches.length + orMatches.length +
 notMatches.length;

  // Count functions (attribute_exists, attribute_not_exists, attribute_type,
 begins_with, contains, size)
```

```
  const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
  const functionMatches = expression.match(functionRegex) || [];
  counts.functions = functionMatches.length;

  // Count arithmetic operators (+ and -)
  const arithmeticMatches = expression.match(/[a-zA-Z0-9_)\]]\s*[\+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
  counts.arithmeticOperators = arithmeticMatches.length;

  // Count special operators (BETWEEN, IN)
  const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
  const inMatches = expression.match(/\bIN\b/g) || [];
  counts.specialOperators = betweenMatches.length + inMatches.length;

  // Add extra operators for BETWEEN (each BETWEEN includes an AND)
  counts.logicalOperators += betweenMatches.length;

  // Calculate total
  counts.total = counts.comparisonOperators +
                 counts.logicalOperators +
                 counts.functions +
                 counts.arithmeticOperators +
                 counts.specialOperators;

  return counts;
}
```

Exemplo de uso da contagem de operadores de expressão com o AWS SDK para JavaScript.

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating DynamoDB expression operator counting and the 300
 operator limit");
```

```
 try {
   // Example 1: Analyze a simple expression
   console.log("\nExample 1: Analyzing a simple expression");
   const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
(:cat1, :cat2, :cat3)";
   const simpleCount = countOperatorsInExpression(simpleExpression);

   console.log(`Expression: ${simpleExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
   console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
   console.log(`- Functions: ${simpleCount.functions}`);
   console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
   console.log(`- Special operators: ${simpleCount.specialOperators}`);
   console.log(`- Total operators: ${simpleCount.total}`);

   // Example 2: Analyze a complex expression
   console.log("\nExample 2: Analyzing a complex expression");
   const complexExpression =
     "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
     "(Price > :price AND contains(Description, :keyword) AND " +
     "(Rating >= :minRating OR Reviews > :minReviews))";
   const complexCount = countOperatorsInExpression(complexExpression);

   console.log(`Expression: ${complexExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
   console.log(`- Logical operators: ${complexCount.logicalOperators}`);
   console.log(`- Functions: ${complexCount.functions}`);
   console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
   console.log(`- Special operators: ${complexCount.specialOperators}`);
   console.log(`- Total operators: ${complexCount.total}`);

   // Example 3: Test approaching the operator limit
   console.log("\nExample 3: Testing an expression approaching the operator
limit");
   const approachingLimit = await testOperatorLimit(config, tableName, key,
290);
   console.log(approachingLimit.message);

   // Example 4: Test exceeding the operator limit
   console.log("\nExample 4: Testing an expression exceeding the operator
limit");
   const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
```

```
    console.log(exceedingLimit.message);

    // Example 5: Breaking down a complex expression
    console.log("\nExample 5: Breaking down a complex expression into multiple
 operations");
    const breakdownResult = await breakDownComplexExpression(config, tableName,
 key, 500);
    console.log(`Processed ${breakdownResult.results.length} of
 ${breakdownResult.totalBatches} batches`);

    // Explain the operator counting rules
    console.log("\nKey points about DynamoDB expression operator counting:");
    console.log("1. The maximum number of operators in any expression is 300");
    console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
 operator");
    console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
    console.log("4. Each function call (attribute_exists, contains, etc.) counts
 as 1 operator");
    console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
    console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND
 within it)");
    console.log("7. IN counts as 1 operator regardless of the number of values");
    console.log("8. Parentheses for grouping and attribute paths don't count as
 operators");
    console.log("9. When you exceed the limit, the error always reports '301
 operators'");
    console.log("10. For complex operations, break them into multiple smaller
 operations");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demonstre a contagem de operadores de expressão usando o AWS SDK para Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List, Optional, Tuple



def create_complex_filter_expression(
    attribute_name: str, values: List[Any], use_or: bool = True
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a complex filter expression with multiple conditions.

    This function demonstrates how to build a complex filter expression
    and count the number of operators used.

    Args:
        attribute_name (str): The name of the attribute to filter on.
        values (List[Any]): List of values to compare against.
        use_or (bool, optional): Whether to use OR between conditions. Defaults
 to True.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if not values:
        return "", {}, {}, 0

    # Initialize expression components
    filter_expression = ""
    expression_attribute_values = {}
    expression_attribute_names = {"#attr": attribute_name}
    operator_count = 0
```

```
    # Build the filter expression
    for i, value in enumerate(values):
        value_placeholder = f":val{i}"
        expression_attribute_values[value_placeholder] = value

        if i > 0:
            # Add OR or AND operator between conditions
            filter_expression += " OR " if use_or else " AND "
            operator_count += 1  # Count the OR/AND operator

        # Add the condition
        filter_expression += f"#attr = {value_placeholder}"
        operator_count += 1  # Count the = operator

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def create_nested_filter_expression(
    depth: int, conditions_per_level: int
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a deeply nested filter expression with multiple conditions.

    This function demonstrates how to build a complex nested filter expression
    and count the number of operators used.

    Args:
        depth (int): The depth of nesting.
        conditions_per_level (int): Number of conditions at each level.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if depth <= 0 or conditions_per_level <= 0:
        return "", {}, {}, 0
```

```
    # Initialize expression components
    expression_attribute_values = {}
    expression_attribute_names = {}
    operator_count = 0

    def build_nested_expression(current_depth: int, prefix: str) -> str:
        nonlocal operator_count

        if current_depth <= 0:
            return ""

        # Build conditions at this level
        conditions = []
        for i in range(conditions_per_level):
            attr_name = f"attr{prefix}_{i}"
            attr_placeholder = f"#attr{prefix}_{i}"
            val_placeholder = f":val{prefix}_{i}"

            expression_attribute_names[attr_placeholder] = attr_name
            expression_attribute_values[val_placeholder] = i

            conditions.append(f"{attr_placeholder} = {val_placeholder}")
            operator_count += 1  # Count the = operator

        # Join conditions with AND
        level_expression = " AND ".join(conditions)
        operator_count += max(0, len(conditions) - 1)  # Count the AND operators

        # If not at the deepest level, add nested expressions
        if current_depth > 1:
            nested_expr = build_nested_expression(current_depth - 1,
f"{prefix}_{current_depth}")
            if nested_expr:
                level_expression = f"({level_expression}) OR ({nested_expr})"
                operator_count += 1  # Count the OR operator

        return level_expression

    # Build the expression starting from the top level
    filter_expression = build_nested_expression(depth, "1")

    return (
        filter_expression,
```

```
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def count_operators_in_update_expression(update_expression: str) -> int:
    """
    Count the number of operators in an update expression.

    This function demonstrates how to count operators in an update expression
    based on DynamoDB's rules.

    Args:
        update_expression (str): The update expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count SET operations
    if "SET" in update_expression:
        set_section = (
            update_expression.split("SET")[1].split("REMOVE")[0].split("ADD")
[0].split("DELETE")[0]
        )

        # Count assignment operators (=)
        operator_count += set_section.count("=")

        # Count arithmetic operators (+, -)
        operator_count += set_section.count("+")
        operator_count += set_section.count("-")

        # Count list_append function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("list_append")

        # Count if_not_exists function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("if_not_exists")

    # Count REMOVE operations (no additional operators)

    # Count ADD operations (each ADD counts as 1 operator)
```

```
    if "ADD" in update_expression:
        add_section = (
            update_expression.split("ADD")[1].split("DELETE")[0].split("SET")
[0].split("REMOVE")[0]
        )
        operator_count += add_section.count(",") + 1

    # Count DELETE operations (each DELETE counts as 1 operator)
    if "DELETE" in update_expression:
        delete_section = (
            update_expression.split("DELETE")[1].split("SET")[0].split("ADD")
[0].split("REMOVE")[0]
        )
        operator_count += delete_section.count(",") + 1

    return operator_count


def count_operators_in_condition_expression(condition_expression: str) -> int:
    """
    Count the number of operators in a condition expression.

    This function demonstrates how to count operators in a condition expression
    based on DynamoDB's rules.

    Args:
        condition_expression (str): The condition expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count comparison operators
    comparison_operators = ["=", "<>", "<", "<=", ">", ">="]
    for op in comparison_operators:
        operator_count += condition_expression.count(op)

    # Count logical operators
    operator_count += condition_expression.upper().count(" AND ")
    operator_count += condition_expression.upper().count(" OR ")
    operator_count += condition_expression.upper().count("NOT ")

    # Count BETWEEN operator (counts as 2: BETWEEN + AND)
```

```
    between_count = condition_expression.upper().count(" BETWEEN ")
    operator_count += between_count * 2

    # Count IN operator (counts as 1 regardless of number of values)
    operator_count += condition_expression.upper().count(" IN ")

    # Count functions (each counts as 1 operator)
    functions = [
        "attribute_exists",
        "attribute_not_exists",
        "attribute_type",
        "begins_with",
        "contains",
        "size",
    ]
    for func in functions:
        operator_count += condition_expression.lower().count(func)

    return operator_count


# Note: This function is for demonstration purposes only and should be called
 from example_usage()
# It's not meant to be used directly as a test function
def _test_expression_limit(
    table_name: str, key: Dict[str, Any], operator_count: int, attribute_name:
 str = "TestAttribute"
) -> Tuple[bool, Optional[str]]:
    """
    Test if an expression with a specific number of operators exceeds the limit.

    This function demonstrates how to test the 300 operator limit by creating
    an expression with a specified number of operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        operator_count (int): The number of operators to include in the
 expression.
        attribute_name (str, optional): The name of the attribute to update.
 Defaults to "TestAttribute".

    Returns:
        Tuple[bool, Optional[str]]: A tuple containing:
```

```
        - A boolean indicating if the operation succeeded
        - The error message if it failed, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create an update expression with the specified number of operators
    update_expression = f"SET #{attribute_name} = :val0"
    expression_attribute_names = {f"#{attribute_name}": attribute_name}
    expression_attribute_values = {":val0": 0}

    # Add additional SET operations to reach the desired operator count
    # Each assignment adds 1 operator
    for i in range(1, operator_count):
        attr_name = f"{attribute_name}{i}"
        attr_placeholder = f"#attr{i}"
        val_placeholder = f":val{i}"

        update_expression += f", {attr_placeholder} = {val_placeholder}"
        expression_attribute_names[attr_placeholder] = attr_name
        expression_attribute_values[val_placeholder] = i

    try:
        # Attempt the update operation
        table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues=expression_attribute_values,
        )
        return True, None
    except ClientError as e:
        error_message = e.response["Error"]["Message"]

        if "expression contains too many operators" in error_message.lower():
            return False, error_message
        else:
            # Other error occurred
            raise
```

Exemplo de uso da contagem de operadores de expressão com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use the expression operator counting functions."""

    print("Example 1: Creating a complex filter expression with multiple
 conditions")
    attribute_name = "Status"
    values = ["Active", "Pending", "Processing", "Shipped", "Delivered"]

    filter_expr, expr_attr_vals, expr_attr_names, op_count =
 create_complex_filter_expression(
        attribute_name=attribute_name, values=values, use_or=True
    )

    print(f"Filter Expression: {filter_expr}")
    print(f"Expression Attribute Values: {expr_attr_vals}")
    print(f"Expression Attribute Names: {expr_attr_names}")
    print(f"Operator Count: {op_count}")

    print("\nExample 2: Creating a nested filter expression")
    nested_expr, nested_vals, nested_names, nested_count =
 create_nested_filter_expression(
        depth=3, conditions_per_level=2
    )

    print(f"Nested Filter Expression: {nested_expr}")
    print(f"Operator Count: {nested_count}")

    print("\nExample 3: Counting operators in an update expression")
    update_expression = "SET #name = :name, #age = :age + :increment,
 #address.#city = :city, #status = if_not_exists(#status, :default_status) REMOVE
 #old_field ADD #counter :value DELETE #set_attr :set_val"
    update_op_count = count_operators_in_update_expression(update_expression)

    print(f"Update Expression: {update_expression}")
    print(f"Operator Count: {update_op_count}")

    print("\nExample 4: Counting operators in a condition expression")
    condition_expression = "(#status = :active OR #status = :pending) AND #price
 BETWEEN :min_price AND :max_price AND attribute_exists(#category) AND NOT
 (#stock <= :min_stock)"
```

```
    condition_op_count =
count_operators_in_condition_expression(condition_expression)

    print(f"Condition Expression: {condition_expression}")
    print(f"Operator Count: {condition_op_count}")

    print("\nExample 5: Testing the 300 operator limit")

    # This is just for demonstration - in a real application, you would use your
actual table
    # Note: This function is renamed to _test_expression_limit to avoid pytest
trying to run it
    print("In a real application, you would test with _test_expression_limit
function")
    print("Expression with 250 operators would be under the limit")
    print("Expression with 350 operators would exceed the 300 operator limit")

    print("\nOperator Counting Rules in DynamoDB:")
    print("1. Comparison Operators (=, <>, <, <=, >, >=): 1 operator each")
    print("2. Logical Operators (AND, OR, NOT): 1 operator each")
    print("3. BETWEEN: 2 operators (BETWEEN + AND)")
    print("4. IN: 1 operator (regardless of number of values)")
    print("5. Functions (attribute_exists, begins_with, etc.): 1 operator each")
    print("6. Arithmetic Operators (+, -): 1 operator each")
    print("7. SET assignments (=): 1 operator each")
    print("8. ADD and DELETE operations: 1 operator each")

    print("\nStrategies for Working Within the 300 Operator Limit:")
    print("1. Break operations into multiple requests")
    print("2. Use DynamoDB Transactions for complex operations")
    print("3. Optimize data model to reduce query complexity")
    print("4. Use application-side filtering for less critical filters")
    print("5. Consider using IN operator instead of multiple OR conditions")
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma API REST do API Gateway para monitorar dados da COVID-19

O exemplo de código a seguir mostra como criar uma API REST que simula um sistema para monitorar casos diários de COVID-19 nos Estados Unidos, usando dados fictícios.

Python

SDK para Python (Boto3)

Mostra como usar o AWS Chalice com o AWS SDK para Python (Boto3) para criar uma API REST sem servidor que usa o Amazon API Gateway, o AWS Lambda e o Amazon DynamoDB. A API REST simula um sistema que monitora casos diários de COVID-19 nos Estados Unidos, usando dados fictícios. Aprenda como:

- Usar o AWS Chalice para definir rotas nas funções do Lambda que são chamadas para lidar com solicitações REST provenientes do API Gateway.
- Usar as funções do Lambda para recuperar e armazenar dados em uma tabela do DynamoDB para atender a solicitações REST.
- Definir recursos de função de segurança e estrutura de tabela em um modelo do AWS CloudFormation.
- Usar o AWS Chalice e o CloudFormation para empacotar e implantar todos os recursos necessários.
- Usar o CloudFormation para limpar todos os recursos criados.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação de mensageiro com o Step Functions

O exemplo de código a seguir mostra como criar uma aplicação de mensageiro do AWS Step Functions que recupera registros de mensagens de uma tabela de banco de dados.

Python

SDK para Python (Boto3)

Mostra como usar o AWS SDK para Python (Boto3) com o AWS Step Functions para criar uma aplicação de mensageiro que recupere registros de mensagens de uma tabela do Amazon DynamoDB e os envia com o Amazon Simple Queue Service (Amazon SQS). A máquina de estado se integra a uma função do AWS Lambda para verificar o banco de dados em busca de mensagens não enviadas.

- Crie uma máquina de estado que recupere e atualize registros de mensagens de uma tabela do Amazon DynamoDB.
- Atualize a definição de máquina de estado para enviar mensagens ao Amazon Simple Queue Service (Amazon SQS).
- Inicie e interrompa execuções da máquina de estado.
- Conecte-se ao Lambda, ao DynamoDB e ao Amazon SQS por meio de uma máquina de estado usando integrações de serviço.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

.NET

SDK para .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK para C++

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK para Java 2.x

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK para Kotlin

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK para PHP

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK para Rust

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Crie uma tabela do DynamoDB com um índice secundário global usando o SDK da AWS.

O código de exemplo a seguir mostra como criar uma tabela com um índice secundário global.

Java

SDK para Java 2.x

Crie uma tabela do DynamoDB com um índice secundário global usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
```

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProjectionType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public void createTable() {
        try {
            // Attribute definitions
            final List<AttributeDefinition> attributeDefinitions = new
 ArrayList<>();
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(ISSUE_ID_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(TITLE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(CREATE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
```

```
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(DUE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());

            // Key schema for table
            final List<KeySchemaElement> tableKeySchema = new ArrayList<>();
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.HASH)
                .build()); // Partition key
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.RANGE)
                .build()); // Sort key

            // Initial provisioned throughput settings for the indexes
            final ProvisionedThroughput ptIndex = ProvisionedThroughput.builder()
                .readCapacityUnits(1L)
                .writeCapacityUnits(1L)
                .build();

            // CreateDateIndex
            final List<KeySchemaElement> createDateKeySchema = new ArrayList<>();
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(CREATE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection createDateProjection = Projection.builder()
                .projectionType(ProjectionType.INCLUDE)
                .nonKeyAttributes(DESCRIPTION_ATTR, STATUS_ATTR)
                .build();

            final GlobalSecondaryIndex createDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(CREATE_DATE_INDEX)
                .keySchema(createDateKeySchema)
                .projection(createDateProjection)
                .provisionedThroughput(ptIndex)
```

```
                .build();

            // TitleIndex
            final List<KeySchemaElement> titleKeySchema = new ArrayList<>();
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection titleProjection =

Projection.builder().projectionType(ProjectionType.KEYS_ONLY).build();

            final GlobalSecondaryIndex titleIndex =
GlobalSecondaryIndex.builder()
                .indexName(TITLE_INDEX)
                .keySchema(titleKeySchema)
                .projection(titleProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // DueDateIndex
            final List<KeySchemaElement> dueDateKeySchema = new ArrayList<>();
            dueDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(DUE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());

            final Projection dueDateProjection =
                Projection.builder().projectionType(ProjectionType.ALL).build();

            final GlobalSecondaryIndex dueDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(DUE_DATE_INDEX)
                .keySchema(dueDateKeySchema)
                .projection(dueDateProjection)
                .provisionedThroughput(ptIndex)
                .build();
```

```
            final CreateTableRequest createTableRequest =
CreateTableRequest.builder()
                .tableName(TABLE_NAME)
                .keySchema(tableKeySchema)
                .attributeDefinitions(attributeDefinitions)
                .globalSecondaryIndexes(createDateIndex, titleIndex,
dueDateIndex)
                .provisionedThroughput(ProvisionedThroughput.builder()
                    .readCapacityUnits(1L)
                    .writeCapacityUnits(1L)
                    .build())
                .build();

            System.out.println("Creating table " + TABLE_NAME + "...");
            dynamoDbClient.createTable(createTableRequest);

            // Wait for table to become active
            System.out.println("Waiting for " + TABLE_NAME + " to become
ACTIVE...");
            final DynamoDbWaiter waiter = dynamoDbClient.waiter();
            final DescribeTableRequest describeTableRequest =
                DescribeTableRequest.builder().tableName(TABLE_NAME).build();

            final WaiterResponse<DescribeTableResponse> waiterResponse =
                waiter.waitUntilTableExists(describeTableRequest);
            waiterResponse.matched().response().ifPresent(response ->
System.out.println("Table is now ready for use"));

        } catch (DynamoDbException e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK for Java 2.x.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma tabela do DynamoDB com uma configuração de throughput a quente usando um AWS SDK

Os exemplos de código a seguir mostram como criar uma tabela com o throughput a quente habilitado.

Java

SDK para Java 2.x

Crie uma tabela do DynamoDB com uma configuração de throughput a quente usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
    }

    /**
     * Builds a ProvisionedThroughput object with the specified read and write
 capacity units.
     *
     * @param readCapacityUnits The read capacity units
     * @param writeCapacityUnits The write capacity units
     * @return A configured ProvisionedThroughput object
     */
    public static ProvisionedThroughput buildProvisionedThroughput(
```

```
        final Long readCapacityUnits, final Long writeCapacityUnits) {
        return ProvisionedThroughput.builder()
            .readCapacityUnits(readCapacityUnits)
            .writeCapacityUnits(writeCapacityUnits)
            .build();
    }

    /**
     * Builds an AttributeDefinition with the specified name and type.
     *
     * @param attributeName The attribute name
     * @param scalarAttributeType The attribute type
     * @return A configured AttributeDefinition
     */
    private static AttributeDefinition buildAttributeDefinition(
        final String attributeName, final ScalarAttributeType
scalarAttributeType) {
        return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
    }

    /**
     * Builds a KeySchemaElement with the specified name and key type.
     *
     * @param attributeName The attribute name
     * @param keyType The key type (HASH or RANGE)
     * @return A configured KeySchemaElement
     */
    private static KeySchemaElement buildKeySchemaElement(final String
attributeName, final KeyType keyType) {
        return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
    }

    /**
     * Creates a DynamoDB table with the specified configuration including warm
throughput settings.
     *
     * @param ddb The DynamoDB client
     * @param tableName The name of the table to create
```

```
    * @param partitionKey The partition key attribute name
    * @param sortKey The sort key attribute name
    * @param miscellaneousKeyAttribute Additional key attribute name for GSI
    * @param nonKeyAttribute Non-key attribute to include in GSI projection
    * @param tableReadCapacityUnits Read capacity units for the table
    * @param tableWriteCapacityUnits Write capacity units for the table
    * @param tableWarmReadUnitsPerSecond Warm read units per second for the
table
    * @param tableWarmWriteUnitsPerSecond Warm write units per second for the
table
    * @param globalSecondaryIndexName The name of the GSI to create
    * @param globalSecondaryIndexReadCapacityUnits Read capacity units for the
GSI
    * @param globalSecondaryIndexWriteCapacityUnits Write capacity units for the
GSI
    * @param globalSecondaryIndexWarmReadUnitsPerSecond Warm read units per
second for the GSI
    * @param globalSecondaryIndexWarmWriteUnitsPerSecond Warm write units per
second for the GSI
    */
   public static void createDynamoDBTable(
       final DynamoDbClient ddb,
       final String tableName,
       final String partitionKey,
       final String sortKey,
       final String miscellaneousKeyAttribute,
       final String nonKeyAttribute,
       final Long tableReadCapacityUnits,
       final Long tableWriteCapacityUnits,
       final Long tableWarmReadUnitsPerSecond,
       final Long tableWarmWriteUnitsPerSecond,
       final String globalSecondaryIndexName,
       final Long globalSecondaryIndexReadCapacityUnits,
       final Long globalSecondaryIndexWriteCapacityUnits,
       final Long globalSecondaryIndexWarmReadUnitsPerSecond,
       final Long globalSecondaryIndexWarmWriteUnitsPerSecond) {

       // Define the table attributes
       final AttributeDefinition partitionKeyAttribute =
buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
       final AttributeDefinition sortKeyAttribute =
buildAttributeDefinition(sortKey, ScalarAttributeType.S);
       final AttributeDefinition miscellaneousKeyAttributeDefinition =
```

```
            buildAttributeDefinition(miscellaneousKeyAttribute,
ScalarAttributeType.N);
        final AttributeDefinition[] attributeDefinitions = {
            partitionKeyAttribute, sortKeyAttribute,
miscellaneousKeyAttributeDefinition
        };

        // Define the table key schema
        final KeySchemaElement partitionKeyElement =
buildKeySchemaElement(partitionKey, KeyType.HASH);
        final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
KeyType.RANGE);
        final KeySchemaElement[] keySchema = {partitionKeyElement,
sortKeyElement};

        // Define the provisioned throughput for the table
        final ProvisionedThroughput provisionedThroughput =
            buildProvisionedThroughput(tableReadCapacityUnits,
tableWriteCapacityUnits);

        // Define the Global Secondary Index (GSI)
        final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
        final KeySchemaElement globalSecondaryIndexSortKeyElement =
            buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
        final KeySchemaElement[] gsiKeySchema = {
            globalSecondaryIndexPartitionKeyElement,
globalSecondaryIndexSortKeyElement
        };

        final Projection gsiProjection = Projection.builder()
            .projectionType(PROJECTION_TYPE_INCLUDE)
            .nonKeyAttributes(nonKeyAttribute)
            .build();

        final ProvisionedThroughput gsiProvisionedThroughput =
            buildProvisionedThroughput(globalSecondaryIndexReadCapacityUnits,
globalSecondaryIndexWriteCapacityUnits);

        // Define the warm throughput for the Global Secondary Index (GSI)
        final WarmThroughput gsiWarmThroughput = buildWarmThroughput(
            globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);
```

```
        final GlobalSecondaryIndex globalSecondaryIndex =
 GlobalSecondaryIndex.builder()
            .indexName(globalSecondaryIndexName)
            .keySchema(gsiKeySchema)
            .projection(gsiProjection)
            .provisionedThroughput(gsiProvisionedThroughput)
            .warmThroughput(gsiWarmThroughput)
            .build();

        // Define the warm throughput for the table
        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableWarmReadUnitsPerSecond,
 tableWarmWriteUnitsPerSecond);

        final CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .attributeDefinitions(attributeDefinitions)
            .keySchema(keySchema)
            .provisionedThroughput(provisionedThroughput)
            .globalSecondaryIndexes(globalSecondaryIndex)
            .warmThroughput(tableWarmThroughput)
            .build();

        final CreateTableResponse response = ddb.createTable(request);
        System.out.println(response);
    }
```

- Para obter detalhes da API, consulte CreateTable na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Crie uma tabela do DynamoDB com uma configuração de throughput a quente usando o AWS SDK para JavaScript.

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

export async function createDynamoDBTableWithWarmThroughput(
  tableName,
```

```
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
```

```
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
    return response;
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/
```

- Para ver detalhes da API, consulte CreateTable em AWS SDK para JavaScript SDK for Python (Boto3) API Reference.

Python

SDK para Python (Boto3)

Crie uma tabela do DynamoDB com uma configuração de throughput a quente usando o AWS SDK para Python (Boto3).

```
from boto3 import client
from botocore.exceptions import ClientError


def create_dynamodb_table_warm_throughput(
    table_name,
    partition_key,
    sort_key,
    misc_key_attr,
    non_key_attr,
    table_provisioned_read_units,
    table_provisioned_write_units,
    table_warm_reads,
    table_warm_writes,
    gsi_name,
    gsi_provisioned_read_units,
    gsi_provisioned_write_units,
    gsi_warm_reads,
    gsi_warm_writes,
    region_name="us-east-1",
):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being
 created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned
 read capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's
 warm throughput.
```

```
   :param table_warm_writes: The write units per second setting for the table's
warm throughput.
   :param gsi_name: The name of the Global Secondary Index (GSI) to be created
on the table.
   :param gsi_provisioned_read_units: The configured Global Secondary Index
(GSI) provisioned read capacity units.
   :param gsi_provisioned_write_units: The configured Global Secondary Index
(GSI) provisioned write capacity units.
   :param gsi_warm_reads: The read units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param gsi_warm_writes: The write units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param region_name: The AWS Region name to target. defaults to us-east-1
   """
   try:
       ddb = client("dynamodb", region_name=region_name)

       # Define the table attributes
       attribute_definitions = [
           {"AttributeName": partition_key, "AttributeType": "S"},
           {"AttributeName": sort_key, "AttributeType": "S"},
           {"AttributeName": misc_key_attr, "AttributeType": "N"},
       ]

       # Define the table key schema
       key_schema = [
           {"AttributeName": partition_key, "KeyType": "HASH"},
           {"AttributeName": sort_key, "KeyType": "RANGE"},
       ]

       # Define the provisioned throughput for the table
       provisioned_throughput = {
           "ReadCapacityUnits": table_provisioned_read_units,
           "WriteCapacityUnits": table_provisioned_write_units,
       }

       # Define the global secondary index
       gsi_key_schema = [
           {"AttributeName": sort_key, "KeyType": "HASH"},
           {"AttributeName": misc_key_attr, "KeyType": "RANGE"},
       ]
       gsi_projection = {"ProjectionType": "INCLUDE", "NonKeyAttributes":
[non_key_attr]}
       gsi_provisioned_throughput = {
```

```
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units,
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes,
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput,
            }
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes,
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput,
        )

        print(response)
        return response
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

- Para ver os detalhes da API, consulte CreateTable na Referência da API do AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação Web para monitorar dados do DynamoDB

O exemplo de código a seguir mostra como criar uma aplicação Web que monitora itens de trabalho em uma tabela do Amazon DynamoDB e usa o Amazon Simple Email Service (Amazon SES) para enviar relatórios.

.NET

SDK para .NET

Mostra como usar a API .NET do Amazon DynamoDB para construir uma aplicação Web dinâmica que monitora os dados de trabalho do DynamoDB.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon SES

Java

SDK para Java 2.x

Mostra como usar a API do Amazon DynamoDB para construir uma aplicação Web dinâmica que monitora os dados de trabalho do DynamoDB.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon SES

Kotlin

SDK para Kotlin

Mostra como usar a API do Amazon DynamoDB para construir uma aplicação Web dinâmica que monitora os dados de trabalho do DynamoDB.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon SES

Python

SDK para Python (Boto3).

Mostra como usar o AWS SDK para Python (Boto3) para construir um serviço REST que monitora itens de trabalho no Amazon DynamoDB e envia relatórios por e-mail usando o Amazon Simple Email Service (Amazon SES). Este exemplo usa a estrutura web Flask para lidar com o roteamento HTTP e se integra a uma página da Web do React para apresentar uma aplicação Web totalmente funcional.

- Crie um serviço REST de Flask que se integre com Serviços da AWS.
- Leia, grave e atualize itens de trabalho armazenados em uma tabela do DynamoDB.
- Use o Amazon SES para enviar relatórios por e-mail de itens de trabalho.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no Repositório de exemplos de código da AWS no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon SES

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação de chat websocket com o API Gateway

O exemplo de código a seguir mostra como criar uma aplicação de chat que é atendido por uma API de Websocket criada no Amazon API Gateway.

Python

SDK para Python (Boto3)

Mostra como usar o AWS SDK para Python (Boto3) com o Amazon API Gateway V2 para criar uma API de Websocket que se integra ao AWS Lambda e ao Amazon DynamoDB.

- Crie uma API de Websocket atendida pelo API Gateway.
- Defina um manipulador do Lambda que armazena conexões no DynamoDB e publica mensagens para outros participantes do chat.
- Conecte-se à aplicação de chat websocket e envie mensagens com o pacote Websockets.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar um item do DynamoDB com TTL usando um SDK da AWS

Os exemplos de código a seguir mostram como criar um item com TTL.

Java

SDK para Java 2.x

```
package com.amazon.samplelib.ttl;
```

```
import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";
```

```
    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
```

```
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
```

```
            PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });
```

```
    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Para obter detalhes da API, consulte PutItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e
```

```
# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Para obter detalhes da API, consulte PutItem na Referência da API do AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Crie e gerencie tabelas globais do DynamoDB com consistência forte multirregional usando um SDK da AWS.

O exemplo de código a seguir mostra como criar e gerenciar tabelas globais do DynamoDB com consistência forte multirregional (MRSC).

- Crie uma tabela com consistência forte multirregional.
- Verifique a configuração de MRSC e o status da réplica.
- Teste a consistência forte entre regiões com leituras imediatas.
- Faça gravações condicionais com garantia de MRSC.
- Limpe os recursos da tabela global com MRSC.

Bash

AWS CLI com script Bash

Crie uma tabela com consistência forte multirregional.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
```

```
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Verifique a configuração de MRSC e o status da réplica.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Teste a consistência forte com leituras imediatas entre regiões.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
```

```
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Faça gravações condicionais com garantia de MRSC.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Limpe os recursos da tabela global com MRSC.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK para Java 2.x

Crie uma tabela regional pronta para conversão MRSC usando AWS SDK for Java 2.x.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
```

```
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Converta uma tabela regional para MRSC com réplicas e testemunhas usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
```

```
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
```

```
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Descreva uma configuração de tabela global do MRSC usando AWS SDK for Java 2.x.

```
    public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing MRSC global table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
```

```
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            if (response.table().globalTableWitnesses() != null
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Adicione itens de teste para verificar a forte consistência do MRSC usando AWS SDK for Java 2.x.

```
   public static PutItemResponse putTestItem(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final String artist,
       final String songTitle,
       final String album,
       final String year) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Adding test item to MRSC global table: " + tableName);

           Map<String, AttributeValue> item = new HashMap<>();
           item.put("Artist", AttributeValue.builder().s(artist).build());
           item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           if (album != null && !album.trim().isEmpty()) {
               item.put("Album", AttributeValue.builder().s(album).build());
           }
           if (year != null && !year.trim().isEmpty()) {
               item.put("Year", AttributeValue.builder().n(year).build());
           }

           PutItemRequest putItemRequest =
               PutItemRequest.builder().tableName(tableName).item(item).build();

           PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
           LOGGER.info("Test item added successfully with strong consistency");
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Leia itens com leituras consistentes de réplicas do MRSC usando AWS SDK for Java 2.x.

```
    public static GetItemResponse getItemWithConsistentRead(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            GetItemRequest getItemRequest = GetItemRequest.builder()
```

```
                .tableName(tableName)
                .key(key)
                .consistentRead(true)
                .build();

            GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

            if (response.hasItem()) {
                LOGGER.info("Item found with strong consistency - no wait time
needed");
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Faça atualizações condicionais com garantias de MRSC usando AWS SDK for Java 2.x.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
```

```
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
```

```
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Aguarde até que as réplicas e testemunhas do MRSC se tornem ativas usando AWS SDK for Java 2.x.

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
```

```
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
```

```
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
```

```
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Limpe as réplicas e testemunhas do MRSC usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
```

```
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Demonstração completa do fluxo de trabalho do MRSC usando AWS SDK for Java 2.x.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
```

```
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);
```

```
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
```

```
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.

- CreateTable
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- UpdateItem
- UpdateTable

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Crie e gerencie tabelas globais do DynamoDB demonstrando o MREC usando um SDK da AWS

O exemplo de código a seguir mostra como criar e gerenciar tabelas globais do DynamoDB com réplicas entre várias regiões.

- Crie uma tabela com um índice secundário global e DynamoDB Streams.
- Adicione réplicas em diferentes regiões para criar uma tabela global.
- Remova réplicas de uma tabela global.
- Adicione itens de teste para verificar a replicação entre regiões.
- Descreva a configuração da tabela global e o status da réplica.

Java

SDK para Java 2.x

Crie uma tabela com um índice secundário global e DynamoDB Streams usando AWS SDK for Java 2.x.

```
    public static CreateTableResponse createTableWithGSI(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

        if (dynamoDbClient == null) {
```

```
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }

        try {
            LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(
```

```
Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Aguarde até que uma tabela se torne ativa usando AWS SDK for Java 2.x.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
```

```
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Adicione uma réplica para criar ou estender uma tabela global usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse addReplica(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }

        try {
```

```
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Remova uma réplica de uma tabela global usando AWS SDK for Java 2.x.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {
```

```
        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Adicione itens de teste para verificar a replicação usando AWS SDK for Java 2.x.

```
   public static PutItemResponse putTestItem(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Adding test item to table: " + tableName);

           Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
           item.put(
               "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                   .s(artist)
                   .build());
           item.put(
               "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                   .s(songTitle)
                   .build());

           PutItemRequest putItemRequest =
               PutItemRequest.builder().tableName(tableName).item(item).build();

           PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
           LOGGER.info("Test item added successfully");
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Descreva a configuração e as réplicas da tabela global usando AWS SDK for Java 2.x.

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
```

```
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Exemplo completo de operações de tabela global usando AWS SDK for Java 2.x.

```
    public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
                createTableWithGSI(dynamoDbClient, tableName, indexName);

                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
```

```
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.
  - CreateTable

- DescribeTable
- PutItem
- UpdateTable

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Excluir dados do DynamoDB usando declarações DELETE do PartiQL com um SDK da AWS

O exemplo de código a seguir mostra como excluir dados usando declarações DELETE do PartiQL.

JavaScript

SDK para JavaScript (v3)

Exclua itens de uma tabela do DynamoDB usando declarações DELETE do PartiQL com o AWS SDK para JavaScript.

```
/**
 * This example demonstrates how to delete items from a DynamoDB table using
 PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
```

```
export const deleteItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
```

```
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
```

```
    console.log("Item deleted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};

/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
 AND ${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });
```

```
  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
```

```
};

/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");

  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
```

```
      sortKeyName: "productId",
      sortKeyValue: "prod456",
    },
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Detectar EPI em imagens com o Amazon Rekognition usando um AWS SDK

Os exemplos de código a seguir mostram como criar uma aplicação que usa o Amazon Rekognition para detectar equipamentos de proteção individual (EPI) em imagens.

Java

SDK para Java 2.x

Mostra como criar uma função do AWS Lambda que detecta imagens com equipamento de proteção individual.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon Rekognition
- Amazon S3

- Amazon SES

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Inserir dados do DynamoDB usando declarações INSERT do PartiQL com um SDK da AWS

O exemplo de código a seguir mostra como inserir dados usando declarações INSERT do PartiQL.

JavaScript

SDK para JavaScript (v3)

Insira itens em uma tabela do DynamoDB usando declarações INSERT do PartiQL com o AWS SDK para JavaScript.

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItem = async (tableName: string, item: Record<string, any>) =>
 {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
 any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
    Statements: statements,
  };

  try {
```

```
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
 attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item with condition:", err);
    throw err;
  }
};
```

```
/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
  // Example table with a simple primary key (just partition key)
  const simpleKeyItem = {
    userId: "user123",
    name: "John Doe",
    email: "john@example.com",
  };
  await insertItem("UsersTable", simpleKeyItem);

  // Example table with composite key (partition key + sort key)
  const compositeKeyItem = {
    orderId: "order456",
    productId: "prod789",
    quantity: 2,
    price: 29.99,
  };
  await insertItem("OrdersTable", compositeKeyItem);

  // Example with Global Secondary Index (GSI)
  // The GSI might be on the email attribute
  const gsiItem = {
    userId: "user789",
    email: "jane@example.com",
    name: "Jane Smith",
    userType: "premium",  // This could be part of a GSI
  };
  await insertItem("UsersTable", gsiItem);

  // Example with Local Secondary Index (LSI)
  // LSI uses the same partition key but different sort key
  const lsiItem = {
    orderId: "order567",  // Partition key
    productId: "prod123", // Sort key for the table
    orderDate: "2023-11-15", // Potential sort key for an LSI
    quantity: 1,
    price: 19.99,
  };
  await insertItem("OrdersTable", lsiItem);

  // Batch insert example with multiple items
```

```
  const batchItems = [
    {
      userId: "user234",
      name: "Alice Johnson",
      email: "alice@example.com",
    },
    {
      userId: "user345",
      name: "Bob Williams",
      email: "bob@example.com",
    },
  ];
  await batchInsertItems("UsersTable", batchItems);
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Chamar uma função do Lambda em um navegador

O exemplo de código a seguir mostra como chamar uma função do AWS Lambda em um navegador.

JavaScript

SDK para JavaScript (v2)

É possível criar uma aplicação baseada em navegador que usa uma função do AWS Lambda para atualizar uma tabela do Amazon DynamoDB com seleções de usuário.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB

- Lambda

SDK para JavaScript (v3)

É possível criar uma aplicação baseada em navegador que usa uma função do AWS Lambda para atualizar uma tabela do Amazon DynamoDB com seleções de usuário. Essa aplicação usa o AWS SDK para JavaScript v3.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Gerenciar índices secundários globais no DynamoDB usando a AWS Command Line Interface v2

O exemplo de código a seguir mostra como gerenciar o ciclo de vida completo dos índices secundários globais.

- Crie uma tabela com um índice secundário global.
- Adicione um novo GSI a uma tabela existente.
- Atualize (aumente) o throughput a quente do GSI.
- Consulte dados usando GSIs.
- Exclua um GSI.

Bash

AWS CLI com script Bash

Crie uma tabela com um índice secundário global.

```
# Create a table with a GSI
aws dynamodb create-table \
```

```
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "IndexName=AlbumIndex,\
        KeySchema=[{AttributeName=AlbumTitle,KeyType=HASH}],\
        Projection={ProjectionType=ALL}"
```

Adicione um novo GSI (sob demanda) a uma tabela existente.

```
# Add a new GSI to an existing table
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Genre,AttributeType=S \
    --global-secondary-index-updates \
        "[{\"Create\":{\"IndexName\":\"GenreIndex\",\
        \"KeySchema\":[{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"}],\
        \"Projection\":{\"ProjectionType\":\"ALL\"}}}]"
```

Atualize (aumente) o throughput a quente do GSI.

```
# Increase the warm throughput of a GSI (default values are 12k reads, 4k writes)
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Update\":{\"IndexName\":\"AlbumIndex\",\
        \"WarmThroughput\":{\"ReadUnitsPerSecond\":15000,\"WriteUnitsPerSecond
\":6000}}}]"
```

Consulte dados usando GSIs.

```
# Query the AlbumIndex GSI
```

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Let It Be"}}'

# Query the GenreIndex GSI
aws dynamodb query \
    --table-name MusicCollection \
    --index-name GenreIndex \
    --key-condition-expression "Genre = :genre" \
    --expression-attribute-values '{":genre":{"S":"Jazz"}}'
```

Exclua um GSI.

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable
  - DeleteTable
  - Consulta
  - UpdateTable

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Gerenciar políticas baseadas em recursos do DynamoDB usando a AWS Command Line Interface v2

O exemplo de código a seguir mostra como gerenciar o ciclo de vida completo das políticas baseadas em recursos para tabelas do DynamoDB.

- Crie uma tabela com uma política de recursos.
- Obtenha uma política de recursos.
- Atualize uma política de recursos.
- Exclua uma política de recursos.

Bash

AWS CLI com script Bash

Crie uma tabela com uma política de recursos.

```
# Step 1: Create a DynamoDB table
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST

# Step 2: Create a resource-based policy document
cat > policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/DynamoDBReadOnly"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
```

```
EOF

# Step 3: Attach the resource-based policy to the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://policy.json
```

Obtenha uma política de recursos.

```
# Get the resource-based policy attached to a table
aws dynamodb get-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

Atualize uma política de recursos.

```
# Step 1: Create an updated policy document
cat > updated-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:role/DynamoDBReadOnly",
          "arn:aws:iam::123456789012:role/DynamoDBAnalytics"
        ]
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
EOF
```

```
# Step 2: Update the resource-based policy on the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://updated-policy.json
```

Exclua uma política de recursos.

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Monitorar a performance do Amazon DynamoDB usando um SDK da AWS

O exemplo de código, apresentado a seguir, mostra como configurar o uso do DynamoDB por uma aplicação para monitorar o desempenho.

Java

SDK para Java 2.x

Este exemplo mostra como configurar uma aplicação em Java para monitorar o desempenho do DynamoDB. A aplicação envia dados de métricas para o CloudWatch, que é um local em que você pode monitorar o desempenho.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços utilizados neste exemplo

- CloudWatch
- DynamoDB

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Executar operações de consulta avançada no DynamoDB usando um SDK da AWS

Os exemplos de código a seguir mostram como realizar operações de consulta avançada no DynamoDB.

- Consulte tabelas usando várias técnicas de filtragem e condição.
- Implemente a paginação para grandes conjuntos de resultados.
- Use índices secundários globais para padrões de acesso alternativos.
- Aplique controles de consistência com base nos requisitos da aplicação.

Java

SDK para Java 2.x

Consulta com leituras altamente consistentes usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
   public QueryResponse queryWithConsistentReads(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final boolean useConsistentRead) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
       expressionAttributeValues.put(
           EXPRESSION_ATTRIBUTE_VALUE_PK,
           AttributeValue.builder().s(partitionKeyValue).build());

       // Create the query request
       final QueryRequest queryRequest = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression(KEY_CONDITION_EXPRESSION)
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .consistentRead(useConsistentRead)
           .build();

       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
           LOGGER.log(Level.INFO, "Query successful. Found {0} items",
response.count());
           return response;
       } catch (ResourceNotFoundException e) {
           LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
           throw e;
       } catch (DynamoDbException e) {
           LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
           throw e;
       }
   }
```

Consulta com um índice secundário global usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
```

```
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
```

```
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

Consulta com paginação usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }
```

```
                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
        }
    }
```

Consulta com filtros complexos usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
 priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
 categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
 Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
 Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
```

```
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

Consulta com uma expressão de filtro criada dinamicamente usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
 QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
 partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
 e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
 e.getMessage());
            throw e;
        } finally {
```

```
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
```

```
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

Consulta com uma expressão de filtro e limite usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
 filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
 filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
 filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulta com leituras altamente consistentes usando o AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
```

```
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

Consulta com um índice secundário global usando o AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
```

```
    throw error;
  }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

Consulta com paginação usando o AWS SDK para JavaScript.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
```

```
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}
```

```
/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

Consulta com filtros complexos usando o AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
```

```
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

Consulta com uma expressão de filtro criada dinamicamente usando o AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
```

```
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulta com leituras altamente consistentes usando o AWS SDK para Python (Boto3).

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
```

```
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
 reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
 ConsistentRead=consistent_read)

    return response
```

Consulta com um índice secundário global usando o AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
 KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

Consulta com paginação usando o AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
```

```
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
```

```
    """
    Query a DynamoDB table with pagination using a generator to handle large
result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
```

```
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

Consulta com filtros complexos usando o AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition
```

```
    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)
```

```
    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
```

```
    response = table.query(**query_params)
    return response
```

Consulta com uma expressão de filtro criada dinamicamente usando o AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
 {'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}
```

```
    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Consulta com uma expressão de filtro e limite usando o AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}
```

```
    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Executar operações de lista no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como realizar operações de lista no DynamoDB.

- Adicione elementos a um atributo de lista.
- Remova elementos de um atributo de lista.
- Atualize elementos específicos em uma lista por índice.
- Use as funções list.append e list.index.

Java

SDK para Java 2.x

Demonstre as operações de lista usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Appends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the end of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToAppend The items to append to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse appendToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToAppend) {

        // Create a list value from the items to append
        AttributeValue listValue =
 AttributeValue.builder().l(itemsToAppend).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName =
 list_append(if_not_exists(#attrName, :emptyList), :newItems)")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
```

```
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Prepends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the beginning of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToPrepend The items to prepend to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse prependToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToPrepend) {

        // Create a list value from the items to prepend
        AttributeValue listValue =
AttributeValue.builder().l(itemsToPrepend).build();

        // Define the update parameters
        // Note: To prepend, we put the new items first in the list_append
function
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName = list_append(:newItems,
if_not_exists(#attrName, :emptyList))")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
```

```
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates a specific element in a list attribute.
     *
     * <p>This method demonstrates how to update a specific element in a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to update
     * @param newValue The new value for the element
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index,
        AttributeValue newValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName[" + index + "] = :newValue")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(":newValue", newValue))
            .returnValues("UPDATED_NEW")
```

```
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes a specific element from a list attribute.
     *
     * <p>This method demonstrates how to remove a specific element from a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #attrName[" + index + "]")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a list attribute.
     *
     * <p>Helper method to retrieve the current value of a list attribute.
```

```
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param listAttributeName The name of the list attribute
     * @return The list attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static List<AttributeValue> getListAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String listAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(listAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the list attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(listAttributeName)) {
                return response.item().get(listAttributeName).l();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get list attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Exemplo de uso de operações de lista com o AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
```

```
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating list operations in DynamoDB");

        try {
            // Example 1: Append items to a list
            System.out.println("\nExample 1: Appending items to a list");
            List<AttributeValue> tagsToAppend = List.of(
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Gadget").build());

            UpdateItemResponse appendResponse = appendToList(dynamoDbClient,
tableName, key, "Tags", tagsToAppend);

            System.out.println("Updated list attribute: " +
appendResponse.attributes());

            // Example 2: Prepend items to a list
            System.out.println("\nExample 2: Prepending items to a list");
            List<AttributeValue> tagsToPrepend = List.of(
                AttributeValue.builder().s("Featured").build(),
                AttributeValue.builder().s("New").build());

            UpdateItemResponse prependResponse = prependToList(dynamoDbClient,
tableName, key, "Tags", tagsToPrepend);

            System.out.println("Updated list attribute: " +
prependResponse.attributes());

            // Example 3: Update a specific element in a list
            System.out.println("\nExample 3: Updating a specific element in a
list");
            UpdateItemResponse updateResponse = updateListElement(
                dynamoDbClient,
                tableName,
                key,
                "Tags",
                0,
                AttributeValue.builder().s("BestSeller").build());

            System.out.println("Updated list attribute: " +
updateResponse.attributes());
```

```
            // Example 4: Remove a specific element from a list
            System.out.println("\nExample 4: Removing a specific element from a
list");
            UpdateItemResponse removeResponse = removeListElement(dynamoDbClient,
tableName, key, "Tags", 1);

            System.out.println("Updated list attribute: " +
removeResponse.attributes());

            // Example 5: Get the current value of a list attribute
            System.out.println("\nExample 5: Getting the current value of a list
attribute");
            List<AttributeValue> currentList = getListAttribute(dynamoDbClient,
tableName, key, "Tags");

            if (currentList != null) {
                System.out.println("Current list attribute:");
                for (int i = 0; i < currentList.size(); i++) {
                    System.out.println("  [" + i + "]: " +
currentList.get(i).s());
                }
            } else {
                System.out.println("List attribute not found");
            }

            // Explain list operations
            System.out.println("\nKey points about DynamoDB list operations:");
            System.out.println("1. Lists are ordered collections of attributes");
            System.out.println("2. Use list_append to add items to a list");
            System.out.println("3. To append items, use list_append(existingList,
newItems)");
            System.out.println("4. To prepend items, use list_append(newItems,
existingList)");
            System.out.println("5. Use index notation (list[0]) to access or
update specific elements");
            System.out.println("6. Use REMOVE to delete elements from a list");
            System.out.println("7. List indices are zero-based");
            System.out.println("8. Use if_not_exists to handle the case where the
list doesn't exist yet");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demonstre as operações de lista usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
```

```
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} =
 list_append(if_not_exists(${listName}, :empty_list), :values)`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
 if_not_exists(${listName}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
```

```
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `SET ${resultListName} =
 list_append(if_not_exists(${listName1}, :empty_list),
 if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list
 structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
  config,
  tableName,
  key,
  listName,
  nestedLists
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters to create a nested list
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${listName} = :nested_lists`,
    ExpressionAttributeValues: {
      ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
```

```
}
```

Exemplo de uso de operações de lista com o AWS SDK para JavaScript.

```
/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");

  try {
    // Example 1: Append elements to a list
    console.log("\nExample 1: Appending elements to a list");
    const response1 = await appendToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["laptop", "headphones", "monitor"]
    );

    console.log("Appended to list:", response1.Attributes);

    // Example 2: Prepend elements to a list
    console.log("\nExample 2: Prepending elements to a list");
    const response2 = await prependToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["keyboard", "mouse"]
    );

    console.log("Prepended to list:", response2.Attributes);

    // Get the current state of the item
    let currentItem = await getItem(config, tableName, key);
```

```
    console.log("\nCurrent state of RecentSearches:",
currentItem?.RecentSearches);

    // Example 3: Update a specific element in a list
    console.log("\nExample 3: Updating a specific element in a list");
    const response3 = await updateListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      0, // Update the first element
      "mechanical keyboard" // New value
    );

    console.log("Updated list element:", response3.Attributes);

    // Example 4: Remove an element from a list
    console.log("\nExample 4: Removing an element from a list");
    const response4 = await removeListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      2 // Remove the third element
    );

    console.log("List after removing element:", response4.Attributes);

    // Example 5: Create and concatenate lists
    console.log("\nExample 5: Creating and concatenating lists");

    // First, create two separate lists
    await updateWithMultipleActions(
      config,
      tableName,
      key,
      "SET WishList = :wishlist, SavedItems = :saveditems",
      null,
      {
        ":wishlist": ["gaming laptop", "wireless earbuds"],
        ":saveditems": ["smartphone", "tablet"]
      }
    );
```

```
    // Then, concatenate them
    const response5 = await concatenateLists(
      config,
      tableName,
      key,
      "WishList",
      "SavedItems",
      "AllItems"
    );

    console.log("Concatenated lists:", response5.Attributes);

    // Example 6: Create a nested list structure
    console.log("\nExample 6: Creating a nested list structure");
    const response6 = await createNestedList(
      config,
      tableName,
      key,
      "Categories",
      [
        ["Electronics", "Computers", "Accessories"],
        ["Books", "Magazines", "E-books"],
        ["Clothing", "Shoes", "Watches"]
      ]
    );

    console.log("Created nested list:", response6.Attributes);

    // Example 7: Update an element in a nested list
    console.log("\nExample 7: Updating an element in a nested list");
    const response7 = await updateNestedListElement(
      config,
      tableName,
      key,
      "Categories",
      0, // First inner list
      1, // Second element in that list
      "Laptops" // New value
    );

    console.log("Updated nested list element:", response7.Attributes);

    // Get the final state of the item
    currentItem = await getItem(config, tableName, key);
```

```
    console.log("\nFinal state of the item:", JSON.stringify(currentItem, null,
 2));

    // Explain list operations
    console.log("\nKey points about list operations in DynamoDB:");
    console.log("1. Use list_append to add elements to a list");
    console.log("2. To append elements, use list_append(existingList,
 newElements)");
    console.log("3. To prepend elements, use list_append(newElements,
 existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not
 exist yet");
    console.log("5. Use index notation (list[0]) to access or update specific
 elements");
    console.log("6. Use REMOVE with index notation to remove elements from a
 list");
    console.log("7. Lists can contain elements of different types");
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested
 list elements");

  } catch (error) {
    console.error("Error:", error);
  }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name
 placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
```

```
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demonstre as operações de lista usando o AWS SDK para Python (Boto3).

```
import boto3
import json
```

```
from typing import Any, Dict, List, Optional, Union


def create_list_attribute(
    table_name: str, key: Dict[str, Any], list_name: str, list_values: List[Any]
) -> Dict[str, Any]:
    """
    Create a new list attribute or replace an existing one.

    This function demonstrates how to create a new list attribute or replace
    an existing list with new values.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        list_values (List[Any]): The values to set in the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation to create or replace the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = :list_values",
        ExpressionAttributeValues={":list_values": list_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_append:
 List[Any]
) -> Dict[str, Any]:
    """
    Append values to the end of a list attribute.
```

```
    This function demonstrates how to use the list_append function to add
 elements
    to the end of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append to add values to the end of the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append({list_name}, :values)",
        ExpressionAttributeValues={":values": values_to_append},
        ReturnValues="UPDATED_NEW",
    )

    return response


def prepend_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_prepend:
 List[Any]
) -> Dict[str, Any]:
    """
    Prepend values to the beginning of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the beginning of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
```

```
        values_to_prepend (List[Any]): The values to prepend to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append with reversed order to add values to the beginning of the
 list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append(:values, {list_name})",
        ExpressionAttributeValues={":values": values_to_prepend},
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int, new_value:
 Any
) -> Dict[str, Any]:
    """
    Update a specific element in a list attribute.

    This function demonstrates how to update a specific element in a list
 attribute
    using the index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to update.
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the index notation to update a specific element
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name}[{index}] = :value",
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int
) -> Dict[str, Any]:
    """
    Remove a specific element from a list attribute.

    This function demonstrates how to remove a specific element from a list
 attribute
    using the REMOVE action with index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to remove.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the REMOVE action with index notation to remove a specific element
    response = table.update_item(
        Key=key, UpdateExpression=f"REMOVE {list_name}[{index}]",
 ReturnValues="UPDATED_NEW"
    )
```

```
    return response


def update_nested_list_element(
    table_name: str, key: Dict[str, Any], path: str, new_value: Any
) -> Dict[str, Any]:
    """
    Update an element in a nested list structure.

    This function demonstrates how to update an element in a nested list
 structure
    using expression attribute names for the path components.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested element (e.g., "parent[0].child[1]").
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Define a type for path parts
    path_part = Dict[str, Union[str, int]]
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Parse the path to extract attribute names and indices
    path_parts: List[path_part] = []
    current_part = ""
    in_bracket = False

    for char in path:
        if char == "[":
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
            in_bracket = True
        elif char == "]":
            if current_part:
                # Fix for mypy: Use a properly typed dictionary with Union type
```

```
                path_parts.append({"type": "index", "value": int(current_part)})
                current_part = ""
            in_bracket = False
        elif char == "." and not in_bracket:
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
        else:
            current_part += char

    if current_part:
        path_parts.append({"type": "attribute", "value": current_part})

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        if part["type"] == "attribute":
            name_placeholder = f"#attr{i}"
            expression_attribute_names[name_placeholder] = part["value"]

            if path_expression:
                path_expression += "."
            path_expression += name_placeholder
        elif part["type"] == "index":
            path_expression += f"[{part['value']}]"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def create_list_if_not_exists(
    table_name: str, key: Dict[str, Any], list_name: str, default_values:
 List[Any]
) -> Dict[str, Any]:
    """
    Create a list attribute if it doesn't exist.

    This function demonstrates how to use if_not_exists to create a list
 attribute
    with default values if it doesn't already exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        default_values (List[Any]): The default values for the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to create the list if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 if_not_exists({list_name}, :default)",
        ExpressionAttributeValues={":default": default_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list_safely(
    table_name: str,
    key: Dict[str, Any],
    list_name: str,
    values_to_append: List[Any],
    default_values: Optional[List[Any]] = None,
```

```
) -> Dict[str, Any]:
    """
    Append values to a list, creating it if it doesn't exist.

    This function demonstrates how to safely append values to a list attribute,
    creating the list with default values if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.
        default_values (Optional[List[Any]]): The default values if the list
 doesn't exist.
            If not provided, values_to_append will be used as the default.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # If default_values is not provided, use values_to_append
    if default_values is None:
        default_values = values_to_append

    # Use if_not_exists with list_append to safely append to the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 list_append(if_not_exists({list_name}, :default), :values)",
        ExpressionAttributeValues={
            ":default": default_values if default_values else [],
            ":values": values_to_append,
        },
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Exemplo de uso de operações de lista com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use list operations in DynamoDB."""
    # Example parameters
    table_name = "UserData"
    key = {"UserId": "user123"}

    print("Example 1: Creating a list attribute")
    try:
        response = create_list_attribute(
            table_name=table_name,
            key=key,
            list_name="Interests",
            list_values=["Reading", "Hiking", "Photography"],
        )
        print(
            f"List attribute created successfully:
 {json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list attribute: {e}")

    print("\nExample 2: Appending values to a list")
    try:
        response = append_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_append=["Cooking", "Gardening"],
        )
        print(
            f"Values appended to list successfully:
 {json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error appending to list: {e}")

    print("\nExample 3: Prepending values to a list")
    try:
        response = prepend_to_list(
```

```
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_prepend=["Travel", "Music"],
        )
        print(
            f"Values prepended to list successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error prepending to list: {e}")

    print("\nExample 4: Updating a specific list element")
    try:
        response = update_list_element(
            table_name=table_name,
            key=key,
            list_name="Interests",
            index=2,
            new_value="Mountain Hiking",
        )
        print(
            f"List element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating list element: {e}")

    print("\nExample 5: Removing a list element")
    try:
        response = remove_list_element(
            table_name=table_name, key=key, list_name="Interests", index=0
        )
        print(
            f"List element removed successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error removing list element: {e}")

    print("\nExample 6: Working with nested lists")
    try:
        # First, create an item with a nested structure
        dynamodb = boto3.resource("dynamodb")
```

```
        table = dynamodb.Table(table_name)

        table.update_item(
            Key={"UserId": "user456"},
            UpdateExpression="SET #skills = :skills",
            ExpressionAttributeNames={"#skills": "Skills"},
            ExpressionAttributeValues={
                ":skills": [
                    {"Category": "Programming", "Languages": ["Python", "Java",
"JavaScript"]},
                    {"Category": "Database", "Systems": ["DynamoDB", "MongoDB",
"PostgreSQL"]},
                ]
            },
        )

        # Now update a nested element
        response = update_nested_list_element(
            table_name=table_name,
            key={"UserId": "user456"},
            path="Skills[0].Languages[1]",
            new_value="TypeScript",
        )
        print(
            f"Nested list element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error working with nested lists: {e}")

    print("\nExample 7: Creating a list if it doesn't exist")
    try:
        response = create_list_if_not_exists(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Preferences",
            default_values=["Default1", "Default2", "Default3"],
        )
        print(
            f"List created with default values:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list with default values: {e}")
```

```
    print("\nExample 8: Safely appending to a list")
    try:
        response = append_to_list_safely(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Notifications",
            values_to_append=["New message received"],
            default_values=[],
        )
        print(f"Safely appended to list: {json.dumps(response.get('Attributes',
{}), default=str)}")
    except Exception as e:
        print(f"Error safely appending to list: {e}")

    print("\nKey Points About Working with Lists in DynamoDB:")
    print("1. Lists are ordered collections of elements that can be of different
types")
    print("2. Use the SET operation with direct assignment to create or replace a
list")
    print("3. Use list_append() to add elements to a list without replacing the
entire list")
    print("4. To append to the end: list_append(list_name, :values)")
    print("5. To prepend to the beginning: list_append(:values, list_name)")
    print("6. Use index notation list_name[index] to access or update specific
elements")
    print("7. Use the REMOVE action with index notation to remove specific
elements")
    print("8. Lists can contain nested structures like maps and other lists")
    print("9. Use if_not_exists() to create a list with default values if it
doesn't exist")
    print("10. List indices are zero-based (the first element is at index 0)")
    print("11. Attempting to access an index beyond the list bounds will result
in an error")
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Executar operações de mapa no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como realizar operações de mapa no DynamoDB.

- Adicione e atualize atributos aninhados em estruturas de mapa.
- Remova campos específicos dos mapas.
- Trabalhe com atributos de mapa profundamente aninhados.

Java

SDK para Java 2.x

Demonstre as operações de mapa usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates a map attribute that may not exist.
     *
     * <p>This method demonstrates how to safely update a map attribute
     * by using if_not_exists to handle the case where the map doesn't exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to update
     * @param value The value to set
```

```
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateMapAttributeSafe(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey,
        AttributeValue value) {

        // Create an empty map to use if the map doesn't exist
        Map<String, AttributeValue> emptyMap = new HashMap<>();
        AttributeValue emptyMapValue =
AttributeValue.builder().m(emptyMap).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = if_not_exists(#mapName, :emptyMap),
#mapName.#mapKey = :value")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .expressionAttributeValues(Map.of(
                ":value",
                value,
                ":emptyMap",
                AttributeValue.builder().m(new HashMap<>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds an attribute to a nested map.
     *
     * <p>This method demonstrates how to update a nested attribute without
     * overwriting the entire map.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param path The path to the nested attribute as a list
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNestedMap(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> path,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < path.size(); i++) {
            expressionAttributeNames.put("#attr" + i, path.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < path.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
 = :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }
```

```
    /**
     * Removes an attribute from a map.
     *
     * <p>This method demonstrates how to remove a specific attribute from a map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeMapAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #mapName.#mapKey")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Creates a map with multiple attributes in a single operation.
     *
     * <p>This method demonstrates how to create a map with multiple attributes
     * in a single update operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
```

```
     * @param mapName The name of the map attribute
     * @param attributes The attributes to set in the map
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createMapWithAttributes(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        Map<String, AttributeValue> attributes) {

        // Create a map value from the attributes
        AttributeValue mapValue = AttributeValue.builder().m(attributes).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = :mapValue")
            .expressionAttributeNames(Map.of("#mapName", mapName))
            .expressionAttributeValues(Map.of(":mapValue", mapValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a map attribute.
     *
     * <p>Helper method to retrieve the current value of a map attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param mapName The name of the map attribute
     * @return The map attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, AttributeValue> getMapAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String mapName) {
```

```
        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(mapName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the map attribute if it exists, otherwise null
            if (response.item() != null && response.item().containsKey(mapName))
{
                return response.item().get(mapName).m();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get map attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Exemplo de uso de operações de mapa com o AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating map operations in DynamoDB");

        try {
            // Example 1: Create a map with multiple attributes
            System.out.println("\nExample 1: Creating a map with multiple
attributes");
            Map<String, AttributeValue> productDetails = new HashMap<>();
```

```
            productDetails.put("Color",
AttributeValue.builder().s("Red").build());
            productDetails.put("Weight",
AttributeValue.builder().n("2.5").build());
            productDetails.put(
                "Dimensions", AttributeValue.builder().s("10x20x5").build());

            UpdateItemResponse createResponse =
                createMapWithAttributes(dynamoDbClient, tableName, key,
"Details", productDetails);

            System.out.println("Created map attribute: " +
createResponse.attributes());

            // Example 2: Update a specific attribute in a map
            System.out.println("\nExample 2: Updating a specific attribute in a
map");
            UpdateItemResponse updateResponse = updateMapAttributeSafe(
                dynamoDbClient,
                tableName,
                key,
                "Details",
                "Color",
                AttributeValue.builder().s("Blue").build());

            System.out.println("Updated map attribute: " +
updateResponse.attributes());

            // Example 3: Add an attribute to a nested map
            System.out.println("\nExample 3: Adding an attribute to a nested
map");
            UpdateItemResponse nestedResponse = addToNestedMap(
                dynamoDbClient,
                tableName,
                key,
                List.of("Specifications", "Technical", "Resolution"),
                AttributeValue.builder().s("1920x1080").build());

            System.out.println("Added to nested map: " +
nestedResponse.attributes());

            // Example 4: Remove an attribute from a map
            System.out.println("\nExample 4: Removing an attribute from a map");
            UpdateItemResponse removeResponse =
```

```
                removeMapAttribute(dynamoDbClient, tableName, key, "Details",
"Dimensions");

            System.out.println("Updated map after removal: " +
removeResponse.attributes());

            // Example 5: Get the current value of a map attribute
            System.out.println("\nExample 5: Getting the current value of a map
attribute");
            Map<String, AttributeValue> currentMap =
getMapAttribute(dynamoDbClient, tableName, key, "Details");

            if (currentMap != null) {
                System.out.println("Current map attribute:");
                for (Map.Entry<String, AttributeValue> entry :
currentMap.entrySet()) {
                    System.out.println("  " + entry.getKey() + ": " +
entry.getValue());
                }
            } else {
                System.out.println("Map attribute not found");
            }

            // Explain map operations
            System.out.println("\nKey points about DynamoDB map operations:");
            System.out.println("1. Maps are unordered collections of name-value
pairs");
            System.out.println("2. Use dot notation (map.key) to access or update
specific attributes");
            System.out.println("3. You can update individual attributes without
overwriting the entire map");
            System.out.println("4. Maps can be nested to create complex data
structures");
            System.out.println("5. Use REMOVE to delete attributes from a map");
            System.out.println("6. You can create a map with multiple attributes
in a single operation");
            System.out.println("7. Map keys are case-sensitive");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demonstre as operações de mapa usando o AWS SDK para JavaScript.

```
/**
 * Example of updating map attributes in DynamoDB.
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update
 scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not
 exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeSafe(
```

```
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update
 expression is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };

      return await docClient.send(new UpdateCommand(createParams));
```

```
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
 ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
```

```
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
      updateExpression += ` ${parentPath} =
 if_not_exists(${parentPath}, :emptyMap${i}),`;
```

```
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[":value"] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
  updates
) {
```

```
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
```

```
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
 DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
    );
```

```
    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists
(elegant approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );

    console.log("Updated settings:", response2.Attributes);

    // Example 3: Update a deeply nested attribute
    console.log("\nExample 3: Updating a deeply nested attribute");
    const response3 = await addToNestedMap(
      config,
      tableName,
      key,
      ["Profile", "Address", "City"],
      "Seattle"
    );

    console.log("Updated nested attribute:", response3.Attributes);

    // Example 4: Update multiple fields in a map
    console.log("\nExample 4: Updating multiple fields in a map");
    const response4 = await updateMultipleMapFields(
      config,
      tableName,
      key,
      "ContactInfo",
      {
        Email: "user@example.com",
        Phone: "555-123-4567",
        PreferredContact: "Email"
      }
    );

    console.log("Updated multiple fields:", response4.Attributes);
```

```
    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Explain the benefits of different approaches
    console.log("\nKey points about updating map attributes:");
    console.log("1. Use if_not_exists to handle maps that might not exist");
    console.log("2. Multiple updates can be combined in a single operation");
    console.log("3. Deeply nested attributes require creating parent maps");
    console.log("4. DynamoDB expressions are atomic - the entire update succeeds
 or fails");
    console.log("5. Using a single operation is more efficient than multiple
 separate updates");

  } catch (error) {
    console.error("Error:", error);
  }
}

// Export the functions
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
  exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
  exampleUsage();
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demonstre as operações de mapa usando o AWS SDK para Python (Boto3).

```
"""
Example of updating map attributes in DynamoDB.

This module demonstrates how to update map attributes in DynamoDB, including
handling cases where the map attribute might not exist yet.
"""


import boto3
from typing import Any, Dict, Optional



def update_map_attribute_safe(
    table_name: str, key: Dict[str, Any], map_name: str, map_key: str, value: Any
) -> Dict[str, Any]:
    """
    Update a specific key in a map attribute, creating the map if it doesn't
 exist.

    This function demonstrates how to safely update a key within a map attribute,
    even if the map doesn't exist yet in the item.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Use SET with attribute_not_exists to safely update the map
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map.#key = :value",
        ExpressionAttributeNames={"#map": map_name, "#key": map_key},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_nested_map(
    table_name: str, key: Dict[str, Any], path: str, value: Any
) -> Dict[str, Any]:
    """
    Add or update a value in a deeply nested map structure.

    This function demonstrates how to update a value at a specific path in a
    nested map structure, creating any intermediate maps as needed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested attribute (e.g.,
 "user.preferences.theme").
        value (Any): The value to set at the specified path.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}
```

```
    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_map_with_if_not_exists(
    table_name: str,
    key: Dict[str, Any],
    map_name: str,
    map_key: str,
    value: Any,
    default_map: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update a key in a map, creating the map with default values if it doesn't
 exist.

    This function demonstrates how to use if_not_exists to initialize a map with
    default values if it doesn't exist yet, and then update a specific key.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
```

```
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.
        default_map (Optional[Dict[str, Any]]): Default map values if the map
 doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Set default map if not provided
    if default_map is None:
        default_map = {}

    # Create a map with the new key-value pair
    updated_map = default_map.copy()
    updated_map[map_key] = value

    # Use if_not_exists to initialize the map if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map = if_not_exists(#map, :default_map)",
        ExpressionAttributeNames={"#map": map_name},
        ExpressionAttributeValues={":default_map": updated_map},
        ReturnValues="UPDATED_NEW",
    )

    return response


def merge_into_map(
    table_name: str, key: Dict[str, Any], map_name: str, values_to_merge:
 Dict[str, Any]
) -> Dict[str, Any]:
    """
    Merge multiple key-value pairs into a map attribute.

    This function demonstrates how to update multiple keys in a map attribute
    in a single operation, without overwriting the entire map.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        values_to_merge (Dict[str, Any]): Key-value pairs to merge into the map.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression for each key-value pair
    update_expression = "SET "
    expression_attribute_names = {"#map": map_name}
    expression_attribute_values = {}

    # Add each key-value pair to the update expression
    for i, (k, v) in enumerate(values_to_merge.items()):
        key_placeholder = f"#key{i}"
        value_placeholder = f":value{i}"

        expression_attribute_names[key_placeholder] = k
        expression_attribute_values[value_placeholder] = v

        if i > 0:
            update_expression += ", "
        update_expression += f"#map.{key_placeholder} = {value_placeholder}"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def example_usage():
    """Example of how to use the map attribute update functions."""
    # Example parameters
    table_name = "UserProfiles"
    key = {"UserId": "user123"}

    print("Example 1: Updating a specific key in a map attribute")
    try:
        response = update_map_attribute_safe(
            table_name=table_name, key=key, map_name="Preferences",
 map_key="Theme", value="Dark"
        )
        print(f"Map attribute updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating map attribute: {e}")

    print("\nExample 2: Adding a value to a deeply nested map")
    try:
        response = add_to_nested_map(
            table_name=table_name, key=key, path="Settings.Notifications.Email",
 value=True
        )
        print(f"Nested map updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating nested map: {e}")

    print("\nExample 3: Initializing a map with default values if it doesn't
 exist")
    try:
        default_map = {"Language": "English", "Currency": "USD", "TimeZone":
 "UTC"}

        response = update_map_with_if_not_exists(
            table_name=table_name,
            key={"UserId": "newuser456"},
            map_name="Preferences",
            map_key="Theme",
            value="Light",
            default_map=default_map,
        )
        print(f"Map initialized with defaults: {response.get('Attributes', {})}")
```

```
    except Exception as e:
        print(f"Error initializing map: {e}")

    print("\nExample 4: Merging multiple values into a map")
    try:
        values_to_merge = {
            "NotificationsEnabled": True,
            "EmailFrequency": "Daily",
            "PushNotifications": False,
        }

        response = merge_into_map(
            table_name=table_name,
            key=key,
            map_name="NotificationSettings",
            values_to_merge=values_to_merge,
        )
        print(f"Multiple values merged into map: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error merging values into map: {e}")

    print("\nBest practices for working with map attributes in DynamoDB:")
    print("1. Use dot notation to access and update nested attributes")
    print("2. Use ExpressionAttributeNames to handle reserved words and special
 characters")
    print("3. Use if_not_exists() to handle cases where attributes might not
 exist")
    print("4. Update specific map keys rather than overwriting the entire map")
    print("5. Use a single update operation to modify multiple map keys for
 better performance")
    print("6. Consider your data model carefully to minimize the need for deeply
 nested attributes")



if __name__ == "__main__":
    example_usage()
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Executar operações de conjunto no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como realizar operações de conjunto no DynamoDB.

- Adicione elementos a um atributo de conjunto.
- Remova elementos de um atributo de conjunto.
- Use as operações ADD e DELETE com conjuntos.

Java

SDK para Java 2.x

Demonstre as operações de conjunto usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

    /**
     * Adds values to a string set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a string set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
```

```
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToStringSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<String> valuesToAdd) {

        // Create a string set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ss(valuesToAdd).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds values to a number set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a number set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNumberSet(
```

```
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<Number> valuesToAdd) {

        // Convert numbers to strings for DynamoDB
        Set<String> stringValues = new HashSet<>();
        for (Number value : valuesToAdd) {
            stringValues.add(value.toString());
        }

        // Create a number set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ns(stringValues).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes values from a set attribute.
     *
     * <p>This method demonstrates how to use the DELETE operation to remove
values
     * from a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToRemove The values to remove from the set
     * @param isNumberSet Whether the set is a number set (true) or string set
(false)
```

```
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeFromSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> valuesToRemove,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("DELETE #setAttr :valuesToRemove")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToRemove", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
```

```
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Checks if a value exists in a set attribute.
     *
     * <p>This method demonstrates how to use the contains function to check
     * if a value exists in a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to check
     * @param setAttributeName The name of the set attribute
     * @param valueToCheck The value to check for
     * @return Map containing the result of the check
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> checkIfValueInSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        String valueToCheck) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters with a condition expression
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #tempAttr = :tempVal")
                .conditionExpression("contains(#setAttr, :valueToCheck)")
                .expressionAttributeNames(Map.of("#setAttr", setAttributeName,
"#tempAttr", "TempAttribute"))
                .expressionAttributeValues(Map.of(
                    ":valueToCheck",
AttributeValue.builder().s(valueToCheck).build(),
                    ":tempVal", AttributeValue.builder().s("TempValue").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();
```

```
            // Attempt the update operation
            dynamoDbClient.updateItem(request);

            // If we get here, the condition was met
            result.put("exists", true);
            result.put("message", "Value '" + valueToCheck + "' exists in the
set");

            // Clean up the temporary attribute
            UpdateItemRequest cleanupRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("REMOVE #tempAttr")
                .expressionAttributeNames(Map.of("#tempAttr", "TempAttribute"))
                .build();

            dynamoDbClient.updateItem(cleanupRequest);

        } catch (DynamoDbException e) {
            if (e.getMessage().contains("ConditionalCheckFailed")) {
                // The condition was not met
                result.put("exists", false);
                result.put("message", "Value '" + valueToCheck + "' does not
exist in the set");
            } else {
                // Some other error occurred
                result.put("exists", false);
                result.put("message", "Error checking set: " + e.getMessage());
                result.put("error", e.getClass().getSimpleName());
            }
        }

        return result;
    }

    /**
     * Creates a set with multiple values in a single operation.
     *
     * <p>This method demonstrates how to create a set with multiple values
     * in a single update operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param setValues The values to include in the set
     * @param isNumberSet Whether to create a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createSetWithValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> setValues,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
```

```
            .key(key)
            .updateExpression("SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a set attribute.
     *
     * <p>Helper method to retrieve the current value of a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param setAttributeName The name of the set attribute
     * @return The set attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static AttributeValue getSetAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String setAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(setAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the set attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(setAttributeName)) {
                return response.item().get(setAttributeName);
            }
```

```
            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get set attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Exemplo de uso de operações de conjunto com o AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating set operations in DynamoDB");

        try {
            // Example 1: Create a string set with multiple values
            System.out.println("\nExample 1: Creating a string set with multiple
values");
            Set<String> tags = new HashSet<>();
            tags.add("Electronics");
            tags.add("Gadget");
            tags.add("Smartphone");

            UpdateItemResponse createResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Tags", tags, false // Not a
number set
                );

            System.out.println("Created set attribute: " +
createResponse.attributes());

            // Example 2: Add values to a string set
            System.out.println("\nExample 2: Adding values to a string set");
            Set<String> additionalTags = new HashSet<>();
            additionalTags.add("Mobile");
            additionalTags.add("Wireless");
```

```
            UpdateItemResponse addResponse = addToStringSet(dynamoDbClient,
tableName, key, "Tags", additionalTags);

            System.out.println("Updated set attribute: " +
addResponse.attributes());

            // Example 3: Create a number set with multiple values
            System.out.println("\nExample 3: Creating a number set with multiple
values");
            Set<Number> ratings = new HashSet<>();
            ratings.add(4);
            ratings.add(5);
            ratings.add(4.5);

            UpdateItemResponse createNumberSetResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Ratings", ratings, true // Is a
number set
                );

            System.out.println("Created number set attribute: " +
createNumberSetResponse.attributes());

            // Example 4: Add values to a number set
            System.out.println("\nExample 4: Adding values to a number set");
            Set<Number> additionalRatings = new HashSet<>();
            additionalRatings.add(3.5);
            additionalRatings.add(4.2);

            UpdateItemResponse addNumberResponse =
                addToNumberSet(dynamoDbClient, tableName, key, "Ratings",
additionalRatings);

            System.out.println("Updated number set attribute: " +
addNumberResponse.attributes());

            // Example 5: Remove values from a set
            System.out.println("\nExample 5: Removing values from a set");
            Set<String> tagsToRemove = new HashSet<>();
            tagsToRemove.add("Gadget");

            UpdateItemResponse removeResponse = removeFromSet(
                dynamoDbClient, tableName, key, "Tags", tagsToRemove, false //
Not a number set
                );
```

```
            System.out.println("Updated set after removal: " +
removeResponse.attributes());

            // Example 6: Check if a value exists in a set
            System.out.println("\nExample 6: Checking if a value exists in a
set");
            Map<String, Object> checkResult = checkIfValueInSet(dynamoDbClient,
tableName, key, "Tags", "Electronics");

            System.out.println("Check result: " + checkResult.get("message"));

            // Example 7: Get the current value of a set attribute
            System.out.println("\nExample 7: Getting the current value of a set
attribute");
            AttributeValue currentStringSet = getSetAttribute(dynamoDbClient,
tableName, key, "Tags");

            if (currentStringSet != null && currentStringSet.ss() != null) {
                System.out.println("Current string set values: " +
currentStringSet.ss());
            } else {
                System.out.println("String set attribute not found");
            }

            AttributeValue currentNumberSet = getSetAttribute(dynamoDbClient,
tableName, key, "Ratings");

            if (currentNumberSet != null && currentNumberSet.ns() != null) {
                System.out.println("Current number set values: " +
currentNumberSet.ns());
            } else {
                System.out.println("Number set attribute not found");
            }

            // Explain set operations
            System.out.println("\nKey points about DynamoDB set operations:");
            System.out.println(
                "1. DynamoDB supports three set types: string sets (SS), number
sets (NS), and binary sets (BS)");
            System.out.println("2. Sets can only contain elements of the same
type");
            System.out.println("3. Use ADD to add elements to a set");
            System.out.println("4. Use DELETE to remove elements from a set");
```

```
            System.out.println("5. Sets automatically remove duplicate values");
            System.out.println("6. Sets are unordered collections");
            System.out.println("7. Use the contains function to check if a value
 exists in a set");
            System.out.println("8. You can create a set with multiple values in a
 single operation");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demonstre as operações de conjunto usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
```

```
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove elements from a set attribute.
 *
```

```
 * This function demonstrates using the DELETE operation to remove elements from
 a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
```

```
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set
 attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
```

```
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}
```

```
/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }

  // Get the set values
  const setValues = Array.from(currentItem[setName]);

  // If there's only one element left, remove the attribute entirely
  if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `REMOVE ${setName}`,
```

```
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Last element removed, attribute has been deleted",
      removedValue: setValues[0]
    };
  } else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
      config,
      tableName,
      key,
      setName,
      [lastElement],
      setType
    );

    return {
      success: true,
      message: "Last element removed, set still contains elements",
      removedValue: lastElement,
      remainingSet: response.Attributes[setName]
    };
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
```

```
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemplo de uso de operações de conjunto com o AWS SDK para JavaScript.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(
      config,
```

```
      tableName,
      key,
      "Interests",
      ["Reading", "Hiking", "Cooking"],
      "string"
    );

    console.log("Created set:", response1.Attributes);

    // Example 2: Add elements to a set
    console.log("\nExample 2: Adding elements to a set");
    const response2 = await addToSet(
      config,
      tableName,
      key,
      "Interests",
      ["Photography", "Travel"],
      "string"
    );

    console.log("Updated set after adding elements:", response2.Attributes);

    // Example 3: Remove elements from a set
    console.log("\nExample 3: Removing elements from a set");
    const response3 = await removeFromSet(
      config,
      tableName,
      key,
      "Interests",
      ["Cooking"],
      "string"
    );

    console.log("Updated set after removing elements:", response3.Attributes);

    // Example 4: Create a number set
    console.log("\nExample 4: Creating a number set");
    const response4 = await createSet(
      config,
      tableName,
      key,
      "FavoriteNumbers",
      [7, 42, 99],
      "number"
```

```
    );

    console.log("Created number set:", response4.Attributes);

    // Example 5: Replace an entire set
    console.log("\nExample 5: Replacing an entire set");
    const response5 = await replaceSet(
      config,
      tableName,
      key,
      "Interests",
      ["Gaming", "Movies", "Music"],
      "string"
    );

    console.log("Replaced set:", response5.Attributes);

    // Example 6: Remove the last element from a set
    console.log("\nExample 6: Removing the last element from a set");

    // First, create a set with just one element
    await createSet(
      config,
      tableName,
      { UserId: "U67890" },
      "Tags",
      ["LastTag"],
      "string"
    );

    // Then, remove the last element
    const response6 = await removeLastElementFromSet(
      config,
      tableName,
      { UserId: "U67890" },
      "Tags"
    );

    console.log(response6.message);
    console.log("Removed value:", response6.removedValue);

    // Get the final state of the items
    console.log("\nFinal state of the items:");
    const item1 = await getItem(config, tableName, key);
```

```
    console.log("User U12345:", JSON.stringify(item1, null, 2));

    const item2 = await getItem(config, tableName, { UserId: "U67890" });
    console.log("User U67890:", JSON.stringify(item2, null, 2));

    // Explain set operations
    console.log("\nKey points about set operations in DynamoDB:");
    console.log("1. Use ADD to add elements to a set (duplicates are
 automatically removed)");
    console.log("2. Use DELETE to remove elements from a set");
    console.log("3. Use SET to create a new set or replace an existing one");
    console.log("4. DynamoDB supports three types of sets: string sets, number
 sets, and binary sets");
    console.log("5. When you delete the last element from a set, the attribute
 remains as an empty set");
    console.log("6. To remove an empty set, use the REMOVE operation");
    console.log("7. Sets automatically maintain unique values (no duplicates)");
    console.log("8. You cannot mix data types within a set");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demonstre as operações de conjunto usando o AWS SDK para Python (Boto3).

```
import boto3
from typing import Any, Dict, List


def create_set_attribute(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    set_values: List[Any],
```

```
    set_type: str = "string",
) -> Dict[str, Any]:
    """
    Create a new set attribute or add elements to an existing set.

    This function demonstrates how to use the ADD operation to create a new set
    or add elements to an existing set.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        set_values (List[Any]): The values to add to the set.
        set_type (str, optional): The type of set to create: "string", "number",
or "binary".
            Defaults to "string".

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a DynamoDB set based on the specified type
    if set_type == "string":
        dynamo_set = set(str(value) for value in set_values)
    elif set_type == "number":
        # We need to use actual float values for the DynamoDB API
        # but mypy expects strings in sets, so we need to use type: ignore
        dynamo_set = set(float(value) for value in set_values)  # type: ignore
    else:  # binary set is not directly supported in high-level API, handled
differently
        raise ValueError("Binary sets are not supported in this example")

    # Use the ADD operation to create or update the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :set_values",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":set_values": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def add_to_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_add: List[Any]
) -> Dict[str, Any]:
    """
    Add elements to an existing set attribute.

    This function demonstrates how to use the ADD operation to add elements to an
 existing set.
    If the set doesn't exist, it will be created.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_add (List[Any]): The values to add to the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_add)

    # Use the ADD operation to add values to the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :values_to_add",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_add": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_from_set(
```

```
    table_name: str, key: Dict[str, Any], set_name: str, values_to_remove:
 List[Any]
) -> Dict[str, Any]:
    """
    Remove elements from a set attribute.

    This function demonstrates how to use the DELETE operation to remove elements
 from a set.
    If the last element is removed, the attribute will be deleted entirely.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_remove (List[Any]): The values to remove from the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_remove)

    # Use the DELETE operation to remove values from the set
    response = table.update_item(
        Key=key,
        UpdateExpression="DELETE #set_attr :values_to_remove",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_remove": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def check_if_set_exists(table_name: str, key: Dict[str, Any], set_name: str) ->
 bool:
    """
    Check if a set attribute exists in an item.
```

```
    This function demonstrates how to check if a set attribute exists after
    potentially removing all elements from it.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to check.
        set_name (str): The name of the set attribute.

    Returns:
        bool: True if the set attribute exists, False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Get the item
    response = table.get_item(
        Key=key, ProjectionExpression="#set_attr",
 ExpressionAttributeNames={"#set_attr": set_name}
    )

    # Check if the item exists and has the set attribute
    return "Item" in response and set_name in response["Item"]


def demonstrate_last_element_removal(
    table_name: str, key: Dict[str, Any], set_name: str
) -> Dict[str, Any]:
    """
    Demonstrate what happens when you remove the last element from a set.

    This function creates a set with a single element, then removes that element,
    showing that the attribute is completely removed when the last element is
 deleted.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Step 1: Create a set with a single element
```

```
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=["last_element"],
        set_type="string",
    )

    # Step 2: Check that the set exists
    exists_before = check_if_set_exists(table_name, key, set_name)

    # Step 3: Remove the last element
    delete_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_remove=["last_element"]
    )

    # Step 4: Check if the set still exists
    exists_after = check_if_set_exists(table_name, key, set_name)

    # Return the results
    return {
        "create_response": create_response,
        "exists_before": exists_before,
        "delete_response": delete_response,
        "exists_after": exists_after,
    }


def work_with_number_set(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    initial_values: List[float],
    values_to_add: List[float],
    values_to_remove: List[float],
) -> Dict[str, Any]:
    """
    Demonstrate working with a number set in DynamoDB.

    This function shows how to create and manipulate a set of numbers.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        initial_values (List[float]): The initial values for the set.
        values_to_add (List[float]): Values to add to the set.
        values_to_remove (List[float]): Values to remove from the set.

    Returns:
        Dict[str, Any]: A dictionary containing the responses from each
operation.
    """
    # Step 1: Create the number set
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=initial_values,
        set_type="number",
    )

    # Step 2: Add more numbers to the set
    add_response = add_to_set(
        table_name=table_name, key=key, set_name=set_name,
values_to_add=values_to_add
    )

    # Step 3: Remove some numbers from the set
    remove_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
values_to_remove=values_to_remove
    )

    # Step 4: Get the final state
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    get_response = table.get_item(
        Key=key,
        ProjectionExpression=f"#{set_name}",
        ExpressionAttributeNames={f"#{set_name}": set_name},
    )

    # Return all responses
    return {
        "create_response": create_response,
```

```
        "add_response": add_response,
        "remove_response": remove_response,
        "final_state": get_response.get("Item", {}),
    }
```

Exemplo de uso de operações de conjunto com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use the set operations functions."""
    # Example parameters
    table_name = "UserPreferences"
    key = {"UserId": "user123"}

    print("Example 1: Creating a string set attribute")
    try:
        response = create_set_attribute(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            set_values=["AWS", "DynamoDB", "NoSQL"],
            set_type="string",
        )
        print(f"Set attribute created successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error creating set attribute: {e}")

    print("\nExample 2: Adding elements to an existing set")
    try:
        response = add_to_set(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            values_to_add=["Database", "Serverless"],
        )
        print(f"Elements added to set successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error adding to set: {e}")
```

```
    print("\nExample 3: Removing elements from a set")
    try:
        response = remove_from_set(
            table_name=table_name, key=key, set_name="FavoriteTags",
values_to_remove=["NoSQL"]
        )
        print(f"Elements removed from set successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error removing from set: {e}")

    print("\nExample 4: Demonstrating what happens when you remove the last
element from a set")
    try:
        results = demonstrate_last_element_removal(
            table_name=table_name, key={"UserId": "tempUser"},
set_name="SingleElementSet"
        )

        print(f"Set exists before removal: {results['exists_before']}")
        print(f"Set exists after removal: {results['exists_after']}")

        if not results["exists_after"]:
            print("The set attribute was completely removed when the last element
was deleted.")
        else:
            print("The set attribute still exists after removing the last
element.")
    except Exception as e:
        print(f"Error in last element removal demonstration: {e}")

    print("\nExample 5: Working with a number set")
    try:
        results = work_with_number_set(
            table_name=table_name,
            key={"UserId": "user456"},
            set_name="LuckyNumbers",
            initial_values=[7, 13, 42],
            values_to_add=[99, 100],
            values_to_remove=[13],
        )

        print(f"Initial number set: {results['create_response'].get('Attributes',
{})}")
```

```
        print(f"After adding numbers: {results['add_response'].get('Attributes',
{})}")
        print(f"After removing numbers:
{results['remove_response'].get('Attributes', {})}")
        print(f"Final state: {results['final_state']}")
    except Exception as e:
        print(f"Error working with number set: {e}")

    print("\nKey Points About DynamoDB Sets:")
    print("1. Sets can only contain elements of the same type (string, number, or
binary)")
    print("2. Sets automatically eliminate duplicate values")
    print("3. The ADD operation creates a set if it doesn't exist")
    print("4. The DELETE operation removes specified elements from a set")
    print("5. When the last element is removed from a set, the entire attribute
is deleted")
    print("6. Empty sets are not allowed in DynamoDB")
    print("7. Sets are unordered collections")
    print("8. The ADD operation is atomic for sets")
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB usando lotes de instruções PartiQL e um AWS SDK

Os exemplos de código a seguir mostram como:

- Obter um lote de itens executando várias instruções SELECT.
- Adicionar um lote de itens executando várias instruções INSERT.
- Atualizar um lote de itens executando várias instruções UPDATE.
- Excluir um lote de itens executando várias instruções DELETE.

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = @"..\..\..\..\..\..\..\..\resources\sample_files
\movies.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
```

```
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1,
 year1, producer2, title2, year2);
if (success)
{
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
```

```
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
```

```
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT
 statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch
 method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting
 the table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}
```

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
```

```
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
```

```
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
```

```
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
```

```
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }

        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
```

```
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
```

```
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlBatchExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using PartiQL batch statements.
/*!
  \sa partiqlBatchExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::partiqlBatchExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {

    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
```

```
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
```

```
            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the movie data with a batch \"SELECT\" statement."
              << std::endl;

    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
```

```
                << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

```
   // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

   for (size_t i = 0; i < titles.size(); ++i) {
       ratings[i] = askQuestionForFloatRange(
               Aws::String("\nLet's update your the movie, \"") + titles[i] +
               ".\nYou rated it  " + std::to_string(ratings[i])
               + ", what new rating would you give it? ", 1, 10);
   }

   std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());

       std::stringstream sqlStream;
       sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);

           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);
       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
```

```
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to update movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
   }

   std::cout << "Retrieving the updated movie data with a batch \"SELECT\"
statement."
             << std::endl;

   // 5. Get the updated data for multiple movies using "Select" statements.
(BatchExecuteStatement)
   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);
           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);

       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (outcome.IsSuccess()) {
           const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();
```

```
            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movies information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Deleting the movie data with a batch \"DELETE\" statement."
              << std::endl;

    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;
```

```
        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
```

```
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
```

```
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
```

```
                                        const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute um cenário que crie uma tabela e execute lotes de consultas do PartiQL.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
```

```
}()

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}
runner := actions.PartiQLRunner{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovies := []actions.Movie{{
 Title: "House PartiQL",
 Year:  currentYear - 5,
 Info: map[string]interface{}{
  "plot":   "Wacky high jinks result from querying a mysterious database.",
  "rating": 8.5}}, {
 Title: "House PartiQL 2",
 Year:  currentYear - 3,
 Info: map[string]interface{}{
  "plot":   "Moderate high jinks result from querying another mysterious
database.",
```

```
  "rating": 6.5}}, {
 Title: "House PartiQL 3",
 Year:  currentYear - 1,
 Info: map[string]interface{}{
  "plot":   "Tepid high jinks result from querying yet another mysterious
database.",
  "rating": 2.5},
},
}

log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(ctx, customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(ctx, customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))

newRatings := []float64{7.7, 4.4, 1.1}
log.Println("Updating a batch of movies with new ratings.")
err = runner.UpdateMovieBatch(ctx, customMovies, newRatings)
if err == nil {
 log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting projected data from the table to verify our update.")
log.Println("Using a page size of 2 to demonstrate paging.")
projections, err := runner.GetAllMovies(ctx, 2)
if err == nil {
 log.Println("All movies:")
 for _, projection := range projections {
  log.Println(projection)
 }
}
log.Println(strings.Repeat("-", 88))
```

```
 log.Println("Deleting a batch of movies.")
 err = runner.DeleteMovieBatch(ctx, customMovies)
 if err == nil {
  log.Printf("Deleted %v movies.\n", len(customMovies))
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
```

```
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Crie uma estrutura e métodos que executem declarações PartiQL.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
```

```
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
```

```
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
```

```
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}



// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
```

```
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }
```

```
 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
public class ScenarioPartiQLBatch {
    public static void main(String[] args) throws IOException {
        String tableName = "MoviesPartiQBatch";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println("Creating an Amazon DynamoDB table named " + tableName
            + " with a key named year and a sort key named title.");
        createTable(ddb, tableName);

        System.out.println("Adding multiple records into the " + tableName
            + " table using a batch command.");
        putRecordBatch(ddb);
```

```
        // Update multiple movies by using the BatchExecute statement.
        String title1 = "Star Wars";
        int year1 = 1977;
        String title2 = "Wizard of Oz";
        int year2 = 1939;

        System.out.println("Query two movies.");
        getBatch(ddb, tableName, title1, title2, year1, year2);

        System.out.println("Updating multiple records using a batch command.");
        updateTableItemBatch(ddb);

        System.out.println("Deleting multiple records using a batch command.");
        deleteItemBatch(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static boolean getBatch(DynamoDbClient ddb, String tableName, String
title1, String title2, int year1, int year2) {
        String getBatch = "SELECT * FROM " + tableName + " WHERE title = ? AND
year = ?";

        List<BatchStatementRequest> statements = new ArrayList<>();
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title1).build(),
                AttributeValue.builder().n(String.valueOf(year1)).build())
            .build());
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title2).build(),
                AttributeValue.builder().n(String.valueOf(year2)).build())
            .build());

        BatchExecuteStatementRequest batchExecuteStatementRequest =
BatchExecuteStatementRequest.builder()
            .statements(statements)
            .build();

        try {
```

```
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchExecuteStatementRequest);
            if (!response.responses().isEmpty()) {
                response.responses().forEach(r -> {
                    System.out.println(r.item().get("title") + "\\t" +
r.item().get("year"));
                });
                return true;
            } else {
                System.out.println("Couldn't find either " + title1 + " or " +
title2 + ".");
                return false;
            }
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            return false;
        }
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();
```

```
        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse = dbWaiter
                .waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecordBatch(DynamoDbClient ddb) {
        String sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            // Create three movies to add to the Amazon DynamoDB table.
            // Set data for Movie 1.
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n("1977")
                .build();
```

```
            AttributeValue att2 = AttributeValue.builder()
                .s("Star Wars")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            BatchStatementRequest statementRequestMovie1 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parameters)
                .build();

            // Set data for Movie 2.
            List<AttributeValue> parametersMovie2 = new ArrayList<>();
            AttributeValue attMovie2 = AttributeValue.builder()
                .n("1939")
                .build();

            AttributeValue attMovie2A = AttributeValue.builder()
                .s("Wizard of Oz")
                .build();

            AttributeValue attMovie2B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie2.add(attMovie2);
            parametersMovie2.add(attMovie2A);
            parametersMovie2.add(attMovie2B);

            BatchStatementRequest statementRequestMovie2 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie2)
                .build();

            // Set data for Movie 3.
            List<AttributeValue> parametersMovie3 = new ArrayList<>();
```

```
            AttributeValue attMovie3 = AttributeValue.builder()
                .n(String.valueOf("2022"))
                .build();

            AttributeValue attMovie3A = AttributeValue.builder()
                .s("My Movie 3")
                .build();

            AttributeValue attMovie3B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie3.add(attMovie3);
            parametersMovie3.add(attMovie3A);
            parametersMovie3.add(attMovie3B);

            BatchStatementRequest statementRequestMovie3 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie3)
                .build();

            // Add all three movies to the list.
            List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
            myBatchStatementList.add(statementRequestMovie1);
            myBatchStatementList.add(statementRequestMovie2);
            myBatchStatementList.add(statementRequestMovie3);

            BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
                .statements(myBatchStatementList)
                .build();

            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Added new movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
    public static void updateTableItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "UPDATE MoviesPartiQBatch SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Update three records.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Update record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Update record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
```

```
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);
        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Updated three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    public static void deleteItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and
title=?";
```

```
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Specify three records to delete.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Specify record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Specify record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();
```

```
        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);

        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            ddb.batchExecuteStatement(batchRequest);
            System.out.println("Deleted three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,

 List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
            .build();

        return ddb.executeStatement(request);
    }
}
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute instruções PartiQL em lote.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
```

```
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({}, { confirmAll });
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
```

```
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");
```

```
  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log("Cities inserted.");

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
```

```
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log("Updated cities.");

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
```

```
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQBatch"
    println("Creating an Amazon DynamoDB table named $tableName with a key named
 id and a sort key named title.")
    createTablePartiQLBatch(ddb, tableName, "year")
    putRecordBatch(ddb)
    updateTableItemBatchBatch(ddb)
    deleteItemsBatch(ddb)
    deleteTablePartiQLBatch(tableName)
}
```

```
suspend fun createTablePartiQLBatch(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
```

```
}

suspend fun putRecordBatch(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
 'title' : ?, 'info' : ?}"

    // Create three movies to add to the Amazon DynamoDB table.
    val parametersMovie1 = mutableListOf<AttributeValue>()
    parametersMovie1.add(AttributeValue.N("2022"))
    parametersMovie1.add(AttributeValue.S("My Movie 1"))
    parametersMovie1.add(AttributeValue.S("No Information"))

    val statementRequestMovie1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie1
        }

    // Set data for Movie 2.
    val parametersMovie2 = mutableListOf<AttributeValue>()
    parametersMovie2.add(AttributeValue.N("2022"))
    parametersMovie2.add(AttributeValue.S("My Movie 2"))
    parametersMovie2.add(AttributeValue.S("No Information"))

    val statementRequestMovie2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie2
        }

    // Set data for Movie 3.
    val parametersMovie3 = mutableListOf<AttributeValue>()
    parametersMovie3.add(AttributeValue.N("2022"))
    parametersMovie3.add(AttributeValue.S("My Movie 3"))
    parametersMovie3.add(AttributeValue.S("No Information"))

    val statementRequestMovie3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
```

```
    myBatchStatementList.add(statementRequestMovie1)
    myBatchStatementList.add(statementRequestMovie2)
    myBatchStatementList.add(statementRequestMovie3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }
    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: " + response.toString())
    println("Added new movies using a batch command.")
}

suspend fun updateTableItemBatchBatch(ddb: DynamoDbClient) {
    val sqlStatement =
        "UPDATE MoviesPartiQBatch SET info = 'directors\":[\"Merian C. Cooper\",
\"Ernest B. Schoedsack' where year=? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))
    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Update record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Update record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
```

```
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: $response")
    println("Updated three movies using a batch command.")
    println("Items were updated!")
}

suspend fun deleteItemsBatch(ddb: DynamoDbClient) {
    // Specify three records to delete.
    val sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))

    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Specify record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Specify record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
```

```
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    ddb.batchExecuteStatement(batchRequest)
    println("Deleted three movies using a batch command.")
}

suspend fun deleteTablePartiQLBatch(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithPartiQLBatch
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
```

```
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQLBatch($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
```

```
        $service->updateItemByPartiQLBatch($statement, $parameters);
        echo "Movie added and updated.\n";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "\nThe movie {$movie['Responses'][0]['Item']['title']['S']}
        was released in {$movie['Responses'][0]['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Responses'][0]['Item']
['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "Okay, you have rated {$movie['Responses'][0]['Item']['title']
['S']}
        as a {$movie['Responses'][0]['Item']['rating']['N']}\n";

        $service->deleteItemByPartiQLBatch($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
```

```
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
```

```
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
```

```
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para PHP.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma classe que possa executar lotes de instruções PartiQL.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
```

```
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
the
        resource transforms input and output from plain old Python objects
(POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
with
                           each statement. This list must be in the same order as
the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
```

```
            raise
        else:
            return output
```

Execute um cenário que crie uma tabela e execute consultas do PartiQL em lotes.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL batch statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    movie_data = [
        {
            "title": f"House PartiQL",
            "year": datetime.now().year - 5,
            "info": {
                "plot": "Wacky high jinks result from querying a mysterious
 database.",
                "rating": Decimal("8.5"),
            },
        },
        {
            "title": f"House PartiQL 2",
            "year": datetime.now().year - 3,
            "info": {
                "plot": "Moderate high jinks result from querying another
 mysterious database.",
                "rating": Decimal("6.5"),
            },
        },
        {
            "title": f"House PartiQL 3",
            "year": datetime.now().year - 1,
```

```
            "info": {
                "plot": "Tepid high jinks result from querying yet another
mysterious database.",
                "rating": Decimal("2.5"),
            },
        },
    ]

    print(f"Inserting a batch of movies into table '{table_name}.")
    statements = [
        f'INSERT INTO "{table_name}" ' f"VALUE {{'title': ?, 'year': ?,
'info': ?}}"
    ] * len(movie_data)
    params = [list(movie.values()) for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting data for a batch of movies.")
    statements = [f'SELECT * FROM "{table_name}" WHERE title=? AND year=?'] *
len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    output = wrapper.run_partiql(statements, params)
    for item in output["Responses"]:
        print(f"\n{item['Item']['title']}, {item['Item']['year']}")
        pprint(item["Item"])
    print("-" * 88)

    ratings = [Decimal("7.7"), Decimal("5.5"), Decimal("1.3")]
    print(f"Updating a batch of movies with new ratings.")
    statements = [
        f'UPDATE "{table_name}" SET info.rating=? ' f"WHERE title=? AND year=?"
    ] * len(movie_data)
    params = [
        [rating, movie["title"], movie["year"]]
        for rating, movie in zip(ratings, movie_data)
    ]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting projected data from the table to verify our update.")
```

```
    output = wrapper.dyn_resource.meta.client.execute_statement(
        Statement=f'SELECT title, info.rating FROM "{table_name}"'
    )
    pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting a batch of movies from the table.")
    statements = [f'DELETE FROM "{table_name}" WHERE title=? AND year=?'] * len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLBatchWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute um cenário que crie uma tabela e execute consultas do PartiQL em lotes.

```
 table_name = "doc-example-table-movies-partiql-#{rand(10**4)}"
 scaffold = Scaffold.new(table_name)
 sdk = DynamoDBPartiQLBatch.new(table_name)

 new_step(1, 'Create a new DynamoDB table if none already exists.')
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Populate DynamoDB table with movie data.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
 print "Done!\n".green

 new_step(3, 'Select a batch of items from the movies table.')
 puts "Let's select some popular movies for side-by-side comparison."
 response = sdk.batch_execute_select([['Mean Girls', 2004], ['Goodfellas',
1977], ['The Prancing of the Lambs', 2005]])
 puts("Items selected: #{response['responses'].length}\n")
 print "\nDone!\n".green

 new_step(4, 'Delete a batch of items from the movies table.')
 sdk.batch_execute_write([['Mean Girls', 2004], ['Goodfellas', 1977], ['The
Prancing of the Lambs', 2005]])
 print "\nDone!\n".green
```

```
  new_step(5, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Para obter detalhes da API, consulte BatchExecuteStatement na Referência da API do AWS SDK para Ruby.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB usando o PartiQL e um AWS SDK

Os exemplos de código a seguir mostram como:

- Obter um item executando uma instrução SELECT.
- Adicionar um item executando uma instrução INSERT.
- Atualizar um item executando a instrução UPDATE.
- Excluir um item executando uma instrução DELETE.

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
```

```
        private static readonly AmazonDynamoDBClient Client = new
AmazonDynamoDBClient();


        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
```

```
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }
```

```
            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }


        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string,
AttributeValue>> items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
```

```
                items.ForEach(item =>
 Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
```

```
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using single PartiQL
 statements.
/*!
  \sa partiqlExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::partiqlExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
```

```
   Aws::String title;
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                         1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                 << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                 << INFO_KEY << "': ?}";

       request.SetStatement(sqlStream.str());

       // Create the parameter attributes.
       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
       attributes.push_back(infoMapAttribute);
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
```

```
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();
```

```
            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }

    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
```

```
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    //  5. Get the updated data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
```

```
    }

    std::cout << "Deleting the movie" << std::endl;

    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Movie successfully deleted." << std::endl;
    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;
```

```
    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
```

```
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
```

```
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
```

```
        count++;
    }
    return false;
}
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute um cenário que crie uma tabela e execute consultas do PartiQL.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
```

```
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(ctx context.Context, sdkConfig aws.Config,
 tableName string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
 log.Println(strings.Repeat("-", 88))

 currentYear, _, _ := time.Now().Date()
 customMovie := actions.Movie{
  Title: "24 Hour PartiQL People",
```

```
 Year:  currentYear,
 Info: map[string]interface{}{
  "plot":   "A group of data developers discover a new query language they can't
stop using.",
  "rating": 9.9,
 },
}

log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
err = runner.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
movie, err := runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

newRating := 6.6
log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
newRating)
err = runner.UpdateMovie(ctx, customMovie, newRating)
if err == nil {
 log.Printf("Updated %v with a new rating.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data again to verify the update.")
movie, err = runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Deleting movie '%v'.\n", customMovie.Title)
err = runner.DeleteMovie(ctx, customMovie)
if err == nil {
 log.Printf("Deleted %v.\n", customMovie.Title)
```

```
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Defina uma estrutura de filme usada neste exemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
```

```
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Crie uma estrutura e métodos que executem declarações PartiQL.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
```

```
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
```

```
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
public class ScenarioPartiQ {
    public static void main(String[] args) throws IOException {
        String fileName = "../../../resources/sample_files/movies.json";
        String tableName = "MoviesPartiQ";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(
            "******* Creating an Amazon DynamoDB table named MoviesPartiQ with a
 key named year and a sort key named title.");
```

```
        createTable(ddb, tableName);

        System.out.println("Loading data into the MoviesPartiQ table.");
        loadData(ddb, fileName);

        System.out.println("Getting data from the MoviesPartiQ table.");
        getItem(ddb);

        System.out.println("Putting a record into the MoviesPartiQ table.");
        putRecord(ddb);

        System.out.println("Updating a record.");
        updateTableItem(ddb);

        System.out.println("Querying the movies released in 2013.");
        queryTable(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
```

```
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //Scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String fileName) throws
IOException {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
```

```
        ObjectNode currentNode;
        int t = 0;
        List<AttributeValue> parameters = new ArrayList<>();
        while (iter.hasNext()) {

            // Add 200 movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf(year))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s(title)
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s(info)
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            // Insert the movie into the Amazon DynamoDB table.
            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added Movie " + title);

            parameters.remove(att1);
            parameters.remove(att2);
            parameters.remove(att3);
            t++;
        }
    }

    public static void getItem(DynamoDbClient ddb) {
```

```
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and
title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n("2012")
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The Perks of Being a Wallflower")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecord(DynamoDbClient ddb) {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2020"))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s("My Movie")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();
```

```
            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added new movie.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItem(DynamoDbClient ddb) {

        String sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2013"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The East")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            executeStatementRequest(ddb, sqlStatement, parameters);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    // Query the table where the year is 2013.
    public static void queryTable(DynamoDbClient ddb) {
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year = ? ORDER BY
 year";
```

```
        try {

            List<AttributeValue> parameters = new ArrayList<>();
            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2013"))
                .build();
            parameters.add(att1);

            // Get items in the table and write out the ID value.
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{

        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
executeStatementRequest(DynamoDbClient ddb, String statement,

List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
```

```
            .build();

        return ddb.executeStatement(request);
    }

    private static void processResults(ExecuteStatementResponse
 executeStatementResult) {
        System.out.println("ExecuteStatement successful: " +
 executeStatementResult.toString());
    }
}
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute instruções PartiQL individuais.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */
```

```
  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log("Coffee inserted.");

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log("Updated coffee");

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQ"
    val fileName = "../../../resources/sample_files/movies.json"
    println("Creating an Amazon DynamoDB table named MoviesPartiQ with a key
 named id and a sort key named title.")
    createTablePartiQL(ddb, tableName, "year")
    loadDataPartiQL(ddb, fileName)
```

```
    println("******* Getting data from the MoviesPartiQ table.")
    getMoviePartiQL(ddb)

    println("******* Putting a record into the MoviesPartiQ table.")
    putRecordPartiQL(ddb)

    println("******* Updating a record.")
    updateTableItemPartiQL(ddb)

    println("******* Querying the movies released in 2013.")
    queryTablePartiQL(ddb)

    println("******* Deleting the MoviesPartiQ table.")
    deleteTablePartiQL(tableName)
}

suspend fun createTablePartiQL(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }
```

```
    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun loadDataPartiQL(
    ddb: DynamoDbClient,
    fileName: String,
) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
    var t = 0

    while (iter.hasNext()) {
        if (t == 200) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
        parameters.add(AttributeValue.N(year.toString()))
        parameters.add(AttributeValue.S(title))
        parameters.add(AttributeValue.S(info))
```

```
        executeStatementPartiQL(ddb, sqlStatement, parameters)
        println("Added Movie $title")
        parameters.clear()
        t++
    }
}

suspend fun getMoviePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and title=?"
    val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
    parameters.add(AttributeValue.N("2012"))
    parameters.add(AttributeValue.S("The Perks of Being a Wallflower"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun putRecordPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2020"))
    parameters.add(AttributeValue.S("My Movie"))
    parameters.add(AttributeValue.S("No Info"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Added new movie.")
}

suspend fun updateTableItemPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":[\"Merian C.
 Cooper\",\"Ernest B. Schoedsack\' where year=? and title=?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    parameters.add(AttributeValue.S("The East"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Item was updated!")
}

// Query the table where the year is 2013.
suspend fun queryTablePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year = ?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
```

```
    println("ExecuteStatement successful: $response")
}

suspend fun deleteTablePartiQL(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun executeStatementPartiQL(
    ddb: DynamoDbClient,
    statementVal: String,
    parametersVal: List<AttributeValue>,
): ExecuteStatementResponse {
    val request =
        ExecuteStatementRequest {
            statement = statementVal
            parameters = parametersVal
        }

    return ddb.executeStatement(request)
}
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\testable_readline;
use function AwsUtilities\loadMovieData;

class GettingStartedWithPartiQL
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
```

```
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQL($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);
        echo "Movie added and updated.\n";



        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQL($tableName, $key);
```

```
        echo "\nThe movie {$movie['Items'][0]['title']['S']} was released in
 {$movie['Items'][0]['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Items'][0]['title']
['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "Okay, you have rated {$movie['Items'][0]['title']['S']} as a
 {$movie['Items'][0]['rating']['N']}\n";

        $service->deleteItemByPartiQL($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
```

```
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
```

```
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para PHP.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie uma classe que possa executar instruções PartiQL.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint
```

```
import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
```

```
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

Execute um cenário que crie uma tabela e execute consultas do PartiQL.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL single statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    title = "24 Hour PartiQL People"
    year = datetime.now().year
    plot = "A group of data developers discover a new query language they can't
 stop using."
    rating = Decimal("9.9")

    print(f"Inserting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f"INSERT INTO \"{table_name}\" VALUE {{'title': ?, 'year': ?,
 'info': ?}}",
        [title, year, {"plot": plot, "rating": rating}],
    )
    print("Success!")
    print("-" * 88)
```

```
    print(f"Getting data for movie '{title}' released in {year}.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    rating = Decimal("2.4")
    print(f"Updating movie '{title}' with a rating of {float(rating)}.")
    wrapper.run_partiql(
        f'UPDATE "{table_name}" SET info.rating=? WHERE title=? AND year=?',
        [rating, title, year],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data again to verify our update.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f'DELETE FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
```

```
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute um cenário que crie uma tabela e execute consultas do PartiQL.

```
table_name = "doc-example-table-movies-partiql-#{rand(10**8)}"
scaffold = Scaffold.new(table_name)
sdk = DynamoDBPartiQLSingle.new(table_name)

new_step(1, 'Create a new DynamoDB table if none already exists.')
unless scaffold.exists?(table_name)
  puts("\nNo such table: #{table_name}. Creating it...")
  scaffold.create_table(table_name)
  print "Done!\n".green
end

new_step(2, 'Populate DynamoDB table with movie data.')
download_file = 'moviedata.json'
puts("Downloading movie database to #{download_file}...")
movie_data = scaffold.fetch_movie_data(download_file)
puts("Writing movie data from #{download_file} into your table...")
scaffold.write_batch(movie_data)
puts("Records added: #{movie_data.length}.")
print "Done!\n".green
```

```
  new_step(3, 'Select a single item from the movies table.')
  response = sdk.select_item_by_title('Star Wars')
  puts("Items selected for title 'Star Wars': #{response.items.length}\n")
  print response.items.first.to_s.yellow
  print "\n\nDone!\n".green

  new_step(4, 'Update a single item from the movies table.')
  puts "Let's correct the rating on The Big Lebowski to 10.0."
  sdk.update_rating_by_title('The Big Lebowski', 1998, 10.0)
  print "\nDone!\n".green

  new_step(5, 'Delete a single item from the movies table.')
  puts "Let's delete The Silence of the Lambs because it's just too scary."
  sdk.delete_item_by_title('The Silence of the Lambs', 1991)
  print "\nDone!\n".green

  new_step(6, 'Insert a new item into the movies table.')
  puts "Let's create a less-scary movie called The Prancing of the Lambs."
  sdk.insert_item('The Prancing of the Lambs', 2005, 'A movie about happy
 livestock.', 5.0)
  print "\nDone!\n".green

  new_step(7, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência da API AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

```
async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

    match client
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn add_item(client: &Client, item: Item) -> Result<(),
 SdkError<ExecuteStatementError>> {
    match client
        .execute_statement()
        .statement(format!(
            r#"INSERT INTO "{}" VALUE {{
                "{}": ?,
                "acount_type": ?,
                "age": ?,
                "first_name": ?,
                "last_name": ?
        }} "#,
```

```
            item.table, item.key
        ))
        .set_parameters(Some(vec![
            AttributeValue::S(item.utype),
            AttributeValue::S(item.age),
            AttributeValue::S(item.first_name),
            AttributeValue::S(item.last_name),
        ]))
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}
```

```
async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
 Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
}
```

- Para obter detalhes da API, consulte ExecuteStatement na Referência de API do AWS SDK para Rust.

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB usando um índice secundário global por meio de um SDK da AWS

O código de exemplo a seguir mostra como consultar uma tabela usando um índice secundário global.

- Consulte uma tabela do DynamoDB usando a respectiva chave primária.
- Consulte um índice secundário global (GSI) para obter padrões de acesso alternativos.
- Compare consultas de tabela e consultas de GSI.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB usando a respectiva chave primária e um índice secundário global (GSI) por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();
```

```
        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

Compare a consulta direta a uma tabela com a consulta a um GSI por meio do AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <basePartitionKeyName> <basePartitionKeyValue>
<gsiName> <gsiPartitionKeyName> <gsiPartitionKeyValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
```

```
                    basePartitionKeyName - The name of the base table partition
key attribute.
                    basePartitionKeyValue - The value of the base table partition
key to query.
                    gsiName - The name of the Global Secondary Index.
                    gsiPartitionKeyName - The name of the GSI partition key
attribute.
                    gsiPartitionKeyValue - The value of the GSI partition key to
query.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """";

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String basePartitionKeyName = args[1];
        final String basePartitionKeyValue = args[2];
        final String gsiName = args[3];
        final String gsiPartitionKeyName = args[4];
        final String gsiPartitionKeyValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        try (DynamoDbClient ddb =
DynamoDbClient.builder().region(region).build()) {
            final QueryTableAndGSI queryHelper = new QueryTableAndGSI(ddb);

            // Query the base table
            System.out.println("Querying base table where " +
basePartitionKeyName + " = " + basePartitionKeyValue);
            final QueryResponse tableResponse =
                queryHelper.queryTable(tableName, basePartitionKeyName,
basePartitionKeyValue);

            System.out.println("Found " + tableResponse.count() + " items in base
table:");
            tableResponse.items().forEach(item -> System.out.println(item));

            // Query the GSI
            System.out.println(
```

```
                "\nQuerying GSI '" + gsiName + "' where " + gsiPartitionKeyName +
" = " + gsiPartitionKeyValue);
            final QueryResponse gsiResponse =
                queryHelper.queryGlobalSecondaryIndex(tableName, gsiName,
gsiPartitionKeyName, gsiPartitionKeyValue);

            System.out.println("Found " + gsiResponse.count() + " items in
GSI:");
            gsiResponse.items().forEach(item -> System.out.println(item));

            // Explain the differences between querying a table and a GSI
            System.out.println("\nKey differences between querying a table and a
GSI:");
            System.out.println("1. When querying a GSI, you must specify the
indexName parameter");
            System.out.println("2. GSIs may not contain all attributes from the
base table (projection)");
            System.out.println("3. GSIs consume read capacity units from the
GSI's capacity, not the base table's");
            System.out.println("4. GSIs may have eventually consistent data
(cannot use ConsistentRead=true)");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table or index not found: " + e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte uma tabela do DynamoDB usando a chave primária por meio do AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}
```

Consulte um Índice Secundário Global (GSI) do DynamoDB por meio do AWS SDK para JavaScript.

```
/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB usando a respectiva chave primária e um índice secundário global (GSI) por meio do AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB usando uma condição begins_with por meio de um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela usando uma condição begins_with.

- Use a função begins_with em uma expressão de condição de chave.
- Filtre itens com base em um padrão de prefixo na chave de classificação.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB usando uma condição begins_with na chave de classificação por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithBeginsWithCondition(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String sortKeyName,
        final String sortKeyPrefix) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Sort key name", sortKeyName);
        CodeSampleUtils.validateStringParameter("Sort key prefix",
 sortKeyPrefix);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, sortKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_SK_PREFIX,
            AttributeValue.builder().s(sortKeyPrefix).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with begins_with condition successful.
Found {0} items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with begins_with condition",
e);
            throw e;
        }
    }
```

Demonstre o uso de begins_with com diferentes comprimentos de prefixo por meio do AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        try {
            CodeSampleUtils.BeginsWithQueryConfig config =
CodeSampleUtils.BeginsWithQueryConfig.fromArgs(args);
            LOGGER.log(Level.INFO, "Querying items where {0} = {1} and {2} begins
with ''{3}''", new Object[] {
                config.getPartitionKeyName(),
                config.getPartitionKeyValue(),
                config.getSortKeyName(),
                config.getSortKeyPrefix()
            });
```

```
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new BeginsWithQueryBuilder()
                .withTableName(config.getTableName())
                .withPartitionKeyName(config.getPartitionKeyName())
                .withPartitionKeyValue(config.getPartitionKeyValue())
                .withSortKeyName(config.getSortKeyName())
                .withSortKeyPrefix(config.getSortKeyPrefix())
                .withRegion(config.getRegion())
                .execute();

            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> LOGGER.info(item.toString()));

            // Demonstrate with a different prefix
            if (!config.getSortKeyPrefix().isEmpty()) {
                String shorterPrefix = config.getSortKeyPrefix()
                    .substring(0, Math.max(1,
config.getSortKeyPrefix().length() / 2));
                LOGGER.log(Level.INFO, "\nNow querying with a shorter prefix:
''{0}''", shorterPrefix);

                final QueryResponse response2 = new BeginsWithQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withSortKeyName(config.getSortKeyName())
                    .withSortKeyPrefix(shorterPrefix)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with shorter prefix:",
response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }
        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            printUsage();
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found", e);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error", e);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error", e);
```

```
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte uma tabela do DynamoDB usando uma condição begins_with na chave de classificação por meio do AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
```

```
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB usando uma condição begins_with na chave de classificação por meio do AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_begins_with(
    table_name, partition_key_name, partition_key_value, sort_key_name, prefix
):
    """
    Query a DynamoDB table with a begins_with condition on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute.
        prefix (str): The prefix to match at the beginning of the sort key.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query with a begins_with condition on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) & Key(
        sort_key_name
    ).begins_with(prefix)
    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consulte uma tabela do DynamoDB usando um intervalo de datas na chave de classificação por meio de um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela usando um intervalo de datas na chave de classificação.

- Consulte itens em um intervalo de datas específico.
- Use operadores de comparação em chaves de classificação formatadas por data.

Java

SDK para Java 2.x

Consulte itens em uma tabela do DynamoDB em um intervalo de datas por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();
```

```
        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

Demonstra como consultar uma tabela do DynamoDB com filtragem de intervalo de datas.

```
   public static void main(String[] args) {
       final String usage =
           """
               Usage:
                   <tableName> <partitionKeyName> <partitionKeyValue>
<dateKeyName> <startDate> <endDate> [region]
               Where:
                   tableName - The Amazon DynamoDB table to query.
                   partitionKeyName - The name of the partition key attribute.
                   partitionKeyValue - The value of the partition key to query.
                   dateKeyName - The name of the date attribute to filter on.
                   startDate - The start date for the range query (YYYY-MM-DD).
                   endDate - The end date for the range query (YYYY-MM-DD).
                   region (optional) - The AWS region where the table exists.
(Default: us-east-1)
               """;

       if (args.length < 6) {
           System.out.println(usage);
           System.exit(1);
       }

       try {
           // Parse command line arguments into a config object
           CodeSampleUtils.DateRangeQueryConfig config =
CodeSampleUtils.DateRangeQueryConfig.fromArgs(args);

           LOGGER.log(
               Level.INFO, "Querying items from {0} to {1}", new Object[]
{config.getStartDate(), config.getEndDate()
               });

           // Using the builder pattern to create and execute the query
           final QueryResponse response = new DateRangeQueryBuilder()
               .withTableName(config.getTableName())
               .withPartitionKeyName(config.getPartitionKeyName())
               .withPartitionKeyValue(config.getPartitionKeyValue())
               .withDateKeyName(config.getDateKeyName())
               .withStartDate(config.getStartDate())
               .withEndDate(config.getEndDate())
               .withRegion(config.getRegion())
               .execute();
```

```
            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> {
                LOGGER.info(item.toString());

                // Extract and display the date attribute for clarity
                if (item.containsKey(config.getDateKeyName())) {
                    LOGGER.log(
                        Level.INFO,
                        "  Date attribute: {0}",
                        item.get(config.getDateKeyName()).s());
                }
            });

            // Demonstrate with a different date range
            LocalDate narrowerStartDate = config.getStartDate().plusDays(1);
            LocalDate narrowerEndDate = config.getEndDate().minusDays(1);

            if (!narrowerStartDate.isAfter(narrowerEndDate)) {
                LOGGER.log(Level.INFO, "\nNow querying with a narrower date
range: {0} to {1}", new Object[] {
                    narrowerStartDate, narrowerEndDate
                });

                final QueryResponse response2 = new DateRangeQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withDateKeyName(config.getDateKeyName())
                    .withStartDate(narrowerStartDate)
                    .withEndDate(narrowerEndDate)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with narrower date
range:", response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }

            LOGGER.info("\nNote: When storing dates in DynamoDB:");
            LOGGER.info("1. Use ISO format (YYYY-MM-DD) for lexicographical
ordering");
```

```
            LOGGER.info("2. Use the BETWEEN operator for inclusive date range
 queries");
            LOGGER.info("3. Consider using ISO-8601 format for timestamps with
 time components");

        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error: {0}", e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error: {0}", e.getMessage());
            System.exit(1);
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte itens em uma tabela do DynamoDB em um intervalo de datas por meio do AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
```

```
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte itens em uma tabela do DynamoDB em um intervalo de datas por meio do AWS SDK para Python (Boto3).

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
```

```
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB com uma expressão de filtro complexa por meio de um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela com uma expressão de filtro complexa.

- Aplique expressões de filtro complexas aos resultados da consulta.
- Combine várias condições usando operadores lógicos.
- Filtre itens com base em atributos não chave.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB com uma expressão de filtro complexa por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
```

```
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
```

```
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte uma tabela do DynamoDB com uma expressão de filtro complexa por meio do AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);
```

```
    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB com uma expressão de filtro complexa por meio do AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
```

```
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
```

```
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition
```

```
    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB com uma expressão de filtro dinâmica por meio de um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela com uma expressão de filtro dinâmica.

- Crie expressões de filtro dinamicamente no tempo de execução.
- Crie condições de filtro com base na entrada do usuário ou no estado da aplicação.
- Adicione ou remova critérios de filtro condicionalmente.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB com uma expressão de filtro criada dinamicamente por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
 QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
 partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
```

```
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
e.getMessage());
            throw e;
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }
```

Demonstre como usar expressões de filtro dinâmico com o AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
```

```
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
```

```
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte uma tabela do DynamoDB com uma expressão de filtro criada dinamicamente por meio do AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
```

```
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
```

```
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB com uma expressão de filtro criada dinamicamente por meio do AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
 {'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
```

```
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}

    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition
```

```
    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Demonstre como usar expressões de filtro dinâmico com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use the query_with_dynamic_filter function."""
    # Example parameters
    table_name = "Products"
    partition_key_name = "Category"
    partition_key_value = "Electronics"

    # Define dynamic filter conditions based on user input or runtime conditions
    user_min_rating = 4  # This could come from user input
    user_status_filter = "active"  # This could come from user input

    filter_conditions = {}

    # Only add conditions that are actually specified
    if user_min_rating is not None:
        filter_conditions["rating"] = {"operator": ">=", "value":
 user_min_rating}

    if user_status_filter:
        filter_conditions["status"] = {"operator": "=", "value":
 user_status_filter}

    print(
        f"Querying products in category '{partition_key_value}' with filter
 conditions: {filter_conditions}"
    )

    # Execute the query with dynamic filter
    response = query_with_dynamic_filter(
```

```
        table_name, partition_key_name, partition_key_value, filter_conditions
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Product: {item}")
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB com uma expressão de filtro e limite por meio de um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela com uma expressão de filtro e limite.

- Aplique expressões de filtro aos resultados da consulta com um limite nos itens avaliados.
- Entenda como o limite afeta os resultados da consulta filtrada.
- Controle o número máximo de itens processados em uma consulta.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB com uma expressão de filtro e limite por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
 filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
 filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
 filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());
```

```
        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB com uma expressão de filtro e limite por meio do AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key
```

```
def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}

    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
 Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
 filter_value}

    # Execute the query
```

```
    response = table.query(**query_params)
    return response
```

Demonstra como usar expressões de filtro com limites no AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use the query_with_filter_and_limit function."""
    # Example parameters
    table_name = "ProductReviews"
    partition_key_name = "ProductId"
    partition_key_value = "P123456"
    filter_attribute = "Rating"
    filter_value = 3  # Filter for ratings > 3
    limit = 5

    print(f"Querying reviews for product '{partition_key_value}' with rating >
 {filter_value}")
    print(f"Limiting to {limit} evaluated items")

    # Execute the query with filter and limit
    response = query_with_filter_and_limit(
        table_name, partition_key_name, partition_key_value, filter_attribute,
 filter_value, limit
    )

    # Process the results
    items = response.get("Items", [])
    print(f"\nReturned {len(items)} items that passed the filter")

    for item in items:
        print(f"Review: {item}")

    # Explain the difference between Limit and actual results
    explain_limit_vs_results(response)

    # Check if there are more results
    if "LastEvaluatedKey" in response:
        print("\nThere are more results available. Use the LastEvaluatedKey for
 pagination.")
    else:
```

```
        print("\nAll matching results have been retrieved.")
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB com atributos aninhados por meio de um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela com atributos aninhados.

- Acesse e filtre por atributos aninhados nos itens do DynamoDB.
- Use expressões de caminho de documento para se referir a elementos aninhados.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB com atributos aninhados usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithNestedAttributes(
        final String tableName,
```

```
        final String partitionKeyName,
        final String partitionKeyValue,
        final String nestedPath,
        final String nestedAttr,
        final String nestedValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Nested path", nestedPath);
        CodeSampleUtils.validateStringParameter("Nested attribute", nestedAttr);
        CodeSampleUtils.validateStringParameter("Nested value", nestedValue);

        // Split the nested path into components
        final String[] pathComponents = nestedPath.split("\\.");

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

        // Build the nested attribute reference using document path notation
        final StringBuilder nestedAttributeRef = new StringBuilder();
        for (int i = 0; i < pathComponents.length; i++) {
            final String aliasName = "#n" + i;
            expressionAttributeNames.put(aliasName, pathComponents[i]);

            if (i > 0) {
                nestedAttributeRef.append(".");
            }
            nestedAttributeRef.append(aliasName);
        }

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_NESTED,
            AttributeValue.builder().s(nestedValue).build());

        // Create the filter expression using the nested attribute reference
        final String filterExpression = nestedAttributeRef + " = :nestedValue";
```

```
        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query with nested attribute filter successful.
Found " + response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with nested attribute filter: " +
e.getMessage());
            throw e;
        }
    }
```

Demonstra como consultar uma tabela do DynamoDB com atributos aninhados.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<nestedPath> <nestedAttr> <nestedValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    nestedPath - The path to the nested map attribute (e.g.,
"address").
                    nestedAttr - The name of the nested attribute (e.g., "city").
                    nestedValue - The value to filter by (e.g., "Seattle").
```

```
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String nestedPath = args[3];
        final String nestedAttr = args[4];
        final String nestedValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        System.out.println("Querying items where " + partitionKeyName + " = " +
partitionKeyValue + " and " + nestedPath
            + "." + nestedAttr + " = " + nestedValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new NestedAttributeQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withNestedPath(nestedPath)
                .withNestedAttribute(nestedAttr)
                .withNestedValue(nestedValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> {
                System.out.println(item);

                // Extract and display the nested attribute for clarity
                if (item.containsKey(nestedPath) && item.get(nestedPath).hasM())
{
                    Map<String, AttributeValue> nestedMap =
item.get(nestedPath).m();
```

```
                    if (nestedMap.containsKey(nestedAttr)) {
                        System.out.println("  Nested attribute " + nestedPath +
"." + nestedAttr + ": "
                                + formatAttributeValue(nestedMap.get(nestedAttr)));
                    }
                }
            });

            System.out.println("\nNote: When working with nested attributes in
DynamoDB:");
            System.out.println("1. Use dot notation in filter expressions to
access nested attributes");
            System.out.println("2. Use expression attribute names for each
component of the path");
            System.out.println("3. Check if the nested attribute exists before
accessing it");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte uma tabela do DynamoDB com atributos aninhados usando o AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");
```

```
/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with nested attribute: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB com atributos aninhados usando o AWS SDK para Python (Boto3).

```
from typing import Any, Dict, List

import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_path: str,
    comparison_operator: str,
    comparison_value: Any,
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_path (str): The path to the nested attribute (e.g.,
 'specs.weight').
        comparison_operator (str): The comparison operator to use ('=', '!=',
 '<', '<=', '>', '>=').
        comparison_value (any): The value to compare against.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)
```

```
    # Build the filter expression based on the nested attribute path and
 comparison operator
    filter_expression = None
    if comparison_operator == "=":
        filter_expression = Attr(nested_path).eq(comparison_value)
    elif comparison_operator == "!=":
        filter_expression = Attr(nested_path).ne(comparison_value)
    elif comparison_operator == "<":
        filter_expression = Attr(nested_path).lt(comparison_value)
    elif comparison_operator == "<=":
        filter_expression = Attr(nested_path).lte(comparison_value)
    elif comparison_operator == ">":
        filter_expression = Attr(nested_path).gt(comparison_value)
    elif comparison_operator == ">=":
        filter_expression = Attr(nested_path).gte(comparison_value)
    elif comparison_operator == "contains":
        filter_expression = Attr(nested_path).contains(comparison_value)
    elif comparison_operator == "begins_with":
        filter_expression = Attr(nested_path).begins_with(comparison_value)

    # Execute the query with the filter expression
    response = table.query(KeyConditionExpression=key_condition,
 FilterExpression=filter_expression)

    return response


def query_with_multiple_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_conditions: List[Dict[str, Any]],
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by multiple nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_conditions (list): A list of dictionaries, each containing:
            - path (str): The path to the nested attribute
            - operator (str): The comparison operator
            - value (any): The value to compare against
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the combined filter expression for all nested attributes
    combined_filter = None

    for condition in nested_conditions:
        if not isinstance(condition, dict):
            continue
        path = condition.get("path", "")
        operator = condition.get("operator", "")
        value = condition.get("value")

        if not path or not operator:
            continue

        # Build the individual filter expression
        current_filter = None
        if operator == "=":
            current_filter = Attr(path).eq(value)
        elif operator == "!=":
            current_filter = Attr(path).ne(value)
        elif operator == "<":
            current_filter = Attr(path).lt(value)
        elif operator == "<=":
            current_filter = Attr(path).lte(value)
        elif operator == ">":
            current_filter = Attr(path).gt(value)
        elif operator == ">=":
            current_filter = Attr(path).gte(value)
        elif operator == "contains":
            current_filter = Attr(path).contains(value)
        elif operator == "begins_with":
            current_filter = Attr(path).begins_with(value)

        # Combine with the existing filter using AND
```

```
        if current_filter:
            if combined_filter is None:
                combined_filter = current_filter
            else:
                combined_filter = combined_filter & current_filter

    # Execute the query with the combined filter expression
    response = table.query(KeyConditionExpression=key_condition,
FilterExpression=combined_filter)

    return response
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela no DynamoDB com paginação usando um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela com paginação.

- Implemente a paginação para os resultados da consulta do DynamoDB.
- Use LastEvaluatedKey para recuperar as páginas subsequentes.
- Controle o número de itens por página com o parâmetro Limit.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB com paginação usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;
```

```
        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }

                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
```

```
        }
    }
```

Demonstra como consultar uma tabela do DynamoDB com paginação.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue> [pageSize]
[region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    pageSize (optional) - The maximum number of items to return
per page. (Default: 10)
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final int pageSize = args.length > 3 ? Integer.parseInt(args[3]) : 10;
        final Region region = args.length > 4 ? Region.of(args[4]) :
Region.US_EAST_1;

        System.out.println("Querying items with pagination (page size: " +
pageSize + ")");

        try {
            // Using the builder pattern to create and execute the query
            final List<Map<String, AttributeValue>> allItems = new
PaginationQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
```

```
                .withPartitionKeyValue(partitionKeyValue)
                .withPageSize(pageSize)
                .withRegion(region)
                .executeWithPagination();

            // Process the results
            System.out.println("\nSummary: Retrieved a total of " +
allItems.size() + " items");

            // Display the first few items as a sample
            final int sampleSize = Math.min(5, allItems.size());
            if (sampleSize > 0) {
                System.out.println("\nSample of retrieved items (first " +
sampleSize + "):");
                for (int i = 0; i < sampleSize; i++) {
                    System.out.println(allItems.get(i));
                }

                if (allItems.size() > sampleSize) {
                    System.out.println("... and " + (allItems.size() -
sampleSize) + " more items");
                }
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte uma tabela do DynamoDB com paginação usando o AWS SDK para JavaScript.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;
```

```
   // Loop until all pages are retrieved
   do {
     // Construct the query input
     const input = {
       TableName: tableName,
       KeyConditionExpression: "#pk = :pkValue",
       Limit: pageSize,
       ExpressionAttributeNames: {
         "#pk": partitionKeyName
       },
       ExpressionAttributeValues: {
         ":pkValue": { S: partitionKeyValue }
       }
     };

     // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
     if (lastEvaluatedKey) {
       input.ExclusiveStartKey = lastEvaluatedKey;
     }

     // Execute the query
     const command = new QueryCommand(input);
     const response = await client.send(command);

     // Process the current page of results
     pageCount++;
     console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

     // Add the items from this page to our collection
     if (response.Items && response.Items.length > 0) {
       allItems.push(...response.Items);
     }

     // Get the LastEvaluatedKey for the next page
     lastEvaluatedKey = response.LastEvaluatedKey;

   } while (lastEvaluatedKey); // Continue until there are no more pages

   console.log(`Query complete. Retrieved ${allItems.length} items in
${pageCount} pages.`);
   return allItems;
 } catch (error) {
```

```
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB com paginação usando o AWS SDK para Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key
```

```
def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
```

```
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
    """
    Query a DynamoDB table with pagination using a generator to handle large
 result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar uma tabela do DynamoDB com leituras altamente consistentes usando um SDK da AWS

Os exemplos de código a seguir mostram como consultar uma tabela com leituras altamente consistentes.

- Configure o nível de consistência das consultas do DynamoDB.
- Use leituras altamente consistentes para obter os dados mais atualizados.
- Entenda as vantagens e desvantagens entre consistência final e consistência forte.

Java

SDK para Java 2.x

Consulte uma tabela do DynamoDB com consistência de leitura configurável usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithConsistentReads(
```

```
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final boolean useConsistentRead) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .consistentRead(useConsistentRead)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query successful. Found {0} items",
response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte uma tabela do DynamoDB com consistência de leitura configurável usando o AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
```

```
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte uma tabela do DynamoDB com a opção de leituras altamente consistentes usando o AWS SDK para Python (Boto3).

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
```

```
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
ConsistentRead=consistent_read)

    return response
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar dados do DynamoDB usando declarações SELECT do PartiQL com um SDK da AWS

O exemplo de código a seguir mostra como consultar dados usando declarações SELECT do PartiQL.

JavaScript

SDK para JavaScript (v3)

Consulte itens de uma tabela do DynamoDB usando declarações SELECT do PartiQL com o AWS SDK para JavaScript.

```
/**
 * This example demonstrates how to query items from a DynamoDB table using
 PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
```

```
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };
```

```
  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
 begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select items using a between condition for range queries.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for range
 * @param startValue - The start value of the range
 * @param endValue - The end value of the range
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ?
 AND ?`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");
```

```
  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456",
 "productId", "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01",
 "2023-12-31");
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar itens com TTL em uma tabela do DynamoDB usando um SDK da AWS

Os exemplos de código a seguir mostram como consultar itens com TTL.

Java

SDK para Java 2.x

Use uma expressão filtrada na consulta para reunir os itens com TTL em uma tabela do DynamoDB por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Use uma expressão filtrada na consulta para reunir os itens com TTL em uma tabela do DynamoDB por meio do AWS SDK para JavaScript.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}
```

```
// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Use uma expressão filtrada na consulta para reunir os itens com TTL em uma tabela do DynamoDB por meio do AWS SDK para Python (Boto3).

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
```

```
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Consultar tabelas do DynamoDB usando padrões de data e hora com um SDK da AWS

Os exemplos de código a seguir mostram como consultar tabelas usando padrões de data e hora.

- Armazene e consulte valores de data/hora no DynamoDB.
- Implemente consultas de intervalo de datas usando chaves de classificação.
- Formate strings de data para obter uma consulta eficaz.

Java

SDK para Java 2.x

Consulte usando intervalos de datas em chaves de classificação com o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();
```

```
        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

Consulte usando variáveis de data e hora por meio do AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final String startDate,
        final String endDate) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateDateRangeParameters(dateKeyName, startDate,
 endDate);
        CodeSampleUtils.validateDateFormat("Start date", startDate);
        CodeSampleUtils.validateDateFormat("End date", endDate);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put("#dateKey", dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            ":startDate", AttributeValue.builder().s(startDate).build());
```

```
        expressionAttributeValues.put(
            ":endDate", AttributeValue.builder().s(endDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query successful. Found " + response.count() + "
items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with date range: " +
e.getMessage());
            throw e;
        }
    }
```

Consulte dentro dos intervalos de datas nos carimbos de data/hora de época do Unix com o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;
```

```
   public QueryResponse queryWithDateTimeEpoch(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final String dateKeyName,
       final long startEpoch,
       final long endEpoch) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
       CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
       CodeSampleUtils.validateEpochTimestamp("Start epoch", startEpoch);
       CodeSampleUtils.validateEpochTimestamp("End epoch", endEpoch);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
       expressionAttributeNames.put("#dateKey", dateKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
       expressionAttributeValues.put(
           EXPRESSION_ATTRIBUTE_VALUE_PK,
           AttributeValue.builder().s(partitionKeyValue).build());
       expressionAttributeValues.put(
           ":startDate",
AttributeValue.builder().n(String.valueOf(startEpoch)).build());
       expressionAttributeValues.put(
           ":endDate",
AttributeValue.builder().n(String.valueOf(endEpoch)).build());

       // Create the query request
       final QueryRequest queryRequest = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression(KEY_CONDITION_EXPRESSION)
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .build();

       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
```

```
            System.out.println("Query successful. Found " + response.count() + "
 items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with epoch timestamps: " +
 e.getMessage());
            throw e;
        }
    }
```

Consulte dentro dos intervalos de datas usando objetos LocalDateTime com o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTimeLocalDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDateTime startDateTime,
        final LocalDateTime endDateTime) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
        if (startDateTime == null || endDateTime == null) {
```

```
            throw new IllegalArgumentException("Start and end LocalDateTime must
 not be null");
        }

        // Convert LocalDateTime to ISO-8601 strings in UTC with the correct
 format
        final String startDate =
 startDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);
        final String endDate =
 endDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);

        return queryWithDateTime(tableName, partitionKeyName, partitionKeyValue,
 dateKeyName, startDate, endDate);
    }
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte usando intervalos de datas em chaves de classificação com o AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
```

```
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

Consulte usando variáveis de data e hora por meio do AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
 AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
```

```
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte usando intervalos de datas em chaves de classificação com o AWS SDK para Python (Boto3).

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
```

```
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

Consulte usando variáveis de data e hora por meio do AWS SDK para Python (Boto3).

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_datetime(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range filter on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date/
time values).
        start_date (datetime): The start date/time for the query range.
        end_date (datetime): The end date/time for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date/time values as ISO 8601 strings
    # DynamoDB works well with ISO format for date/time values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
```

```
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response



def example_usage():
    """Example of how to use the query_with_datetime function."""
    # Example parameters
    table_name = "Events"
    partition_key_name = "EventType"
    partition_key_value = "UserLogin"
    sort_key_name = "Timestamp"

    # Create date/time variables for the query
    end_date = datetime.now()
    start_date = end_date - timedelta(days=7)  # Query events from the last 7
 days

    print(f"Querying events from {start_date.isoformat()} to
 {end_date.isoformat()}")

    # Execute the query
    response = query_with_datetime(
        table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Event: {item}")
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Salvar o EXIF e outras informações de imagem usando um AWS SDK

O código de exemplo a seguir mostra como:

- Obter informações de EXIF de um arquivo JPG, JPEG ou PNG.
- Fazer upload do arquivo de imagem para um bucket do Amazon S3.
- Usar o Amazon Rekognition para identificar os três principais atributos (rótulos) no arquivo.
- Adicione as informações de EXIF e rótulo a uma tabela do Amazon DynamoDB na região.

Rust

SDK para Rust

Obtenha informações de EXIF de um arquivo JPG, JPEG ou PNG, faça upload do arquivo de imagem para um bucket do Amazon S3, use o Amazon Rekognition para identificar os três principais atributos (rótulos no Amazon Rekognition) no arquivo e adicione as informações de EXIF e de rótulo a uma tabela do Amazon DynamoDB na região.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon Rekognition
- Amazon S3

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Configurar o controle de acesso por atributo para o DynamoDB usando a AWS Command Line Interface v2

O exemplo de código apresentado a seguir demonstra como implementar o controle de acesso por atributo (ABAC) no DynamoDB.

- Crie uma política do IAM para o ABAC.
- Crie tabelas com etiquetas que representem diferentes departamentos.
- Liste e filtre tabelas com base nas etiquetas.

Bash

AWS CLI com script Bash

Crie uma política do IAM para o ABAC.

```
# Step 1: Create a policy document for ABAC
cat > abac-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
```

```
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}",
          "aws:ResourceTag/Environment": "Development"
        }
      }
    }
  ]
}
EOF

# Step 2: Create the IAM policy
aws iam create-policy \
    --policy-name DynamoDBDepartmentBasedAccess \
    --policy-document file://abac-policy.json
```

Crie tabelas com etiquetas que representem diferentes departamentos.

```
# Create a DynamoDB table with tags for ABAC
aws dynamodb create-table \
    --table-name FinanceData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Finance \
        Key=Environment,Value=Development

# Create another table with different tags
aws dynamodb create-table \
    --table-name MarketingData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
```

```
    --tags \
        Key=Department,Value=Marketing \
        Key=Environment,Value=Production
```

Liste e filtre tabelas com base nas etiquetas.

```
# List all DynamoDB tables
echo "Listing all tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nGetting ARNs for all tables:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# For each table ARN, list its tags
echo -e "\nListing tags for each table:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    echo -e "\nTags for table: $TABLE_NAME"
    aws dynamodb list-tags-of-resource --resource-arn $ARN
done

# Example: Find tables with a specific tag
echo -e "\nFinding tables with Environment=Production tag:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreatePolicy
  - CreateTable
  - ListTables

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Com funciona a ordem da expressão de atualização no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como funciona a ordem da expressão de atualização.

- Saiba como o DynamoDB processa expressões de atualização.
- Entenda como funciona a ordem das operações nas expressões de atualização.
- Evite resultados inesperados entendendo a avaliação da expressão.

Java

SDK para Java 2.x

Demonstre a ordem da expressão de atualização usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Demonstrates the effect of update expression order.
     *
     * <p>This method shows how the order of operations in an update expression
     * affects the result of the update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return Map containing the results of different update orders
```

```
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> demonstrateUpdateOrder(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Initialize the item with a counter
            UpdateItemRequest initRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :zero, OldCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            dynamoDbClient.updateItem(initRequest);

            // Example 1: SET first, then ADD
            UpdateItemRequest setFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value ADD
OldCounter :increment")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse setFirstResponse =
dynamoDbClient.updateItem(setFirstRequest);
            results.put("setFirstResponse", setFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 2: ADD first, then SET
            UpdateItemRequest addFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
```

```
                .updateExpression("ADD Counter :increment SET OldCounter
= :value")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse addFirstResponse =
dynamoDbClient.updateItem(addFirstRequest);
            results.put("addFirstResponse", addFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 3: SET with multiple attributes
            UpdateItemRequest multiSetRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value, OldCounter = Counter")
                .expressionAttributeValues(
                    Map.of(":value", AttributeValue.builder().n("10").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse multiSetResponse =
dynamoDbClient.updateItem(multiSetRequest);
            results.put("multiSetResponse", multiSetResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 4: SET with expression using the same attribute
            UpdateItemRequest selfReferenceRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = Counter + :increment, OldCounter
= Counter")
                .expressionAttributeValues(
                    Map.of(":increment",
AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();
```

```
            UpdateItemResponse selfReferenceResponse =
dynamoDbClient.updateItem(selfReferenceRequest);
            results.put("selfReferenceResponse", selfReferenceResponse);

            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Updates an item with SET first, then REMOVE.
     *
     * <p>This method demonstrates updating an item with SET operation first,
     * followed by a REMOVE operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithSetFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue REMOVE #removeAttr")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
```

```
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with SET first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with REMOVE first, then SET.
     *
     * <p>This method demonstrates updating an item with REMOVE operation first,
     * followed by a SET operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithRemoveFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
```

```
            .updateExpression("REMOVE #removeAttr SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with REMOVE first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with all operation types in a specific order.
     *
     * <p>This method demonstrates using all operation types (SET, REMOVE, ADD,
DELETE)
     * in a specific order in a single update expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithAllOperationTypes(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #stringAttr = :stringVal, #mapAttr.#nestedAttr
= :nestedVal " + "REMOVE #oldAttr "
                + "ADD #counterAttr :increment "
```

```
                + "DELETE #stringSetAttr :stringSetVal")
            .expressionAttributeNames(Map.of(
                "#stringAttr", "StringAttribute",
                "#mapAttr", "MapAttribute",
                "#nestedAttr", "NestedAttribute",
                "#oldAttr", "OldAttribute",
                "#counterAttr", "CounterAttribute",
                "#stringSetAttr", "StringSetAttribute"))
            .expressionAttributeValues(Map.of(
                ":stringVal", AttributeValue.builder().s("New Value").build(),
                ":nestedVal", AttributeValue.builder().s("Nested Value").build(),
                ":increment", AttributeValue.builder().n("1").build(),
                ":stringSetVal", AttributeValue.builder().ss("Value1").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with all operation types: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Gets the current state of an item.
     *
     * <p>Helper method to retrieve the current state of an item.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @return The item or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, AttributeValue> getItem(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the get parameters
```

```
        GetItemRequest request =
            GetItemRequest.builder().tableName(tableName).key(key).build();

        // Perform the get operation
        try {
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the item if it exists, otherwise null
            return response.item();
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get item: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Exemplo de uso da ordem da expressão de atualização com o AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating update expression order in DynamoDB");

        try {
            // Example 1: Demonstrate update order effects
            System.out.println("\nExample 1: Demonstrating update order
effects");
            Map<String, Object> orderResults =
demonstrateUpdateOrder(dynamoDbClient, tableName, key);

            if ((boolean) orderResults.get("success")) {
                System.out.println("SET first, then ADD:");
                System.out.println("  " + orderResults.get("setFirstResponse"));

                System.out.println("ADD first, then SET:");
                System.out.println("  " + orderResults.get("addFirstResponse"));

                System.out.println("SET with multiple attributes:");
```

```
                System.out.println("  " + orderResults.get("multiSetResponse"));

                System.out.println("SET with self-reference:");
                System.out.println("  " +
orderResults.get("selfReferenceResponse"));
            } else {
                System.out.println("Error: " + orderResults.get("error"));
            }

            // Example 2: Update with SET first, then REMOVE
            System.out.println("\nExample 2: Update with SET first, then
REMOVE");
            UpdateItemResponse setFirstResponse = updateWithSetFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Active").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
setFirstResponse.attributes());

            // Example 3: Update with REMOVE first, then SET
            System.out.println("\nExample 3: Update with REMOVE first, then
SET");
            UpdateItemResponse removeFirstResponse = updateWithRemoveFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Inactive").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
removeFirstResponse.attributes());

            // Example 4: Update with all operation types
            System.out.println("\nExample 4: Update with all operation types");
            UpdateItemResponse allOpsResponse =
updateWithAllOperationTypes(dynamoDbClient, tableName, key);

            System.out.println("Updated attributes: " +
allOpsResponse.attributes());
```

```
            // Example 5: Get the current state of the item
            System.out.println("\nExample 5: Current state of the item");
            Map<String, AttributeValue> item = getItem(dynamoDbClient, tableName,
key);

            if (item != null) {
                System.out.println("Item: " + item);
            } else {
                System.out.println("Item not found");
            }

            // Explain update expression order
            System.out.println("\nKey points about update expression order in
DynamoDB:");
            System.out.println("1. Update expressions are processed in this
order: SET, REMOVE, ADD, DELETE");
            System.out.println("2. Within each clause, operations are processed
from left to right");
            System.out.println("3. SET operations use the item state before any
updates in the expression");
            System.out.println("4. When an attribute is referenced multiple
times, the first operation wins");
            System.out.println("5. To reference a new value, split the update
into multiple operations");
            System.out.println("6. The order of clauses in the expression doesn't
change the evaluation order");
            System.out.println("7. For complex updates, consider using multiple
separate update operations");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demonstre a ordem da expressão de atualização usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
```

```
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}

/**
 * Demonstrate that variables hold copies of existing values before
 modifications.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that uses the values of attributes before they are modified in the same
 expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
```

```
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = { ...key, a: 1, b: 2, c: 3 };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with an expression that uses values before they are
modified
  // This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
the value of 'b'
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE a SET b = a, c = b",
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
```

```
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
 order
 * in which they are processed.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
```

```
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
 counter :increment DELETE set_attr :elements",
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
      ":increment": 5,
      ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
```

```
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET ";
  const expressionAttributeValues = {};

  // Add each attribute to the update expression
  Object.entries(attributes).forEach(([attrName, attrValue], index) => {
    const valuePlaceholder = `:val${index}`;

    if (index > 0) {
      updateExpression += ", ";
    }
    updateExpression += `${attrName} = ${valuePlaceholder}`;

    expressionAttributeValues[valuePlaceholder] = attrValue;
  });

  // Execute the update
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Update an attribute with a value from another attribute or a default value.
```

```
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a
 value
 * from one attribute to another, or use a default value if the source doesn't
 exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
 * @param {any} defaultValue - The default value to use if the source attribute
 doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
 if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
```

```
 * Demonstrate complex update expressions with multiple operations on the same
 attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
    map_attr: {
      nested1: "value1",
      nested2: "value2"
    }
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));
```

```
  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple operations on the same attributes
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `
      SET counter = counter + :inc1,
          counter = counter + :inc2,
          map_attr.nested1 = :new_val1,
          map_attr.nested3 = :new_val3,
          list_attr[0] = list_attr[1],
          list_attr[1] = list_attr[2]
    `,
    ExpressionAttributeValues: {
      ":inc1": 5,
      ":inc2": 3,
      ":new_val1": "updated_value1",
      ":new_val3": "new_value3"
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}
```

Exemplo de uso da ordem da expressão de atualização com o AWS SDK para JavaScript.

```
/**
```

```
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update
 expressions");
    const results1 = await demonstrateValueCopying(
      config,
      tableName,
      { OrderId: "order123" }
    );

    console.log("Initial state:", JSON.stringify(results1.initialState, null,
 2));
    console.log("Update response:", JSON.stringify(results1.updateResponse, null,
 2));
    console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The initial state had a=1, b=2, c=3");
    console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
 following:");
    console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
    console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
    console.log("   - Removed the attribute 'a'");
    console.log("3. The final state has b=1, c=2, and 'a' is removed");
    console.log("4. This demonstrates that DynamoDB uses the values of attributes
 as they were BEFORE any modifications");

    // Example 2: Demonstrating the order of different action types
    console.log("\nExample 2: Demonstrating the order of different action
 types");
    const results2 = await demonstrateActionOrder(
      config,
      tableName,
      { OrderId: "order456" }
    );
```

```
   console.log("Initial state:", JSON.stringify(results2.initialState, null,
2));
   console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The update expression contained multiple action types:
REMOVE, SET, ADD, DELETE");
   console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
   console.log("3. First, 'to_remove' was removed");
   console.log("4. Then, 'to_modify' was set to a new value");
   console.log("5. Next, 'counter' was incremented by 5");
   console.log("6. Finally, 'B' was removed from the set attribute");

   // Example 3: Updating multiple attributes in a single SET action
   console.log("\nExample 3: Updating multiple attributes in a single SET
action");
   const response3 = await updateWithMultipleSetActions(
     config,
     tableName,
     { OrderId: "order789" },
     {
       Status: "Shipped",
       ShippingDate: "2025-05-28",
       TrackingNumber: "1Z999AA10123456784"
     }
   );

   console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

   // Example 4: Conditional value copying with if_not_exists
   console.log("\nExample 4: Conditional value copying with if_not_exists");
   const response4 = await updateWithConditionalValueCopying(
     config,
     tableName,
     { OrderId: "order101" },
     "PreferredShippingMethod",
     "ShippingMethod",
     "Standard"
   );
```

```
   console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

   // Example 5: Multiple operations on the same attribute
   console.log("\nExample 5: Multiple operations on the same attribute");
   const results5 = await demonstrateMultipleOperationsOnSameAttribute(
     config,
     tableName,
     { OrderId: "order202" }
   );

   console.log("Initial state:", JSON.stringify(results5.initialState, null,
2));
   console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The counter was incremented twice (first by 5, then by 3) for
a total of +8");
   console.log("2. The map attribute had one value updated and a new nested
attribute added");
   console.log("3. The list attribute had values shifted (value at index 1 moved
to index 0, value at index 2 moved to index 1)");
   console.log("4. All operations within the SET action are processed from left
to right");

   // Key points about update expression order of operations
   console.log("\nKey Points About Update Expression Order of Operations:");
   console.log("1. Variables in expressions hold copies of attribute values as
they existed BEFORE any modifications");
   console.log("2. Multiple actions in an update expression are processed in
this order: REMOVE, SET, ADD, DELETE");
   console.log("3. Within each action type, operations are processed from left
to right");
   console.log("4. You can reference the same attribute multiple times in an
expression");
   console.log("5. You can use if_not_exists() to conditionally set values based
on attribute existence");
   console.log("6. Using a single update expression with multiple actions is
more efficient than multiple separate updates");
   console.log("7. The update expression is atomic - either all actions succeed
or none do");
```

```
  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demonstre a ordem da expressão de atualização usando o AWS SDK para Python (Boto3).

```
import boto3
import json
from typing import Any, Dict, Optional


def update_with_multiple_actions(
    table_name: str,
    key: Dict[str, Any],
    update_expression: str,
    expression_attribute_names: Optional[Dict[str, str]] = None,
    expression_attribute_values: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update an item with multiple actions in a single update expression.

    This function demonstrates how to use multiple actions in a single update
 expression
    and how DynamoDB processes these actions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        update_expression (str): The update expression with multiple actions.
        expression_attribute_names (Optional[Dict[str, str]]): Expression
 attribute name placeholders.
        expression_attribute_values (Optional[Dict[str, Any]]): Expression
 attribute value placeholders.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Prepare the update parameters
    update_params = {
        "Key": key,
        "UpdateExpression": update_expression,
        "ReturnValues": "UPDATED_NEW",
    }

    # Add expression attribute names if provided
    if expression_attribute_names:
        update_params["ExpressionAttributeNames"] = expression_attribute_names

    # Add expression attribute values if provided
    if expression_attribute_values:
        update_params["ExpressionAttributeValues"] = expression_attribute_values

    # Execute the update
    response = table.update_item(**update_params)

    return response


def demonstrate_value_copying(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate that variables hold copies of existing values before
 modifications.

    This function creates an item with initial values, then updates it with an
 expression
    that uses the values of attributes before they are modified in the same
 expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.
```

```
    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update({"a": 1, "b": 2, "c": 3})

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with an expression that uses values before they are
 modified
    # This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
 the value of 'b'
    update_response = table.update_item(
        Key=key, UpdateExpression="REMOVE a SET b = a, c = b",
 ReturnValues="UPDATED_NEW"
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def demonstrate_action_order(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate the order in which different action types are processed.
```

```
    This function creates an item with initial values, then updates it with an
expression
    that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
order
    in which they are processed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update(
        {
            "counter": 10,
            "set_attr": set(["A", "B", "C"]),
            "to_remove": "This will be removed",
            "to_modify": "Original value",
        }
    )

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with multiple action types
    # The actions will be processed in this order: REMOVE, SET, ADD, DELETE
    update_response = table.update_item(
        Key=key,
        UpdateExpression="REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
        ExpressionAttributeValues={
            ":new_value": "Updated value",
            ":increment": 5,
            ":elements": set(["B"]),
```

```
        },
        ReturnValues="UPDATED_NEW",
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def update_with_multiple_set_actions(
    table_name: str, key: Dict[str, Any], attributes: Dict[str, Any]
) -> Dict[str, Any]:
    """
    Update multiple attributes with a single SET action.

    This function demonstrates how to update multiple attributes in a single SET
 action,
    which is more efficient than using multiple separate update operations.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        attributes (Dict[str, Any]): The attributes to update and their new
 values.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression and expression attribute values
    update_expression = "SET "
    expression_attribute_values = {}
```

```
    # Add each attribute to the update expression
    for i, (attr_name, attr_value) in enumerate(attributes.items()):
        value_placeholder = f":val{i}"

        if i > 0:
            update_expression += ", "
        update_expression += f"{attr_name} = {value_placeholder}"

        expression_attribute_values[value_placeholder] = attr_value

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_with_conditional_value_copying(
    table_name: str,
    key: Dict[str, Any],
    source_attribute: str,
    target_attribute: str,
    default_value: Any,
) -> Dict[str, Any]:
    """
    Update an attribute with a value from another attribute or a default value.

    This function demonstrates how to use if_not_exists to conditionally copy a
 value
    from one attribute to another, or use a default value if the source doesn't
 exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        source_attribute (str): The attribute to copy the value from.
        target_attribute (str): The attribute to update.
        default_value (Any): The default value to use if the source attribute
 doesn't exist.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to conditionally copy the value
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {target_attribute} =
 if_not_exists({source_attribute}, :default)",
        ExpressionAttributeValues={":default": default_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Exemplo de uso da ordem da expressão de atualização com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use update expression order of operations in
 DynamoDB."""
    # Example parameters
    table_name = "OrderProcessing"
    key = {"OrderId": "order123"}

    print("Example 1: Demonstrating value copying in update expressions")
    try:
        results = demonstrate_value_copying(table_name=table_name, key=key)

        print(f"Initial state: {json.dumps(results['initial_state'],
 default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
 default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
```

```
        print("1. The initial state had a=1, b=2, c=3")
        print("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:")
        print("   - Copied the value of 'a' (which was 1) to be used for 'b'")
        print("   - Copied the value of 'b' (which was 2) to be used for 'c'")
        print("   - Removed the attribute 'a'")
        print("3. The final state has b=1, c=2, and 'a' is removed")
        print(
            "4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications"
        )
    except Exception as e:
        print(f"Error demonstrating value copying: {e}")

    print("\nExample 2: Demonstrating the order of different action types")
    try:
        results = demonstrate_action_order(table_name=table_name, key={"OrderId":
"order456"})

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE")
        print("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE")
        print("3. First, 'to_remove' was removed")
        print("4. Then, 'to_modify' was set to a new value")
        print("5. Next, 'counter' was incremented by 5")
        print("6. Finally, 'B' was removed from the set attribute")
    except Exception as e:
        print(f"Error demonstrating action order: {e}")

    print("\nExample 3: Updating multiple attributes in a single SET action")
    try:
        response = update_with_multiple_set_actions(
            table_name=table_name,
            key={"OrderId": "order789"},
            attributes={
                "Status": "Shipped",
```

```
                "ShippingDate": "2025-05-14",
                "TrackingNumber": "1Z999AA10123456784",
            },
        )

        print(
            f"Multiple attributes updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating multiple attributes: {e}")

    print("\nExample 4: Conditional value copying with if_not_exists")
    try:
        response = update_with_conditional_value_copying(
            table_name=table_name,
            key={"OrderId": "order101"},
            source_attribute="PreferredShippingMethod",
            target_attribute="ShippingMethod",
            default_value="Standard",
        )

        print(
            f"Conditional value copying result:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error with conditional value copying: {e}")

    print("\nKey Points About Update Expression Order of Operations:")
    print(
        "1. Variables in expressions hold copies of attribute values as they
existed BEFORE any modifications"
    )
    print(
        "2. Multiple actions in an update expression are processed in this order:
REMOVE, SET, ADD, DELETE"
    )
    print("3. Within each action type, operations are processed from left to
right")
    print("4. You can reference the same attribute multiple times in an
expression")
    print("5. You can use if_not_exists() to conditionally set values based on
attribute existence")
```

```
    print(
        "6. Using a single update expression with multiple actions is more
efficient than multiple separate updates"
    )
    print("7. The update expression is atomic - either all actions succeed or
none do")
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Atualizar uma configuração de tabela do DynamoDB com throughput a quente usando um AWS SDK

Os exemplos de código a seguir mostram como atualizar a configuração de throughput a quente de uma tabela.

Java

SDK para Java 2.x

Atualize a configuração de throughput a quente em uma tabela existente do DynamoDB usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
```

```
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
    }

    /**
     * Updates a DynamoDB table with warm throughput settings for both the table
and a global secondary index.
     *
     * @param ddb The DynamoDB client
     * @param tableName The name of the table to update
     * @param tableReadUnitsPerSecond Read units per second for the table
     * @param tableWriteUnitsPerSecond Write units per second for the table
     * @param globalSecondaryIndexName The name of the global secondary index to
update
     * @param globalSecondaryIndexReadUnitsPerSecond Read units per second for
the GSI
     * @param globalSecondaryIndexWriteUnitsPerSecond Write units per second for
the GSI
     */
    public static void updateDynamoDBTable(
        final DynamoDbClient ddb,
        final String tableName,
        final Long tableReadUnitsPerSecond,
        final Long tableWriteUnitsPerSecond,
        final String globalSecondaryIndexName,
        final Long globalSecondaryIndexReadUnitsPerSecond,
        final Long globalSecondaryIndexWriteUnitsPerSecond) {

        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableReadUnitsPerSecond,
tableWriteUnitsPerSecond);
        final WarmThroughput gsiWarmThroughput =
            buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
globalSecondaryIndexWriteUnitsPerSecond);

        final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                .indexName(globalSecondaryIndexName)
                .warmThroughput(gsiWarmThroughput)
                .build())
            .build();
```

```
        final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

        try {
            ddb.updateTable(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }

        System.out.println(SUCCESS_MESSAGE);
    }
```

- Para obter detalhes da API, consulte UpdateTable na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Atualize a configuração de throughput a quente em uma tabela existente do DynamoDB usando o AWS SDK para JavaScript.

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
```

```
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response:
 ${JSON.stringify(response)}`);
    return response;
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
  5, 5,
  'example-index',
  2, 2
);
*/
```

- Para obter detalhes da API, consulte UpdateTable na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Atualize a configuração de throughput a quente em uma tabela existente do DynamoDB usando o AWS SDK para Python (Boto3).

```
from boto3 import client
from botocore.exceptions import ClientError


def update_dynamodb_table_warm_throughput(
    table_name,
    table_read_units,
    table_write_units,
    gsi_name,
    gsi_read_units,
    gsi_write_units,
    region_name="us-east-1",
):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    :return: The response from the update_table operation
    """
    try:
        ddb = client("dynamodb", region_name=region_name)

        # Update the table's warm throughput
```

```
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units,
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units,
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {"Update": {"IndexName": gsi_name, "WarmThroughput":
gsi_warm_throughput}}
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput,
        }

        # Update the table
        response = ddb.update_table(**update_table_request)
        print("Table updated successfully!")
        return response  # Make sure to return the response
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

- Para detalhes da API, consulte UpdateTable na Referência da API do AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Atualizar um item do DynamoDB com TTL usando um SDK da AWS

Os exemplos de código a seguir mostram como atualizar a TTL de um item.

Java

SDK para Java 2.x

Atualize a TTL em um item do DynamoDB existente em uma tabela.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
```

```
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
```

```
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
```

```
)
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Atualizar dados do DynamoDB usando declarações UPDATE do PartiQL com um SDK da AWS

O exemplo de código a seguir mostra como atualizar dados usando declarações UPDATE do PartiQL.

JavaScript

SDK para JavaScript (v3)

Atualize itens na tabela do DynamoDB usando as declarações UPDATE do PartiQL com o AWS SDK para JavaScript.

```
/**
 * This example demonstrates how to update items in a DynamoDB table using
 PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
```

```
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new
 values
 * @returns The response from the ExecuteStatementCommand
 */
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
```

```
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
 value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName}
 = ?`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
```

```
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item with a condition to ensure the update only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
```

```
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
 ${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
 "newemail@example.com");

  // Update multiple attributes at once
  await updateMultipleAttributes("UsersTable", "userId", "user123", {
    email: "newemail@example.com",
    name: "John Smith",
    lastLogin: new Date().toISOString(),
  });

  // Update an item with a composite key (partition key + sort key)
  await updateItemWithCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
```

```
    "prod789",
    "quantity",
    5
  );

  // Update with a condition
  await updateItemWithCondition(
    "UsersTable",
    "userId",
    "user123",
    "userStatus",
    "active",
    "userType",
    "premium"
  );

  // Batch update multiple items
  await batchUpdateItems("UsersTable", [
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user123",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user456",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
  ]);
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar o API Gateway para invocar uma função do Lambda

Os exemplos de código a seguir mostram como criar uma função do AWS Lambda invocada pelo Amazon API Gateway.

Java

SDK para Java 2.x

Mostra como criar uma função do AWS Lambda usando a API de runtime de Java do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma função do Lambda invocada pelo Amazon API Gateway que verifica uma tabela do Amazon DynamoDB em busca de aniversários de trabalho e usa o Amazon Simple Notification Service (Amazon SNS) para enviar uma mensagem de texto aos seus funcionários que os parabeniza em sua data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Mostra como criar uma função do AWS Lambda usando a API de runtime de JavaScript do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma função do Lambda invocada pelo Amazon API Gateway que verifica uma tabela do Amazon DynamoDB em busca de aniversários de trabalho e usa o Amazon Simple Notification Service (Amazon SNS) para enviar uma

mensagem de texto aos seus funcionários que os parabeniza em sua data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK para JavaScript v3.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK para Python (Boto3).

Este exemplo mostra como criar e usar uma API REST do Amazon API Gateway cujo alvo é uma função do AWS Lambda. O manipulador do Lambda mostra como rotear com base em métodos HTTP; como obter dados da string de consulta, do cabeçalho e do corpo e como retornar uma resposta JSON.

- Implante uma função do Lambda.
- Crie uma API REST do API Gateway.
- Criar um recurso REST cujo alvo seja a função do Lambda.
- Conceda permissão para que o API Gateway possa invocar a função do Lambda.
- Use o pacote Requests para enviar solicitações à API REST.
- Limpe todos os recursos criados durante a demonstração.

Este exemplo é mais bem visualizado no GitHub. Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB

- Lambda
- Amazon SNS

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar Step Functions para invocar funções do Lambda

Os exemplos de código a seguir mostram como criar uma máquina de estado do AWS Step Functions que invoca funções do AWS Lambda em sequência.

Java

SDK para Java 2.x

Mostra como criar um AWS fluxo de trabalho sem servidor usando o AWS Step Functions e o AWS SDK for Java 2.x. Cada etapa do fluxo de trabalho é implementada usando uma função do AWS Lambda.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar um modelo de documento para o DynamoDB usando um AWS SDK

O exemplo de código a seguir mostra como realizar operações de criação, leitura, atualização e exclusão (CRUD) e operações em lote usando um modelo de documento para o DynamoDB e um AWS SDK.

Para obter mais informações, consulte o modelo de documento.

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Execute operações CRUD usando um modelo de documento.

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
        UpdateMultipleAttributes(productCatalog, sampleBookId);
        UpdateBookPriceConditionally(productCatalog, sampleBookId);

        // Delete.
        await DeleteBook(productCatalog, sampleBookId);
    }

    /// <summary>
    /// Loads the contents of a DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB client object.</param>
```

```
        /// <param name="tableName">The name of the table to load.</param>
        /// <returns>A DynamoDB table object.</returns>
        public static Table LoadTable(IAmazonDynamoDB client, string tableName)
        {
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };

            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
```

```
        public static async void RetrieveBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };
```

```
            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally()
***");

            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,
                ["Price"] = 29.99,
            };

            // For conditional price update, creating a condition expression.
            var expr = new Expression
            {
                ExpressionStatement = "Price = :val",
            };
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
```

```
            var config = new UpdateItemOperationConfig
            {
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Deletes the book with the supplied Id value from the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task DeleteBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");

            // Optional configuration.
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
            Document document = await
productCatalog.DeleteItemAsync(sampleBookId, config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
```

```
        {
            if (updatedDocument is null)
            {
                return;
            }

            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
            }
        }
    }
```

Execute operações de gravação em lote usando um modelo de documento.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
```

```
    public class MidLevelBatchWriteItem
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown
at this time.
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in
SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }

        /// <summary>
```

```
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in
MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

Verifique uma tabela usando um modelo de documento.

```
   /// <summary>
   /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
   /// table for values.
   /// </summary>
   public class MidLevelScanOnly
   {
       public static async Task Main()
       {
           IAmazonDynamoDB client = new AmazonDynamoDBClient();

           Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");

           await FindProductsWithNegativePrice(productCatalogTable);
           await FindProductsWithNegativePriceWithConfig(productCatalogTable);
       }

       /// <summary>
       /// Retrieves any products that have a negative price in a DynamoDB
table.
       /// </summary>
       /// <param name="productCatalogTable">A DynamoDB table object.</param>
       public static async Task FindProductsWithNegativePrice(
         Table productCatalogTable)
       {
           // Assume there is a price error. So we scan to find items priced <
0.
           var scanFilter = new ScanFilter();
           scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

           Search search = productCatalogTable.Scan(scanFilter);

           do
           {
               var documentList = await search.GetNextSetAsync();
               Console.WriteLine("\nFindProductsWithNegativePrice:
printing .............");
```

```
                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .............");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }
```

```
        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Consulte e verifique uma tabela usando um modelo de documento.

```
    /// <summary>
    /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class MidLevelQueryAndScan
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();
```

```
            // Query examples.
            Table replyTable = Table.LoadTable(client, "Reply");
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 2";

            await FindRepliesInLast15Days(replyTable);
            await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
            await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

            // Get Example.
            Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");
            int productId = 101;

            await GetProduct(productCatalogTable, productId);
        }

        /// <summary>
        /// Retrieves information about a product from the DynamoDB table
        /// ProductCatalog based on the product ID and displays the information
        /// on the console.
        /// </summary>
        /// <param name="tableName">The name of the table from which to retrieve
        /// product information.</param>
        /// <param name="productId">The ID of the product to retrieve.</param>
        public static async Task GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = await tableName.GetItemAsync(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
```

```
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        public static async Task FindRepliesInLast15Days(
          Table table)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            // Use Query overloads that take the minimum required query
parameters.
            Search search = table.Query(filter);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15Days:
printing .....");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Retrieve replies made during a specific time period.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The subject of the thread, which we are
        /// searching for replies.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          Table table,
          string forumName,
          string threadSubject)
        {
            DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
```

```
            DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0,
0));

            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadSubject);
            filter.AddCondition("ReplyDateTime", QueryOperator.Between,
startDate, endDate);

            var config = new QueryOperationConfig()
            {
                Limit = 2, // 2 items/page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
       {
         "Message",
         "ReplyDateTime",
         "PostedBy",
       },
                ConsistentRead = true,
                Filter = filter,
            };

            Search search = table.Query(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
       }

       /// <summary>
       /// Perform a query for replies made in the last 15 days using a DynamoDB
       /// QueryOperationConfig object.
       /// </summary>
       /// <param name="table">The table we want to query.</param>
```

```
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadName">The bane of the thread that we are searching
        /// for replies.</param>
        public static async Task FindRepliesInLast15DaysWithConfig(
          Table table,
          string forumName,
          string threadName)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadName);
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            var config = new QueryOperationConfig()
            {
                Filter = filter,

                // Optional parameters.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "Message",
                  "ReplyDateTime",
                  "PostedBy",
                },
                ConsistentRead = true,
            };

            Search search = table.Query(config);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .............");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
```

```
        }

        /// <summary>
        /// Displays the contents of the passed DynamoDB document on the console.
        /// </summary>
        /// <param name="document">A DynamoDB document to display.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar um modelo de persistência de objetos de alto nível para o DynamoDB usando um AWS SDK

O exemplo de código a seguir mostra como realizar operações de criação, leitura, atualização e exclusão (CRUD) e operações em lote usando um modelo de persistência de objetos para o DynamoDB e um AWS SDK.

Para obter mais informações, consulte Modelo de persistência de objetos.

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWSCode Examples Repository.

Realize operações CRUD usando um modelo de persistência de objetos de alto nível.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
        int bookId = 1001; // Some unique value.
        Book myBook = new Book
        {
            Id = bookId,
            Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
```

```
            Isbn = "111-1111111001",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
        };

        // Save the book to the ProductCatalog table.
        await context.SaveAsync(myBook);

        // Retrieve the book from the ProductCatalog table.
        Book bookRetrieved = await context.LoadAsync<Book>(bookId);

        // Update some properties.
        bookRetrieved.Isbn = "222-2222221001";

        // Update existing authors list with the following values.
        bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
        await context.SaveAsync(bookRetrieved);

        // Retrieve the updated book. This time, add the optional
        // ConsistentRead parameter using DynamoDBContextConfig object.
        await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
        {
            ConsistentRead = true,
        });

        // Delete the book.
        await context.DeleteAsync<Book>(bookId);

        // Try to retrieve deleted book. It should return null.
        Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
        {
            ConsistentRead = true,
        });

        if (deletedBook == null)
        {
            Console.WriteLine("Book is deleted");
        }
    }
}
```

Realize operações de gravação em lote usando um modelo de persistência de objetos de alto nível.

```
/// <summary>
/// Performs high-level batch write operations to an Amazon DynamoDB table.
/// This example was written using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class HighLevelBatchWriteItem
{
    public static async Task SingleTableBatchWrite(IDynamoDBContext context)
    {
        Book book1 = new Book
        {
            Id = 902,
            InPublication = true,
            Isbn = "902-11-11-1111",
            PageCount = "100",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book3 in batch write",
        };

        Book book2 = new Book
        {
            Id = 903,
            InPublication = true,
            Isbn = "903-11-11-1111",
            PageCount = "200",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book4 in batch write",
        };

        var bookBatch = context.CreateBatchWrite<Book>();
        bookBatch.AddPutItems(new List<Book> { book1, book2 });

        Console.WriteLine("Adding two books to ProductCatalog table.");
        await bookBatch.ExecuteAsync();
    }

    public static async Task MultiTableBatchWrite(IDynamoDBContext context)
```

```
        {
            // New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0,
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
            forumBatch.AddPutItem(newForum);

            // New Thread item.
            Thread newThread = new Thread
            {
                ForumName = "S3 forum",
                Subject = "My sample question",
                KeywordTags = new List<string> { "S3", "Bucket" },
                Message = "Message text",
            };

            DynamoDBOperationConfig config = new DynamoDBOperationConfig();
            config.SkipVersionCheck = true;
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

            Console.WriteLine("Performing batch write in
MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
        }

        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

Mapeie dados arbitrários em uma tabela usando um modelo de persistência de objetos de alto nível.

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table,
retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };

            Book myBook = new Book
            {
                Id = 501,
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };

            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
```

```
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

Consulte e verifique uma tabela usando um modelo de persistência de objetos de alto nível.

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();

            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);

            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";
```

```
            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName,
threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }

        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn}
\n No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15
days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The thread object containing the query
parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string replyId = $"{forumName} #{threadSubject}";
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

            List<object> times = new List<object>();
            times.Add(twoWeeksAgoDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
            scs.Add(sc);
```

```
            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId,
cfg);
            IEnumerable<Reply> latestReplies = await
response.GetRemainingAsync();

            Console.WriteLine("\nReplies in last 15 days:");

            foreach (Reply r in latestReplies)
            {

Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries for replies posted within a specific time period.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">Information about the subject that we're
        /// interested in.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nReplies posted within time period:");

            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

            List<object> times = new List<object>();
            times.Add(startDate);
            times.Add(endDate);

            List<ScanCondition> scs = new List<ScanCondition>();
```

```
            var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId,
cfg);
            IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

            foreach (Reply r in repliesInAPeriod)
            {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries the DynamoDB ProductCatalog table for products costing less
        /// than zero.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to perform the
        /// query.</param>
        public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
        {
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
            scs.Add(sc2);

            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();
```

```
            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
        }
    }
```

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar operações de contador atômico no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como usar operações de contador atômico no DynamoDB.

- Incremente os contadores atomicamente usando as operações ADD e SET.
- Incremente seguramente contadores que talvez não existam.
- Implemente um bloqueio positivo para operações de contador.

Java

SDK para Java 2.x

Demonstre operações de contador atômico usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
```

```
import java.util.Map;

    /**
     * Increments a counter using the ADD operation.
     *
     * <p>This method demonstrates how to use the ADD operation to atomically
     * increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithAdd(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #counterName :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

 AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter using the SET operation.
     *
```

```
    * <p>This method demonstrates how to use the SET operation with an
expression
    * to increment a counter attribute.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param key The key of the item to update
    * @param counterName The name of the counter attribute
    * @param incrementValue The value to increment by
    * @return The response from DynamoDB
    * @throws DynamoDbException if an error occurs during the operation
    */
   public static UpdateItemResponse incrementCounterWithSet(
       DynamoDbClient dynamoDbClient,
       String tableName,
       Map<String, AttributeValue> key,
       String counterName,
       int incrementValue) {

       // Define the update parameters
       UpdateItemRequest request = UpdateItemRequest.builder()
           .tableName(tableName)
           .key(key)
           .updateExpression("SET #counterName = #counterName + :increment")
           .expressionAttributeNames(Map.of("#counterName", counterName))
           .expressionAttributeValues(Map.of(
               ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
           .returnValues(ReturnValue.UPDATED_NEW)
           .build();

       // Perform the update operation
       return dynamoDbClient.updateItem(request);
   }

   /**
    * Increments a counter safely, handling the case where the counter doesn't
exist yet.
    *
    * <p>This method demonstrates how to use if_not_exists to safely increment a
counter
    * that may not exist yet.
    *
```

```
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName =
if_not_exists(#counterName, :zero) + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build(),
                ":zero", AttributeValue.builder().n("0").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Decrements a counter safely, ensuring it doesn't go below zero.
     *
     * <p>This method demonstrates how to use a condition expression to safely
     * decrement a counter without going below zero.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
```

```
     * @param decrementValue The value to decrement by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation or if
the counter would go below zero
     */
    public static UpdateItemResponse decrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int decrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName - :decrement")
            .conditionExpression("#counterName >= :decrement")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":decrement",

AttributeValue.builder().n(String.valueOf(decrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Compares the ADD and SET approaches for incrementing counters.
     *
     * <p>This method demonstrates the differences between using ADD and SET
     * for incrementing counters in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return Map containing the comparison results
     */
    public static Map<String, Object> compareAddVsSet(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {
```

```
        Map<String, Object> results = new HashMap<>();

        try {
            // Reset counters to ensure a fair comparison
            UpdateItemRequest resetRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET AddCounter = :zero, SetCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .build();

            dynamoDbClient.updateItem(resetRequest);

            // Increment with ADD
            long addStartTime = System.nanoTime();
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "AddCounter", 1);
            long addEndTime = System.nanoTime();
            long addDuration = addEndTime - addStartTime;

            // Increment with SET
            long setStartTime = System.nanoTime();
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "SetCounter", 1);
            long setEndTime = System.nanoTime();
            long setDuration = setEndTime - setStartTime;

            // Record results
            results.put("addResponse", addResponse);
            results.put("setResponse", setResponse);
            results.put("addDuration", addDuration);
            results.put("setDuration", setDuration);
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }
```

```
   /**
    * Gets the current value of a counter attribute.
    *
    * <p>Helper method to retrieve the current value of a counter attribute.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param key The key of the item to get
    * @param counterName The name of the counter attribute
    * @return The counter value or null if not found
    * @throws DynamoDbException if an error occurs during the operation
    */
   public static Integer getCounterValue(
       DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String counterName) {

       // Define the get parameters
       GetItemRequest request = GetItemRequest.builder()
           .tableName(tableName)
           .key(key)
           .projectionExpression(counterName)
           .build();

       // Perform the get operation
       GetItemResponse response = dynamoDbClient.getItem(request);

       // Return the counter value if it exists, otherwise null
       if (response.item() != null && response.item().containsKey(counterName))
{
           return Integer.parseInt(response.item().get(counterName).n());
       }

       return null;
   }
```

Exemplo de uso de operações de contador atômico com o AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());
```

```
        System.out.println("Demonstrating atomic counter operations in
DynamoDB");

        try {
            // Example 1: Increment a counter using ADD
            System.out.println("\nExample 1: Incrementing a counter using ADD");
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "ViewCount", 1);

            System.out.println("Updated counter: " + addResponse.attributes());

            // Example 2: Increment a counter using SET
            System.out.println("\nExample 2: Incrementing a counter using SET");
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "LikeCount", 1);

            System.out.println("Updated counter: " + setResponse.attributes());

            // Example 3: Increment a counter safely
            System.out.println("\nExample 3: Incrementing a counter safely");
            UpdateItemResponse safeResponse =
incrementCounterSafely(dynamoDbClient, tableName, key, "ShareCount", 1);

            System.out.println("Updated counter: " + safeResponse.attributes());

            // Example 4: Decrement a counter safely
            System.out.println("\nExample 4: Decrementing a counter safely");
            try {
                UpdateItemResponse decrementResponse =
                    decrementCounterSafely(dynamoDbClient, tableName, key,
"InventoryCount", 1);

                System.out.println("Updated counter: " +
decrementResponse.attributes());
            } catch (DynamoDbException e) {
                if (e.getMessage().contains("ConditionalCheckFailed")) {
                    System.out.println("Cannot decrement counter below zero");
                } else {
                    throw e;
                }
            }

            // Example 5: Compare ADD vs SET
```

```
            System.out.println("\nExample 5: Comparing ADD vs SET");
            Map<String, Object> comparison = compareAddVsSet(dynamoDbClient,
tableName, key);

            if ((boolean) comparison.get("success")) {
                System.out.println("ADD duration: " +
comparison.get("addDuration") + " ns");
                System.out.println("SET duration: " +
comparison.get("setDuration") + " ns");
                System.out.println("ADD response: " +
comparison.get("addResponse"));
                System.out.println("SET response: " +
comparison.get("setResponse"));
            } else {
                System.out.println("Comparison failed: " +
comparison.get("error"));
            }

            // Explain atomic counter operations
            System.out.println("\nKey points about DynamoDB atomic counter
operations:");
            System.out.println("1. Both ADD and SET can be used for atomic
counters");
            System.out.println("2. ADD is more concise for simple increments");
            System.out.println("3. SET with an expression is more flexible for
complex operations");
            System.out.println("4. Use if_not_exists to handle the case where the
counter doesn't exist yet");
            System.out.println("5. Use condition expressions to prevent counters
from going below zero");
            System.out.println("6. Atomic operations are guaranteed to be
isolated from other writes");
            System.out.println("7. ADD can only be used with number and set data
types");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demonstre operações de contador atômico usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 increments.
 * While this approach works, it's less idiomatic for simple increments than
 using ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with an expression
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not
 exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  defaultValue = 0
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${counterName} =
 if_not_exists(${counterName}, :default) + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement
 optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  expectedValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
```

```
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ConditionExpression: `${counterName} = :expected`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":expected": expectedValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      data: response
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        error: "Optimistic locking failed: the counter value has changed"
      };
    }
    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not
 found
 */
```

```
async function getCounterValue(
  config,
  tableName,
  key,
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}
```

Exemplo de uso de operações de contador atômico com o AWS SDK para JavaScript.

```
/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
 DynamoDB");

  try {
```

```
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
    const response1 = await incrementCounterWithAdd(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);

    // Example 2: Using SET operation with an expression
    console.log("\nExample 2: Incrementing counter with SET operation");
    const response2 = await incrementCounterWithSet(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

    // Example 3: Safely incrementing a counter that might not exist
    console.log("\nExample 3: Safely incrementing counter that might not exist");
    const newKey = { ProductId: "P67890" };
    const response3 = await incrementCounterSafely(
      config,
      tableName,
      newKey,
      counterName,
      incrementValue,
      0
    );

    console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

    // Example 4: Incrementing with optimistic locking
    console.log("\nExample 4: Incrementing with optimistic locking");

    // First, get the current counter value
```

```
    const currentValue = await getCounterValue(config, tableName, key,
 counterName);
    console.log(`Current counter value: ${currentValue}`);

    // Then, try to increment with optimistic locking
    const response4 = await incrementCounterWithLocking(
      config,
      tableName,
      key,
      counterName,
      incrementValue,
      currentValue
    );

    if (response4.success) {
      console.log(`Counter successfully incremented to:
 ${response4.data.Attributes[counterName]}`);
    } else {
      console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex
 operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might
 not exist");
    console.log("5. Optimistic locking can be added to either approach for
 additional safety");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demonstre operações de contador atômico usando o AWS SDK para Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Union


def increment_counter_with_add(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the ADD operation.

    This function demonstrates the atomic ADD operation, which is ideal for
    incrementing counters without the risk of race conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the ADD operation to atomically increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #counter :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def increment_counter_with_set(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the SET operation with an expression.

    This function demonstrates using SET with an expression to increment a
 counter.
    While this works, it's generally recommended to use ADD for simple
 increments.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation with an expression to increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = #counter + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_safely(
```

```
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    increment_value: int = 1,
    initial_value: int = 0,
) -> Dict[str, Any]:
    """
    Increment a counter attribute safely, handling the case where it might not
 exist.

    This function demonstrates a best practice for incrementing counters by using
    the if_not_exists function to handle the case where the counter doesn't exist
 yet.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.
        initial_value (int, optional): The initial value if the counter doesn't
 exist. Defaults to 0.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use SET with if_not_exists to safely increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = if_not_exists(#counter, :initial)
 + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value, ":initial":
 initial_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def atomic_conditional_increment(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    condition_attribute: str,
    condition_value: Any,
    increment_value: int = 1,
) -> Union[Dict[str, Any], None]:
    """
    Atomically increment a counter only if a condition is met.

    This function demonstrates combining atomic counter operations with
    conditional expressions for more complex update scenarios.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Optional[Dict[str, Any]]: The response from DynamoDB if successful, None
 if condition failed.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Use ADD with a condition expression
        response = table.update_item(
            Key=key,
            UpdateExpression="ADD #counter :increment",
            ConditionExpression="#condition = :value",
            ExpressionAttributeNames={"#counter": counter_name, "#condition":
 condition_attribute},
            ExpressionAttributeValues={":increment": increment_value, ":value":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
```

```
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

Exemplo de uso de operações de contador atômico com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use the atomic counter operations functions."""
    # Example parameters
    table_name = "GameScores"
    key = {"UserId": "user123", "GameId": "game456"}
    counter_name = "Score"

    print("Example 1: Incrementing a counter with ADD operation")
    try:
        response = increment_counter_with_add(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=10
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with ADD: {e}")

    print("\nExample 2: Incrementing a counter with SET operation")
    try:
        response = increment_counter_with_set(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=5
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
```

```
        )
    except Exception as e:
        print(f"Error incrementing counter with SET: {e}")

    print("\nExample 3: Safely incrementing a counter that might not exist")
    try:
        new_key = {"UserId": "newuser789", "GameId": "game456"}
        response = increment_counter_safely(
            table_name=table_name,
            key=new_key,
            counter_name=counter_name,
            increment_value=15,
            initial_value=100,
        )
        print(
            f"Counter safely incremented. New value: {response.get('Attributes',
{}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error safely incrementing counter: {e}")

    print("\nExample 4: Conditional counter increment")
    try:
        # Fix for mypy: Handle the case where response might be None
        result = atomic_conditional_increment(
            table_name=table_name,
            key=key,
            counter_name="Achievements",
            condition_attribute="Level",
            condition_value=5,
            increment_value=1,
        )

        if result is not None:
            print(
                f"Conditional increment succeeded. New value:
{result.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
        if response:
            print(
                f"Conditional increment succeeded. New value:
{response.get('Attributes', {}).get('Achievements')}"
```

```
            )
        else:
            print("Conditional increment failed because condition was not met.")
    except Exception as e:
        print(f"Error with conditional increment: {e}")

    print("\nComparison of ADD vs SET for counter operations:")
    print("1. ADD is specifically designed for atomic numeric increments and set
operations")
    print("2. SET with an expression can be used for more complex calculations")
    print("3. Both operations are atomic, preventing race conditions")
    print("4. ADD is more concise for simple increments")
    print("5. SET with if_not_exists() is recommended when the attribute might
not exist")
    print("6. For counters, ADD is generally preferred for clarity and
simplicity")
```

- Para obter detalhes da API, consulte UpdateItem na Referência da API AWS SDK para Python (Boto3).

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar operações condicionais no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como usar operações condicionais no DynamoDB.

- Implemente gravações condicionais para evitar a substituição de dados.
- Use expressões condicionais para impor regras de negócios.
- Lide com falhas de verificação condicional com tranquilidade.

Java

SDK para Java 2.x

Demonstre operações condicionais usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Performs a conditional update on an item.
     *
     * <p>This method demonstrates how to use a condition expression to update an
 item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdate(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();
```

```
        try {
            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of(
                    "#condAttr", conditionAttribute,
                    "#updateAttr", updateAttribute))
                .expressionAttributeValues(Map.of(
                    ":condVal", conditionValue,
                    ":updateVal", updateValue))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and update was performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, update was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Performs a conditional delete on an item.
     *
```

```
    * <p>This method demonstrates how to use a condition expression to delete an
item
    * only if a specific condition is met.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param key The key of the item to delete
    * @param conditionAttribute The attribute to check in the condition
    * @param conditionValue The value to compare against
    * @return Map containing the operation result and status
    */
   public static Map<String, Object> conditionalDelete(
       DynamoDbClient dynamoDbClient,
       String tableName,
       Map<String, AttributeValue> key,
       String conditionAttribute,
       AttributeValue conditionValue) {

       Map<String, Object> result = new HashMap<>();

       try {
           // Define the delete parameters
           DeleteItemRequest request = DeleteItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .conditionExpression("#condAttr = :condVal")
               .expressionAttributeNames(Map.of("#condAttr",
conditionAttribute))
               .expressionAttributeValues(Map.of(":condVal", conditionValue))
               .returnValues(ReturnValue.ALL_OLD)
               .build();

           // Perform the delete operation
           DeleteItemResponse response = dynamoDbClient.deleteItem(request);

           // Record success result
           result.put("success", true);
           result.put("message", "Condition was met and delete was performed");
           result.put("attributes", response.attributes());

       } catch (ConditionalCheckFailedException e) {
           // Record failure due to condition not being met
           result.put("success", false);
```

```
            result.put("message", "Condition was not met, delete was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Demonstrates optimistic locking using a version attribute.
     *
     * <p>This method shows how to implement optimistic locking by using a
version
     * attribute that is incremented with each update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param versionAttribute The name of the version attribute
     * @return Map containing the operation result
     */
    public static Map<String, Object> optimisticLockingExample(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String versionAttribute) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Get the current version of the item
            GetItemRequest getRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .projectionExpression(versionAttribute)
                .build();

            GetItemResponse getResponse = dynamoDbClient.getItem(getRequest);

            // Check if the item exists
```

```
            if (getResponse.item() == null || !
getResponse.item().containsKey(versionAttribute)) {
                // Item doesn't exist or doesn't have a version attribute
                // Initialize with version 1
                UpdateItemRequest initRequest = UpdateItemRequest.builder()
                    .tableName(tableName)
                    .key(key)
                    .updateExpression("SET #verAttr = :newVer, #dataAttr
 = :data")
                    .expressionAttributeNames(Map.of("#verAttr",
 versionAttribute, "#dataAttr", "Data"))
                    .expressionAttributeValues(Map.of(
                        ":newVer", AttributeValue.builder().n("1").build(),
                        ":data", AttributeValue.builder().s("Initial
 data").build()))
                    .returnValues(ReturnValue.UPDATED_NEW)
                    .build();

                UpdateItemResponse initResponse =
 dynamoDbClient.updateItem(initRequest);

                result.put("operation", "initialize");
                result.put("success", true);
                result.put("attributes", initResponse.attributes());

                return result;
            }

            // Get the current version number
            int currentVersion =
                Integer.parseInt(getResponse.item().get(versionAttribute).n());
            int newVersion = currentVersion + 1;

            // Update the item with a condition on the version
            UpdateItemRequest updateRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #verAttr = :newVer, #dataAttr = :newData")
                .conditionExpression("#verAttr = :curVer")
                .expressionAttributeNames(Map.of("#verAttr", versionAttribute,
 "#dataAttr", "Data"))
                .expressionAttributeValues(Map.of(
                    ":curVer",
                        AttributeValue.builder()
```

```
                            .n(String.valueOf(currentVersion))
                            .build(),
                    ":newVer",

AttributeValue.builder().n(String.valueOf(newVersion)).build(),
                    ":newData",
                        AttributeValue.builder()
                            .s("Updated data at version " + newVersion)
                            .build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse updateResponse =
dynamoDbClient.updateItem(updateRequest);

            // Record success result
            result.put("operation", "update");
            result.put("success", true);
            result.put("oldVersion", currentVersion);
            result.put("newVersion", newVersion);
            result.put("attributes", updateResponse.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to version mismatch
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Version mismatch, another process may have
updated the item");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Performs a conditional update with multiple conditions.
     *
```

```
     * <p>This method demonstrates how to use multiple conditions in a condition
expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditions Map of attribute names to values for conditions
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdateWithMultipleConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        Map<String, AttributeValue> conditions,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Build the condition expression and attribute names/values
            StringBuilder conditionExpression = new StringBuilder();
            Map<String, String> expressionAttributeNames = new HashMap<>();
            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();

            // Add update attribute
            expressionAttributeNames.put("#updateAttr", updateAttribute);
            expressionAttributeValues.put(":updateVal", updateValue);

            // Add conditions
            int i = 0;
            for (Map.Entry<String, AttributeValue> condition :
conditions.entrySet()) {
                String attrName = condition.getKey();
                AttributeValue attrValue = condition.getValue();

                String nameKey = "#cond" + i;
                String valueKey = ":val" + i;

                expressionAttributeNames.put(nameKey, attrName);
                expressionAttributeValues.put(valueKey, attrValue);
```

```
                // Add AND between conditions (except for the first one)
                if (i > 0) {
                    conditionExpression.append(" AND ");
                }

                conditionExpression.append(nameKey).append(" =
").append(valueKey);
                i++;
            }

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression(conditionExpression.toString())
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "All conditions were met and update was
performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "One or more conditions were not met, update
was not performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }
```

```
        return result;
    }
```

Exemplo de uso de operações condicionais com o AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating conditional operations in DynamoDB");

        try {
            // Example 1: Conditional update
            System.out.println("\nExample 1: Conditional update");
            Map<String, Object> updateResult = conditionalUpdate(
                dynamoDbClient,
                tableName,
                key,
                "InStock",
                AttributeValue.builder().bool(true).build(),
                "Status",
                AttributeValue.builder().s("Available").build());

            System.out.println("Update result: " + updateResult.get("message"));
            if ((boolean) updateResult.get("success")) {
                System.out.println("Updated attributes: " +
updateResult.get("attributes"));
            }

            // Example 2: Conditional delete
            System.out.println("\nExample 2: Conditional delete");
            Map<String, Object> deleteResult = conditionalDelete(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Discontinued").build());

            System.out.println("Delete result: " + deleteResult.get("message"));
```

```
            if ((boolean) deleteResult.get("success")) {
                System.out.println("Deleted item: " +
deleteResult.get("attributes"));
            }

            // Example 3: Optimistic locking
            System.out.println("\nExample 3: Optimistic locking");
            Map<String, Object> lockingResult =
optimisticLockingExample(dynamoDbClient, tableName, key, "Version");

            System.out.println("Optimistic locking result:");
            System.out.println("  Operation: " + lockingResult.get("operation"));
            System.out.println("  Success: " + lockingResult.get("success"));
            if (lockingResult.get("operation").equals("update") && (boolean)
lockingResult.get("success")) {
                System.out.println("  Old version: " +
lockingResult.get("oldVersion"));
                System.out.println("  New version: " +
lockingResult.get("newVersion"));
            }
            System.out.println("  Attributes: " +
lockingResult.get("attributes"));

            // Example 4: Multiple conditions
            System.out.println("\nExample 4: Multiple conditions");
            Map<String, AttributeValue> conditions = new HashMap<>();
            conditions.put("Price",
AttributeValue.builder().n("199.99").build());
            conditions.put("Category",
AttributeValue.builder().s("Electronics").build());

            Map<String, Object> multiConditionResult =
conditionalUpdateWithMultipleConditions(
                dynamoDbClient,
                tableName,
                key,
                conditions,
                "OnSale",
                AttributeValue.builder().bool(true).build());

            System.out.println("Multiple conditions result: " +
multiConditionResult.get("message"));
            if ((boolean) multiConditionResult.get("success")) {
```

```
                System.out.println("Updated attributes: " +
multiConditionResult.get("attributes"));
            }

            // Explain conditional operations
            System.out.println("\nKey points about DynamoDB conditional
operations:");
            System.out.println("1. Conditional operations only succeed if the
condition is met");
            System.out.println("2. ConditionalCheckFailedException is thrown when
the condition fails");
            System.out.println("3. No changes are made to the item if the
condition fails");
            System.out.println("4. Conditions can be used with update, delete,
and put operations");
            System.out.println("5. Multiple conditions can be combined with AND
and OR");
            System.out.println("6. Optimistic locking can be implemented using a
version attribute");
            System.out.println(
                "7. Conditional operations consume the same amount of write
capacity whether they succeed or fail");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

SDK para JavaScript (v3)

Demonstre operações condicionais usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  DeleteCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
```

```
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Condition was met and update was performed",
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, update was not performed",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Perform a conditional delete operation.
 *
 * This function demonstrates how to delete an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to delete
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalDelete(
  config,
```

```
  tableName,
  key,
  conditionAttribute,
  conditionValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the delete parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":condition": conditionValue
    },
    ReturnValues: "ALL_OLD"
  };

  try {
    // Perform the delete operation
    const response = await docClient.send(new DeleteCommand(params));

    return {
      success: true,
      message: "Condition was met and item was deleted",
      deletedItem: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, item was not deleted",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}
```

```
/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {Object} updates - The attributes to update
 * @param {number} expectedVersion - The expected current version number
 * @returns {Promise<Object>} - Result of the operation
 */
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Update succeeded with optimistic locking",
      newVersion: expectedVersion + 1,
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Optimistic locking failed: the item was modified by another
 process",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 operation).
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
```

```
async function createIfNotExists(
  config,
  tableName,
  item
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Extract the primary key attributes
  const keyAttributes = Object.keys(item).filter(attr =>
    attr === "id" || attr === "ID" || attr === "Id" ||
    attr.endsWith("Id") || attr.endsWith("ID") ||
    attr.endsWith("Key")
  );

  if (keyAttributes.length === 0) {
    throw new Error("Could not determine primary key attributes");
  }

  // Create a condition expression that checks if the item doesn't exist
  const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

  // Define the put parameters with a condition expression
  const params = {
    TableName: tableName,
    Item: item,
    ConditionExpression: conditionExpression
  };

  try {
    // Perform the put operation
    await docClient.send(new PutCommand(params));

    return {
      success: true,
      message: "Item was created because it didn't exist",
      item
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
```

```
        message: "Item already exists, creation was skipped",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemplo de uso de operações condicionais com o AWS SDK para JavaScript.

```
/**
 * Example of how to use conditional operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
    console.log("\nExample 1: Conditional update based on attribute value");
    const updateResult = await conditionalUpdate(
      config,
      tableName,
      key,
      "Category",
      "Electronics",
      "Price",
      299.99
    );

    console.log(`Result: ${updateResult.message}`);
    if (updateResult.success) {
      console.log("Updated attributes:", updateResult.updatedAttributes);
    }

    // Example 2: Conditional delete based on attribute value
    console.log("\nExample 2: Conditional delete based on attribute value");
    const deleteResult = await conditionalDelete(
      config,
      tableName,
      key,
      "InStock",
      false
    );

    console.log(`Result: ${deleteResult.message}`);
    if (deleteResult.success) {
      console.log("Deleted item:", deleteResult.deletedItem);
```

```
    }

    // Example 3: Optimistic locking with version number
    console.log("\nExample 3: Optimistic locking with version number");

    // First, get the current item to check its version
    const currentItem = await getItem(config, tableName, { ProductId:
"P67890" });
    const currentVersion = currentItem ? (currentItem.version || 0) : 0;

    console.log(`Current version: ${currentVersion}`);

    // Then, update with optimistic locking
    const lockingResult = await updateWithOptimisticLocking(
      config,
      tableName,
      { ProductId: "P67890" },
      {
        Name: "Updated Product Name",
        Description: "This is an updated description"
      },
      currentVersion
    );

    console.log(`Result: ${lockingResult.message}`);
    if (lockingResult.success) {
      console.log(`New version: ${lockingResult.newVersion}`);
      console.log("Updated attributes:", lockingResult.updatedAttributes);
    }

    // Example 4: Create item only if it doesn't exist
    console.log("\nExample 4: Create item only if it doesn't exist");
    const createResult = await createIfNotExists(
      config,
      tableName,
      {
        ProductId: "P99999",
        Name: "New Product",
        Category: "Accessories",
        Price: 19.99,
        InStock: true
      }
    );
```

```
    console.log(`Result: ${createResult.message}`);
    if (createResult.success) {
      console.log("Created item:", createResult.item);
    }

    // Explain conditional operations
    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is
 met");
    console.log("2. ConditionalCheckFailedException indicates the condition
 wasn't met");
    console.log("3. Optimistic locking prevents race conditions in concurrent
 updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for
 checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed
 completely or fail completely");
    console.log("6. You can use any valid comparison operators and functions in
 condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the
 condition fails");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - DeleteItem
  - PutItem
  - UpdateItem

Python

SDK para Python (Boto3)

Demonstre operações condicionais usando o AWS SDK para Python (Boto3).

```
import boto3
```

```
from botocore.exceptions import ClientError
from typing import Any, Dict, Optional, Tuple, Union


def conditional_update(
    table_name: str,
    key: Dict[str, Any],
    condition_attribute: str,
    condition_value: Any,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item only if a condition is met.

    This function demonstrates how to perform a conditional update operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression="SET #update_attr = :update_val",
            ConditionExpression="#cond_attr = :cond_val",
            ExpressionAttributeNames={
                "#update_attr": update_attribute,
                "#cond_attr": condition_attribute,
```

```
            },
            ExpressionAttributeValues={":update_val": update_value, ":cond_val":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
        # Update succeeded, condition was met
        return True, response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False, None
        else:
            # Other error occurred
            raise


def conditional_delete(
    table_name: str, key: Dict[str, Any], condition_attribute: str,
 condition_value: Any
) -> bool:
    """
    Delete an item only if a condition is met.

    This function demonstrates how to perform a conditional delete operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to delete.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.

    Returns:
        bool: True if the delete succeeded (condition was met), False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional delete
        table.delete_item(
            Key=key,
            ConditionExpression="#attr = :val",
```

```
            ExpressionAttributeNames={"#attr": condition_attribute},
            ExpressionAttributeValues={":val": condition_value},
        )
        # Delete succeeded, condition was met
        return True
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False
        else:
            # Other error occurred
            raise


def optimistic_locking_update(
    table_name: str,
    key: Dict[str, Any],
    version_attribute: str,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item using optimistic locking with a version attribute.

    This function demonstrates how to implement optimistic locking using
    a version attribute that is incremented with each update.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        version_attribute (str): The name of the version attribute.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # First, get the current version
```

```
    try:
        response = table.get_item(
            Key=key,
            ProjectionExpression=f"#{version_attribute}",
            ExpressionAttributeNames={f"#{version_attribute}":
 version_attribute},
        )

        item = response.get("Item", {})
        current_version = item.get(version_attribute, 0)

        # Now, try to update with a condition on the version
        try:
            update_response = table.update_item(
                Key=key,
                UpdateExpression=f"SET #{update_attribute} = :update_val,
 #{version_attribute} = :new_version",
                ConditionExpression=f"#{version_attribute} = :current_version",
                ExpressionAttributeNames={
                    f"#{update_attribute}": update_attribute,
                    f"#{version_attribute}": version_attribute,
                },
                ExpressionAttributeValues={
                    ":update_val": update_value,
                    ":current_version": current_version,
                    ":new_version": current_version + 1,
                },
                ReturnValues="UPDATED_NEW",
            )
            # Update succeeded
            return True, update_response
        except ClientError as e:
            if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
                # Version has changed, optimistic locking failed
                return False, None
            else:
                # Other error occurred
                raise
    except ClientError:
        # Error getting the item
        raise


def conditional_check_and_update(
```

```
    table_name: str,
    key: Dict[str, Any],
    check_attribute: str,
    check_value: Any,
    update_attribute: str,
    update_value: Any,
    create_if_not_exists: bool = False,
) -> Union[Dict[str, Any], None]:
    """
    Check if an attribute has a specific value and update another attribute if it
 does.

    This function demonstrates a more complex conditional update that can also
    create the item if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        check_attribute (str): The attribute to check in the condition.
        check_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.
        create_if_not_exists (bool, optional): Whether to create the item if it
 doesn't exist.

    Returns:
        Union[Dict[str, Any], None]: The response from DynamoDB if successful,
 None otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        if create_if_not_exists:
            # Use attribute_not_exists to create the item if it doesn't exist
            condition_expression = "attribute_not_exists(#pk) OR #check_attr
 = :check_val"
            update_expression = "SET #update_attr = :update_val, #check_attr =
 if_not_exists(#check_attr, :check_val)"

            # Get the partition key name from the key dictionary
            pk_name = next(iter(key))
```

```
            expression_attribute_names = {
                "#pk": pk_name,
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }
        else:
            # Only update if the check attribute has the expected value
            condition_expression = "#check_attr = :check_val"
            update_expression = "SET #update_attr = :update_val"

            expression_attribute_names = {
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }

        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues={":check_val": check_value, ":update_val":
 update_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

Exemplo de uso de operações condicionais com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use the conditional operations functions."""
    # Example parameters
    table_name = "Products"
```

```
    key = {"ProductId": "prod123"}

    print("Example 1: Conditional Update")
    try:
        # Update the price only if the current stock is greater than 10
        success, response = conditional_update(
            table_name=table_name,
            key=key,
            condition_attribute="Stock",
            condition_value=10,
            update_attribute="Price",
            update_value=99.99,
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Update succeeded! New values: {attributes}")
        else:
            print("Update failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional update: {e}")

    print("\nExample 2: Conditional Delete")
    try:
        # Delete the product only if it's discontinued
        success = conditional_delete(
            table_name=table_name,
            key=key,
            condition_attribute="Status",
            condition_value="Discontinued",
        )

        if success:
            print("Delete succeeded! The item was deleted.")
        else:
            print("Delete failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional delete: {e}")

    print("\nExample 3: Optimistic Locking")
    try:
        # Update with optimistic locking using a version attribute
```

```
        success, response = optimistic_locking_update(
            table_name=table_name,
            key=key,
            version_attribute="Version",
            update_attribute="Description",
            update_value="Updated product description",
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Optimistic locking update succeeded! New values:
{attributes}")
        else:
            print("Optimistic locking update failed because the version has
changed.")
    except Exception as e:
        print(f"Error during optimistic locking update: {e}")

    print("\nExample 4: Conditional Check and Update")
    try:
        # Update the featured status if the product is in stock
        response = conditional_check_and_update(
            table_name=table_name,
            key=key,
            check_attribute="InStock",
            check_value=True,
            update_attribute="Featured",
            update_value=True,
            create_if_not_exists=True,
        )

        if response:
            print(
                f"Conditional check and update succeeded! New values:
{response.get('Attributes', {})}"
            )
        else:
            print("Conditional check and update failed because the condition was
not met.")
    except Exception as e:
        print(f"Error during conditional check and update: {e}")
```

```
    print("\nUnderstanding Conditional Operations in DynamoDB:")
    print("1. Conditional operations help maintain data integrity")
    print("2. They prevent race conditions in concurrent environments")
    print("3. Failed conditions result in ConditionalCheckFailedException")
    print("4. No DynamoDB capacity is consumed when conditions fail")
    print("5. Optimistic locking is a common pattern using version attributes")
    print("6. Conditions can be combined with logical operators (AND, OR, NOT)")
    print("7. Conditions can use comparison operators (=, <>, <, <=, >, >=)")
    print(
        "8. attribute_exists() and attribute_not_exists() are useful for checking
attribute presence"
    )
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de API do AWS SDK para Python (Boto3).
  - DeleteItem
  - PutItem
  - UpdateItem

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar nomes de atributo de expressão no DynamoDB com um SDK da AWS

Os exemplos de código a seguir mostram como usar nomes de atributo de expressão no DynamoDB.

- Trabalhe com palavras reservadas nas expressões do DynamoDB.
- Use espaços reservados para nomes de atributo de expressão.
- Manipule caracteres especiais em nomes de atributo.

Java

SDK para Java 2.x

Demonstre nomes de atributo de expressão usando o AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates an attribute that is a reserved word in DynamoDB.
     *
     * <p>This method demonstrates how to use expression attribute names to
 update
     * attributes that are reserved words in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param reservedWordAttribute The reserved word attribute to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", reservedWordAttribute))
```

```
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates an attribute that contains special characters.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * attributes that contain special characters.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param specialCharAttribute The attribute with special characters to
update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateSpecialCharacterAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String specialCharAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", specialCharAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }
```

```
   /**
    * Queries items using an attribute that is a reserved word.
    *
    * <p>This method demonstrates how to use expression attribute names in a
query
    * when the attribute is a reserved word.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param partitionKeyName The name of the partition key attribute
    * @param partitionKeyValue The value of the partition key
    * @param reservedWordAttribute The reserved word attribute to filter on
    * @param value The value to compare against
    * @return The response from DynamoDB
    * @throws DynamoDbException if an error occurs during the operation
    */
   public static QueryResponse queryWithReservedWordAttribute(
       DynamoDbClient dynamoDbClient,
       String tableName,
       String partitionKeyName,
       AttributeValue partitionKeyValue,
       String reservedWordAttribute,
       AttributeValue value) {

       // Define the query parameters using expression attribute names
       Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put("#pkName", partitionKeyName);
       expressionAttributeNames.put("#attr", reservedWordAttribute);

       Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
       expressionAttributeValues.put(":pkValue", partitionKeyValue);
       expressionAttributeValues.put(":value", value);

       QueryRequest request = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression("#pkName = :pkValue")
           .filterExpression("#attr = :value")
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .build();

       // Perform the query operation
       return dynamoDbClient.query(request);
```

```
    }

    /**
     * Updates a nested attribute with a path that contains reserved words.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * nested attributes where the path contains reserved words.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributePath The path to the nested attribute as an array
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateNestedReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> attributePath,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < attributePath.size(); i++) {
            expressionAttributeNames.put("#attr" + i, attributePath.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < attributePath.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
```

```
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Scans a table with multiple attribute name placeholders.
     *
     * <p>This method demonstrates how to use multiple expression attribute names
     * in a complex filter expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param filters Object mapping attribute names to filter values
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithMultipleAttributeNames(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> filters) {

        // Create expression attribute names and values
        Map<String, String> expressionAttributeNames = new HashMap<>();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        StringBuilder filterExpression = new StringBuilder();

        // Build the filter expression
        int index = 0;
        for (Map.Entry<String, AttributeValue> entry : filters.entrySet()) {
            String attrName = entry.getKey();
            AttributeValue attrValue = entry.getValue();

            String nameKey = "#attr" + index;
            String valueKey = ":val" + index;

            expressionAttributeNames.put(nameKey, attrName);
            expressionAttributeValues.put(valueKey, attrValue);
```

```
            // Add AND between conditions (except for the first one)
            if (index > 0) {
                filterExpression.append(" AND ");
            }

            filterExpression.append(nameKey).append(" = ").append(valueKey);
            index++;
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression(filterExpression.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }
```

Exemplo de uso de nomes de atributo de expressão com o AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating expression attribute names in
DynamoDB");

        try {
            // Example 1: Update an attribute that is a reserved word
            System.out.println("\nExample 1: Updating an attribute that is a
reserved word");
            UpdateItemResponse response1 = updateReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Size", // "SIZE" is a reserved word in DynamoDB
                AttributeValue.builder().s("Large").build());
```

```
            System.out.println("Updated attribute: " + response1.attributes());

            // Example 2: Update an attribute with special characters
            System.out.println("\nExample 2: Updating an attribute with special
characters");
            UpdateItemResponse response2 = updateSpecialCharacterAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Product-Type", // Contains a hyphen, which is a special
character
                AttributeValue.builder().s("Electronics").build());

            System.out.println("Updated attribute: " + response2.attributes());

            // Example 3: Query with a reserved word attribute
            System.out.println("\nExample 3: Querying with a reserved word
attribute");
            QueryResponse response3 = queryWithReservedWordAttribute(
                dynamoDbClient,
                tableName,
                "Category",
                AttributeValue.builder().s("Electronics").build(),
                "Count", // "COUNT" is a reserved word in DynamoDB
                AttributeValue.builder().n("10").build());

            System.out.println("Found " + response3.count() + " items");

            // Example 4: Update a nested attribute with reserved words in the
path
            System.out.println("\nExample 4: Updating a nested attribute with
reserved words in the path");
            UpdateItemResponse response4 = updateNestedReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                Arrays.asList("Dimensions", "Size", "Height"), // "SIZE" is a
reserved word
                AttributeValue.builder().n("30").build());

            System.out.println("Updated nested attribute: " +
response4.attributes());
```

```
            // Example 5: Scan with multiple attribute name placeholders
            System.out.println("\nExample 5: Scanning with multiple attribute
 name placeholders");
            Map<String, AttributeValue> filters = new HashMap<>();
            filters.put("Size", AttributeValue.builder().s("Large").build());
            filters.put("Count", AttributeValue.builder().n("10").build());
            filters.put(
                "Product-Type",
 AttributeValue.builder().s("Electronics").build());

            ScanResponse response5 =
 scanWithMultipleAttributeNames(dynamoDbClient, tableName, filters);

            System.out.println("Found " + response5.count() + " items");

            // Show some common reserved words
            System.out.println("\nSome common DynamoDB reserved words:");
            List<String> commonReservedWords = getDynamoDBReservedWords();
            System.out.println(String.join(", ", commonReservedWords));

            // Explain expression attribute names
            System.out.println("\nKey points about expression attribute names:");
            System.out.println("1. Use expression attribute names (#name) for
 reserved words");
            System.out.println("2. Use expression attribute names for attributes
 with special characters");
            System.out.println(
                "3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
            System.out.println("4. Expression attribute names are required for
 nested attributes with reserved words");
            System.out.println("5. You can use multiple expression attribute
 names in a single expression");
            System.out.println("6. Expression attribute names are case-
sensitive");
            System.out.println("7. Expression attribute names are only used in
 expressions, not in the actual data");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.
  - Consulta
  - UpdateItem

JavaScript

SDK para JavaScript (v3)

Demonstre nomes de atributo de expressão usando o AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
  value
) {
```

```
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters
 to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  specialCharAttribute,
  value
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter
 on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
  tableName,
  partitionKeyName,
```

```
  partitionKeyValue,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));

  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
  config,
```

```
  tableName,
  key,
  attributePath,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath
    .map((_, i) => `#attr${i}`)
    .join(".");

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
  config,
  tableName,
  filters
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names and values
  const expressionAttributeNames = {};
  const expressionAttributeValues = {};
  const filterConditions = [];

  // Build the filter expression
  Object.entries(filters).forEach(([attrName, value], index) => {
    const nameKey = `#attr${index}`;
    const valueKey = `:val${index}`;

    expressionAttributeNames[nameKey] = attrName;
    expressionAttributeValues[valueKey] = value;
    filterConditions.push(`${nameKey} = ${valueKey}`);
  });

  // Join the filter conditions with AND
  const filterExpression = filterConditions.join(" AND ");

  // Define the scan parameters
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Perform the scan operation
  const response = await docClient.send(new ScanCommand(params));

  return response;
```

```
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemplo de uso de nomes de atributo de expressão com o AWS SDK para JavaScript.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
```

```
const tableName = "Products";
const key = { ProductId: "P12345" };

console.log("Demonstrating expression attribute names in DynamoDB");

try {
  // Example 1: Update an attribute that is a reserved word
  console.log("\nExample 1: Updating an attribute that is a reserved word");
  const response1 = await updateReservedWordAttribute(
    config,
    tableName,
    key,
    "Size", // "SIZE" is a reserved word in DynamoDB
    "Large"
  );

  console.log("Updated attribute:", response1.Attributes);

  // Example 2: Update an attribute with special characters
  console.log("\nExample 2: Updating an attribute with special characters");
  const response2 = await updateSpecialCharacterAttribute(
    config,
    tableName,
    key,
    "Product-Type", // Contains a hyphen, which is a special character
    "Electronics"
  );

  console.log("Updated attribute:", response2.Attributes);

  // Example 3: Query with a reserved word attribute
  console.log("\nExample 3: Querying with a reserved word attribute");
  const response3 = await queryWithReservedWordAttribute(
    config,
    tableName,
    "Category",
    "Electronics",
    "Count", // "COUNT" is a reserved word in DynamoDB
    10
  );

  console.log(`Found ${response3.Items.length} items`);

  // Example 4: Update a nested attribute with reserved words in the path
```

```
    console.log("\nExample 4: Updating a nested attribute with reserved words in
the path");
    const response4 = await updateNestedReservedWordAttribute(
      config,
      tableName,
      key,
      ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
      30
    );

    console.log("Updated nested attribute:", response4.Attributes);

    // Example 5: Scan with multiple attribute name placeholders
    console.log("\nExample 5: Scanning with multiple attribute name
placeholders");
    const response5 = await scanWithMultipleAttributeNames(
      config,
      tableName,
      {
        "Size": "Large",
        "Count": 10,
        "Product-Type": "Electronics"
      }
    );

    console.log(`Found ${response5.Items.length} items`);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Show some common reserved words
    console.log("\nSome common DynamoDB reserved words:");
    const commonReservedWords = [
      "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
      "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
      "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
      "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
      "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN",
"NAME",
      "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
      "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
      "VIEW", "WHERE"
```

```
    ];
    console.log(commonReservedWords.join(", "));

    // Explain expression attribute names
    console.log("\nKey points about expression attribute names:");
    console.log("1. Use expression attribute names (#name) for reserved words");
    console.log("2. Use expression attribute names for attributes with special
 characters");
    console.log("3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
    console.log("4. Expression attribute names are required for nested attributes
 with reserved words");
    console.log("5. You can use multiple expression attribute names in a single
 expression");
    console.log("6. Expression attribute names are case-sensitive");
    console.log("7. Expression attribute names are only used in expressions, not
 in the actual data");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para JavaScript.
  - Consulta
  - UpdateItem

Python

SDK para Python (Boto3)

Demonstre nomes de atributo de expressão usando o AWS SDK para Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List


def use_reserved_word_attribute(
    table_name: str, key: Dict[str, Any], reserved_word: str, value: Any
```

```
) -> Dict[str, Any]:
    """
    Update an attribute whose name is a DynamoDB reserved word.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names that are DynamoDB reserved words.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        reserved_word (str): The reserved word to use as an attribute name.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle the reserved word
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #reserved_attr = :value",
        ExpressionAttributeNames={"#reserved_attr": reserved_word},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def use_special_character_attribute(
    table_name: str, key: Dict[str, Any], attribute_with_special_chars: str,
 value: Any
) -> Dict[str, Any]:
    """
    Update an attribute whose name contains special characters.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names containing special characters like spaces, dots,
 or hyphens.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        attribute_with_special_chars (str): The attribute name with special
 characters.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle special characters
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #special_attr = :value",
        ExpressionAttributeNames={"#special_attr": attribute_with_special_chars},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def query_with_attribute_names(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    filter_attribute_name: str,
    filter_value: Any,
) -> Dict[str, Any]:
    """
    Query a table using expression attribute names for both key and filter
 attributes.

    This function demonstrates how to use expression attribute names in a query
 operation
    for both the key condition expression and filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute_name (str): The name of the attribute to filter on.
        filter_value (Any): The value to compare against in the filter.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names for both key condition and filter
    response = table.query(
        KeyConditionExpression="#pk = :pk_val",
        FilterExpression="#filter_attr = :filter_val",
        ExpressionAttributeNames={"#pk": partition_key_name, "#filter_attr":
 filter_attribute_name},
        ExpressionAttributeValues={":pk_val": partition_key_value, ":filter_val":
 filter_value},
    )

    return response


def update_nested_attribute_with_dots(
    table_name: str, key: Dict[str, Any], path_with_dots: str, value: Any
) -> Dict[str, Any]:
    """
    Update a nested attribute using a path with dot notation.

    This function demonstrates how to use expression attribute names to work with
    nested attributes specified using dot notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path_with_dots (str): The path to the nested attribute using dot notation
 (e.g., "a.b.c").
        value (Any): The value to set for the nested attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
```

```
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path_with_dots.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def demonstrate_attribute_name_requirements(table_name: str, key: Dict[str, Any])
 -> Dict[str, Any]:
    """
    Demonstrate the requirements and allowed characters for attribute names.

    This function shows examples of valid and invalid attribute names and how to
```

```
    handle them using expression attribute names.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Examples of attribute names with different characteristics
    examples = {
        "valid_standard": "NormalAttribute",  # Standard attribute name (no
placeholder needed)
        "valid_with_underscore": "Normal_Attribute",  # Underscore is allowed
        "valid_with_number": "Attribute123",  # Numbers are allowed
        "reserved_word": "Timestamp",  # Reserved word (requires placeholder)
        "starts_with_number": "123Attribute",  # Starts with number (valid but
may need placeholder in some contexts)
        "with_space": "Attribute Name",  # Contains space (requires placeholder)
        "with_dot": "Attribute.Name",  # Contains dot (requires placeholder)
        "with_hyphen": "Attribute-Name",  # Contains hyphen (requires
placeholder)
        "with_special_chars": "Attribute#$%",  # Contains special characters
(requires placeholder)
    }

    results = {}

    # Try to update each attribute type
    for example_type, attr_name in examples.items():
        try:
            # For attributes that don't need placeholders, try direct reference
            if example_type in ["valid_standard", "valid_with_underscore",
"valid_with_number"]:
                try:
                    # Try without expression attribute names first
                    response = table.update_item(
                        Key=key,
                        UpdateExpression=f"SET {attr_name} = :value",
```

```
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": False,
                        "response": response,
                    }
                except ClientError:
                    # If direct reference fails, try with placeholder
                    response = table.update_item(
                        Key=key,
                        UpdateExpression="SET #attr = :value",
                        ExpressionAttributeNames={"#attr": attr_name},
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": True,
                        "response": response,
                    }
            else:
                # For attributes that definitely need placeholders
                response = table.update_item(
                    Key=key,
                    UpdateExpression="SET #attr = :value",
                    ExpressionAttributeNames={"#attr": attr_name},
                    ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                    ReturnValues="UPDATED_NEW",
                )
                results[example_type] = {
                    "attribute_name": attr_name,
                    "success": True,
                    "needed_placeholder": True,
                    "response": response,
                }
        except ClientError as e:
```

```
            results[example_type] = {"attribute_name": attr_name, "success":
 False, "error": str(e)}

    return results
```

Exemplo de uso de nomes de atributo de expressão com o AWS SDK para Python (Boto3).

```
def example_usage():
    """Example of how to use expression attribute names in DynamoDB."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Using a reserved word as an attribute name")
    try:
        response = use_reserved_word_attribute(
            table_name=table_name, key=key, reserved_word="Timestamp",
 value="2025-05-14T12:00:00Z"
        )
        print(f"Reserved word attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating reserved word attribute: {e}")

    print("\nExample 2: Using an attribute name with special characters")
    try:
        response = use_special_character_attribute(
            table_name=table_name,
            key=key,
            attribute_with_special_chars="Product Info",
            value="Special product information",
        )
        print(f"Special character attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating special character attribute: {e}")

    print("\nExample 3: Querying with expression attribute names")
    try:
        response = query_with_attribute_names(
```

```
            table_name=table_name,
            partition_key_name="Category",
            partition_key_value="Electronics",
            filter_attribute_name="Price",
            filter_value=500,
        )
        print(
            f"Query with expression attribute names returned
{len(response.get('Items', []))} items"
        )
    except Exception as e:
        print(f"Error querying with expression attribute names: {e}")

    print("\nExample 4: Updating a nested attribute with dot notation")
    try:
        response = update_nested_attribute_with_dots(
            table_name=table_name,
            key=key,
            path_with_dots="Product.Details.Specifications",
            value={"Weight": "2.5 kg", "Dimensions": "30x20x10 cm"},
        )
        print(f"Nested attribute updated successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating nested attribute: {e}")

    print("\nExample 5: Demonstrating attribute name requirements")
    try:
        results = demonstrate_attribute_name_requirements(table_name=table_name,
key=key)

        print("Attribute Name Requirements Results:")
        for example_type, result in results.items():
            if result.get("success", False):
                needed_placeholder = result.get("needed_placeholder", True)
                print(
                    f"  - {example_type}: '{result['attribute_name']}' -
{'Requires' if needed_placeholder else 'Does not require'} placeholder"
                )
            else:
                print(
                    f"  - {example_type}: '{result['attribute_name']}' - Failed:
{result.get('error', 'Unknown error')}"
                )
```

```
    except Exception as e:
        print(f"Error demonstrating attribute name requirements: {e}")

    print("\nCommon DynamoDB Reserved Words (sample):")
    reserved_words = get_common_reserved_words()
    print(", ".join(reserved_words[:20]) + "... (and many more)")

    print("\nWhen to Use Expression Attribute Names:")
    print("1. When the attribute name is a DynamoDB reserved word")
    print("2. When the attribute name contains special characters (spaces, dots,
hyphens)")
    print("3. When the attribute name begins with a number")
    print("4. When working with nested attributes using dot notation")
    print("5. When you need to reference the same attribute multiple times in an
expression")

    print("\nExpression Attribute Name Requirements:")
    print("1. Must begin with a pound sign (#)")
    print("2. After the pound sign, must contain at least one character")
    print("3. Can contain alphanumeric characters and underscore (_)")
    print("4. Are case-sensitive")
    print("5. Must be unique within a single expression")

    print("\nAttribute Name Requirements in DynamoDB:")
    print("1. Can begin with a-z, A-Z, or 0-9")
    print("2. Can contain a-z, A-Z, 0-9, underscore (_), dash (-), and dot (.)")
    print("3. Are case-sensitive")
    print("4. No length restrictions, but practical limits apply")
    print("5. Cannot be a DynamoDB reserved word if used directly in
expressions")
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de API do AWS SDK para Python (Boto3).
  - Consulta
  - UpdateItem

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar eventos programados para invocar uma função do Lambda

Os exemplos de código a seguir mostram como criar uma função do AWS Lambda invocada por um evento programado do Amazon EventBridge.

Java

SDK para Java 2.x

Mostra como criar um evento programado do Amazon EventBridge que invoca uma função do AWS Lambda. Configure o EventBridge para usar uma expressão cron para programar o momento em que a função do Lambda é invocada. Neste exemplo, você cria uma função do Lambda usando a API de runtime de Java do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma aplicação que envia uma mensagem de texto móvel para seus funcionários que os parabeniza na data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços utilizados neste exemplo

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Mostra como criar um evento programado do Amazon EventBridge que invoca uma função do AWS Lambda. Configure o EventBridge para usar uma expressão cron para programar o momento em que a função do Lambda é invocada. Neste exemplo, você cria uma função do

Lambda usando a API de runtime de JavaScript do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma aplicação que envia uma mensagem de texto móvel para seus funcionários que os parabeniza na data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK para JavaScript v3.

Serviços utilizados neste exemplo

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK para Python (Boto3).

Este exemplo mostra como registrar uma função do AWS Lambda como o destino de um evento do Amazon EventBridge programado. O handler do Lambda grava uma mensagem amigável e os dados completos do evento no Amazon CloudWatch Logs para recuperação posterior.

- Implanta uma função do Lambda.
- Cria um evento agendado do EventBridge e faz da função do Lambda o destino.
- Concede permissão para que o EventBridge invoque uma função do Lambda.
- Imprime os dados mais recentes do CloudWatch Logs para mostrar o resultado das invocações agendadas.
- Limpa todos os recursos criados durante a demonstração.

Este exemplo é mais bem visualizado no GitHub. Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços utilizados neste exemplo

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Trabalhar com índices secundários locais do DynamoDB usando a AWS Command Line Interface v2

O exemplo de código a seguir mostra como criar e consultar tabelas com índices secundários locais.

- Crie uma tabela com um índice secundário local (LSI).
- Crie uma tabela com vários LSIs com diferentes tipos de projeção.
- Consulte dados usando LSIs.

Bash

AWS CLI com script Bash

Crie uma tabela com um índice secundário local.

```
# Create a table with a Local Secondary Index
aws dynamodb create-table \
    --table-name CustomerOrders \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=OrderID,AttributeType=S \
        AttributeName=OrderDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=OrderID,KeyType=RANGE \
    --local-secondary-indexes \
```

```
        "IndexName=OrderDateIndex,\
        KeySchema=[{AttributeName=CustomerID,KeyType=HASH},
{AttributeName=OrderDate,KeyType=RANGE}],\
        Projection={ProjectionType=ALL}" \
    --billing-mode PAY_PER_REQUEST
```

Crie uma tabela com vários LSIs.

```
# Create a table with multiple Local Secondary Indexes
aws dynamodb create-table \
    --table-name CustomerDetails \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=Name,AttributeType=S \
        AttributeName=Email,AttributeType=S \
        AttributeName=RegistrationDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=Name,KeyType=RANGE \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"EmailIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Email\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Address\",\"Phone\"]}
            },
            {
                \"IndexName\": \"RegistrationIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"RegistrationDate\",\"KeyType\":\"RANGE
\"}
                ],
                \"Projection\": {\"ProjectionType\":\"KEYS_ONLY\"}
            }
        ]" \
    --billing-mode PAY_PER_REQUEST
```

Consulte dados usando LSIs.

```
# Query the OrderDateIndex LSI
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId AND OrderDate BETWEEN :date1
 AND :date2" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":date1": {"S": "2023-01-01"},
        ":date2": {"S": "2023-02-01"}
    }'

# Query with a filter expression
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId" \
    --filter-expression "Amount > :amount" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":amount": {"N": "150"}
    }'
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable
  - Consulta

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Trabalhar com fluxos e vida útil do DynamoDB usando a AWS Command Line Interface v2

O exemplo de código apresentado a seguir demonstra como gerenciar os recursos de fluxos e de vida útil do DynamoDB.

- Crie uma tabela com a opção de fluxos habilitada.
- Descreva os fluxos.
- Crie uma função do Lambda para processar os fluxos.
- Habilite a vida útil (TTL) em uma tabela.
- Adicione itens com atributos de TTL.
- Descreva as configurações de TTL.

Bash

AWS CLI com script Bash

Crie uma tabela com a opção de fluxos habilitada.

```
# Create a table with DynamoDB Streams enabled
aws dynamodb create-table \
    --table-name StreamsDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

Descreva os fluxos.

```
# Get information about the stream
aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.StreamSpecification"

# Get the stream ARN
STREAM_ARN=$(aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.LatestStreamArn" \
    --output text)

echo "Stream ARN: $STREAM_ARN"

# Describe the stream
```

```
aws dynamodbstreams describe-stream \
    --stream-arn $STREAM_ARN
```

Crie uma função do Lambda para os fluxos.

```
# Step 1: Create an IAM role for the Lambda function
cat > trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

aws iam create-role \
    --role-name DynamoDBStreamsLambdaRole \
    --assume-role-policy-document file://trust-policy.json

# Step 2: Attach permissions to the role
aws iam attach-role-policy \
    --role-name DynamoDBStreamsLambdaRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaDynamoDBExecutionRole

# Step 3: Create a Lambda function (code would be in a separate file)
echo "Lambda function creation would be done separately with appropriate code"

# Step 4: Create an event source mapping
echo "Example command to create event source mapping:"
echo "aws lambda create-event-source-mapping \\"
echo "    --function-name ProcessDynamoDBRecords \\"
echo "    --event-source $STREAM_ARN \\"
echo "    --batch-size 100 \\"
echo "    --starting-position LATEST"
```

Habilite a vida útil (TTL) em uma tabela.

```
# Create a table for TTL demonstration
aws dynamodb create-table \
    --table-name TTLDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST

# Wait for table to become active
aws dynamodb wait table-exists --table-name TTLDemo

# Enable TTL on the table
aws dynamodb update-time-to-live \
    --table-name TTLDemo \
    --time-to-live-specification "Enabled=true, AttributeName=ExpirationTime"
```

Adicione itens com atributos de TTL.

```
# Calculate expiration time (current time + 1 day in seconds)
EXPIRATION_TIME=$(date -d "+1 day" +%s)

# Add an item with TTL attribute
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item1"},
        "Data": {"S": "This item will expire in 1 day"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME'"}
    }'

# Add an item that expires in 1 hour
EXPIRATION_TIME_HOUR=$(date -d "+1 hour" +%s)
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item2"},
        "Data": {"S": "This item will expire in 1 hour"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME_HOUR'"}
    }'
```

Descreva as configurações de TTL.

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Trabalhar com tabelas globais do DynamoDB e a consistência final de replicação mutirregional (MREC) usando AWS Command Line Interface v2

O exemplo de código a seguir mostra como gerenciar tabelas globais do DynamoDB com consistência final de replicação mutirregional (MREC).

- Crie uma tabela com replicação mutirregional (MREC).
- Insira e obtenha itens de tabelas-réplica.
- Remova as réplicas uma por uma.
- Exclua a tabela para fazer uma limpeza.

Bash

AWS CLI com script Bash

Crie uma tabela com replicação mutirregional.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
```

```
  ]
}' \
--region us-east-2
```

Descreva a tabela multirregional.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Insira itens em uma tabela-réplica.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Obtenha itens de tabelas-réplica.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Remova as réplicas.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Exclua a tabela para fazer uma limpeza.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable

- DeleteTable
- DescribeTable
- GetItem
- PutItem
- UpdateTable

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Trabalhar com a marcação de recursos do DynamoDB usando a AWS Command Line Interface v2

O exemplo de código a seguir mostra como gerenciar tags para recursos do DynamoDB.

- Crie uma tabela com tags.
- Listar as etiquetas de um recurso.
- Adicione tags a um recurso.
- Remova as tags de um recurso.
- Filtre tabelas por tags.

Bash

AWS CLI com script Bash

Crie uma tabela com tags.

```
# Create a table with tags
aws dynamodb create-table \
    --table-name TaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
```

```
        Key=Environment,Value=Production \
        Key=Project,Value=Analytics \
        Key=Owner,Value=DataTeam
```

Listar as etiquetas de um recurso.

```
# Get the table ARN
TABLE_ARN=$(aws dynamodb describe-table \
    --table-name TaggedTable \
    --query "Table.TableArn" \
    --output text)

# List tags for the table
aws dynamodb list-tags-of-resource \
    --resource-arn $TABLE_ARN
```

Adicione tags a um recurso.

```
# Add tags to an existing table
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

Remova as tags de um recurso.

```
# Remove tags from a table
aws dynamodb untag-resource \
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

Filtre tabelas por tags.

```
# Create another table with different tags
aws dynamodb create-table \
    --table-name AnotherTaggedTable \
```

```
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Environment,Value=Development \
        Key=Project,Value=Testing

# Wait for table to become active
aws dynamodb wait table-exists --table-name AnotherTaggedTable

# List all tables
echo "All tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nFiltering tables by Environment=Production tag:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# Find tables with specific tag
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateTable
  - ListTagsOfResource
  - TagResource
  - UntagResource

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Trabalhar com a criptografia de tabelas do DynamoDB usando a AWS Command Line Interface v2

O exemplo de código a seguir mostra como gerenciar opções de criptografia para tabelas do DynamoDB.

- Crie uma tabela com a criptografia padrão.
- Crie uma tabela com uma chave gerenciada pelo cliente (CMK).
- Atualize as configurações de criptografia da tabela.
- Descreva a criptografia da tabela.

Bash

AWS CLI com script Bash

Crie uma tabela com a criptografia padrão.

```
# Create a table with default encryption (AWS owned key)
aws dynamodb create-table \
    --table-name CustomerData \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS
```

Crie uma tabela com uma chave gerenciada pelo cliente (CMK).

```
# Step 1: Create a customer managed key in KMS
aws kms create-key \
    --description "Key for DynamoDB table encryption" \
    --key-usage ENCRYPT_DECRYPT \
    --customer-master-key-spec SYMMETRIC_DEFAULT
```

```
# Store the key ID for later use
KEY_ID=$(aws kms list-keys --query "Keys[?contains(KeyArn, 'Key for
 DynamoDB')].KeyId" --output text)

# Step 2: Create a table with the customer managed key
aws dynamodb create-table \
    --table-name SensitiveData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Atualize a criptografia da tabela.

```
# Update a table to use a different KMS key
aws dynamodb update-table \
    --table-name CustomerData \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Descreva a criptografia da tabela.

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
    --query "Table.SSEDescription"
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência de comandos da AWS CLI.
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Exemplos de tecnologia sem servidor para o DynamoDB

Os exemplos de código a seguir mostram como usar o DynamoDB com AWS SDKs.

Exemplos

- Invocar uma função do Lambda em um gatilho do DynamoDB
- Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB

Invocar uma função do Lambda em um gatilho do DynamoDB

Os exemplos de código a seguir mostram como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um fluxo do DynamoDB. A função recupera a carga útil do DynamoDB e registra em log o conteúdo do registro.

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
```

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
}
```

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumir um evento do DynamoDB com o Lambda usando TypeScript.

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data
```

```
            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
```

```
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB

Os exemplos de código a seguir mostram como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um fluxo do DynamoDB. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

.NET

SDK para .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
```

```
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""
```

```
 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;
```

```
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando JavaScript.

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando TypeScript.

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
```

```
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do DynamoDB com o Lambda usando PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
```

```
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Para ver uma lista completa dos Guias do desenvolvedor e exemplos de código do SDK da AWS, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Contribuições da comunidade da AWS para o DynamoDB

Contribuições da comunidade da AWS são exemplos criados e mantidos por várias equipes na AWS. Para deixar seu feedback, use o mecanismo fornecido nos repositórios vinculados.

Exemplos

- Compilar e testar uma aplicação com tecnologia sem servidor

Compilar e testar uma aplicação com tecnologia sem servidor

Os exemplos de código a seguir mostram como criar e testar uma aplicação com tecnologia sem servidor usando o API Gateway com o Lambda e o DynamoDB.

.NET

SDK para .NET

Mostra como compilar e testar uma aplicação com tecnologia sem servidor que consiste em um API Gateway com o Lambda e o DynamoDB usando o SDK .NET.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

Go

SDK para Go V2

Mostra como compilar e testar uma aplicação com tecnologia sem servidor que consiste em um API Gateway com o Lambda e o DynamoDB usando o SDK Go.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

Java

SDK para Java 2.x

Mostra como compilar e testar uma aplicação com tecnologia sem servidor que consiste em um API Gateway com o Lambda e o DynamoDB usando o SDK Java.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

Rust

SDK para Rust

Mostra como compilar e testar uma aplicação com tecnologia sem servidor que consiste em um API Gateway com o Lambda e o DynamoDB usando o SDK Rust.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Usar o DynamoDB com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Segurança e compatibilidade no Amazon DynamoDB

A segurança da nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você contará com um data center e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O modelo de responsabilidade compartilhada descreve isto como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem: a AWS é responsável pela proteção da infraestrutura que executa produtos da AWS na Nuvem da AWS. A AWS também fornece serviços que podem ser usados com segurança. A eficácia da nossa segurança é regularmente testada e verificada por auditores de terceiros como parte dos Programas de conformidade da AWS. Para saber mais sobre os programas de compatibilidade aplicáveis ao DynamoDB, consulte Serviços da AWS no escopo do programa de compatibilidade.
- Segurança da nuvem: sua responsabilidade é determinada pelo serviço da AWS que você usa. Também existe a responsabilidade por outros fatores, incluindo a confidencialidade de dados, os requisitos da organização e as leis e regulamentos aplicáveis.

Esta documentação ajudará você a entender como aplicar o modelo de responsabilidade compartilhada ao usar o DynamoDB. Os tópicos a seguir mostram como configurar o DynamoDB para atender aos seus objetivos de segurança e compatibilidade. Você também aprenderá como usar outros serviços da AWS que podem ajudar a monitorar e proteger seus recursos do DynamoDB.

Tópicos

- Políticas gerenciadas pela AWS para o Amazon DynamoDB
- Usar políticas baseadas em recursos para o DynamoDB
- Usar o controle de acesso por atributo com o DynamoDB
- Proteção de dados no DynamoDB
- AWS Identity and Access Management (IAM) e DynamoDB
- Validação de conformidade do DynamoDB por setor
- Resiliência e recuperação de desastres no Amazon DynamoDB
- Segurança da infraestrutura no Amazon DynamoDB
- AWS PrivateLink para DynamoDB

- Análise de configuração e vulnerabilidade no Amazon DynamoDB
- Práticas recomendadas de segurança para o Amazon DynamoDB

Políticas gerenciadas pela AWS para o Amazon DynamoDB

O DynamoDB usa políticas gerenciadas pela AWS para definir um conjunto de permissões que o serviço precisa para realizar ações específicas. O DynamoDB mantém e atualiza as políticas gerenciadas pela AWS. Não é possível alterar as permissões em políticas gerenciadas AWS. Para obter mais informações sobre as políticas gerenciadas pela AWS, consulte Políticas gerenciadas pela AWS no Guia do usuário do IAM.

Ocasionalmente, o DynamoDB pode acrescentar permissões adicionais a uma política gerenciada pela AWS para oferecer suporte a novos recursos. Esse tipo de atualização afeta todas as identidades (usuários, grupos e funções) em que a política está anexada. É mais provável que uma política gerenciada pela AWS seja atualizada quando um novo recurso é iniciado ou quando novas operações são disponibilizadas. O DynamoDB não remove permissões de uma política gerenciada pela AWS, portanto as atualizações de políticas não suspendem suas permissões existentes. Para ter uma lista das políticas gerenciadas da AWS, consulte AWS managed policies.

Política gerenciada pela AWS: DynamoDBReplicationServiceRolePolicy

Não é possível anexar a política `DynamoDBReplicationServiceRolePolicy` às suas entidades do IAM. Essa política é anexada a um perfil vinculado ao serviço que permite que o DynamoDB realize ações em seu nome. Para obter mais informações, consulte Usar o IAM com tabelas globais.

Essa política concede permissões que possibilitam que o perfil vinculado ao serviço execute a replicação de dados entre réplicas de tabelas globais. Ela também concede permissões administrativas para gerenciar réplicas de tabelas globais em seu nome.

Detalhes das permissões

Essa política concede permissões para fazer o seguinte:

- `dynamodb`: executar replicação de dados e gerenciar réplicas de tabelas.
- `application-autoscaling`: recuperar e gerenciar as configurações de ajuste de escala automático da tabela.
- `account`: recuperar o status da região para avaliar a acessibilidade a réplicas.

- `iam`: para criar a função vinculada ao serviço para ajuste de escala automático da aplicação, caso essa função ainda não exista.

A definição dessa política gerenciada pode ser encontrada aqui.

Política gerenciada pela AWS: AmazonDynamoDBFullAccess_v2

A política de escopo reduzido `AmazonDynamoDBFullAccess_v2` concede privilégios de acesso específicos aos usuários. É possível anexar a política `AmazonDynamoDBFullAccess_v2` às identidades do IAM. Essa política concede acesso administrativo a recursos do Amazon DynamoDB e concede a uma identidade do IAM (como usuário, grupo ou perfil) acesso aos Serviços da AWS aos quais o DynamoDB está integrado para usar todos os recursos do DynamoDB. O uso dessa política permite o acesso a todos os recursos do DynamoDB que estão disponíveis no AWS Management Console.

Detalhes de permissões

Esta política inclui as seguintes permissões:

- `Amazon DynamoDB`
- `DynamoDB Accelerator`
- `AWS KMS`
- `AWS Resource Groups Tagging`
- `Lambda`
- `Application Auto Scaling`
- `CloudWatch`
- `Amazon Kinesis`
- `Amazon EC2`
- `IAM`

Para analisar a política no formato `JSON`, consulte AmazonDynamoDBFullAccess_v2.

Política gerenciada da AWS: AmazonDynamoDBReadOnlyAccess

É possível anexar a política `AmazonDynamoDBReadOnlyAccess` às identidades do IAM.

Essa política concede acesso somente leitura ao Amazon DynamoDB.

Detalhes de permissões

Esta política inclui as seguintes permissões:

- `Amazon DynamoDB`: concede acesso somente leitura ao Amazon DynamoDB.
- `Amazon DynamoDB Accelerator (DAX)`: fornece acesso somente leitura ao Amazon DynamoDB Accelerator (DAX).
- `Application Auto Scaling`: isso permite que as entidades principais visualizem configurações do Aplicativo de ajuste de escala automático. Isso é necessário para que os usuários possam visualizar as políticas de escalabilidade automática anexadas a uma tabela.
- `CloudWatch`: permite que as entidades principais visualizem dados métricos e alarmes configurados no CloudWatch. Isso é necessário para que os usuários possam visualizar o tamanho da tabela faturável e os alarmes do CloudWatch que foram configurados para uma tabela.
- `AWS Data Pipeline`: possibilita que as entidades principais visualizem o AWS Data Pipeline e objetos associados.
- `Amazon EC2`: possibilita que as entidades principais visualizem VPCs, sub-redes e grupos de segurança do Amazon EC2.
- `IAM`: possibilita que as entidades principais visualizem os perfis do IAM.
- `AWS KMS`: permite que as entidades principais visualizem as chaves configuradas no AWS KMS. Isso é necessário para que os usuários possam visualizar as AWS KMS keys que eles criam e gerenciam na conta.
- `Amazon SNS`: possibilita que as entidades principais listem tópicos do Amazon SNS e assinaturas por tópico.
- `AWS Resource Groups`: possibilita que as entidades principais visualizem grupos e suas consultas.
- `AWS Resource Groups Tagging`: possibilita que as entidades principais listem todos os recursos marcados ou que foram marcados anteriormente em uma região.
- `Kinesis`: possibilita que as entidades principais visualizem as descrições dos fluxos de dados do Kinesis.
- `Amazon CloudWatch Contributor Insights`: possibilita que as entidades principais visualizem dados de séries temporais coletados pelas regras do Contributor Insights.

Para examinar a política no formato `JSON`, consulte AmazonDynamoDBReadOnlyAccess.

Atualizações do DynamoDB para políticas gerenciadas pela AWS

Esta tabela mostra as atualizações das políticas de gerenciamento de acesso da AWS para o DynamoDB.

Alteração	Descrição	Alterado em
`AmazonDynamoDBFullAccess` : descontinuada	Esta política foi substituída por uma política com escopo reduzido chamada `AmazonDynamoDBFullAccess_v2` . Após abril de 2025, a política `AmazonDynamoDBFullAccess` não poderá ser anexada a nenhum novo usuário, grupo ou perfil. Para obter mais informações, consulte Política gerenciada pela AWS: AmazonDynamoDBFullAccess_v2.	28 de abril de 2025
Atualização de `AmazonDynamoDBReadOnlyAccess` para uma política existente	O `AmazonDynamoDBReadOnlyAccess` adicionou as permissões `dynamodb:GetAbacStatus` e `dynamodb:UpdateAbacStatus` . Essas permissões autorizam que você visualize o status do ABAC e habilite o ABAC para a Conta da AWS na região atual.	18 de novembro de 2024
Atualização de `AmazonDynamoDBReadOnlyAccess` para uma política existente	`AmazonDynamoDBReadOnlyAccess` adicionou a permissão `dynamodb:GetResourcePolicy` . Essa permissão concede acesso para ler políticas baseadas em recursos vinculadas aos recursos do DynamoDB.	20 de março de 2024

Alteração	Descrição	Alterado em
Atualização de `DynamoDBReplicationServiceRolePolicy` para uma política existente	`DynamoDBReplicationServiceRolePolicy` adicionou a permissão `dynamodb:GetResourcePolicy`. Essa permissão possibilita que o perfil vinculado ao serviço leia as políticas baseadas em recursos vinculadas aos recursos do DynamoDB.	15 de dezembro de 2023
Atualização de `DynamoDBReplicationServiceRolePolicy` para uma política existente	`DynamoDBReplicationServiceRolePolicy` adicionou a permissão `account:ListRegions`. Essa permissão possibilita que o perfil vinculado ao serviço avalie a acessibilidade a réplicas.	10 de maio de 2023
Adição de `DynamoDBReplicationServiceRolePolicy` à lista de políticas gerenciadas	Adição de informações sobre a política gerenciada `DynamoDBReplicationServiceRolePolicy`, que é usada pelo perfil vinculado ao serviço de tabelas globais do DynamoDB.	10 de maio de 2023
As tabelas globais do DynamoDB começaram a monitorar alterações	As tabelas globais do DynamoDB começaram a monitorar alterações para suas políticas gerenciadas pela AWS.	10 de maio de 2023

Usar políticas baseadas em recursos para o DynamoDB

O DynamoDB é compatível com políticas baseadas em recursos para tabelas, índices e fluxos. Com as políticas baseadas em recursos, é possível definir as permissões de acesso especificando quem tem acesso a cada recurso e as ações que podem ser realizadas em cada recurso.

É possível associar uma política baseada em recursos aos recursos do DynamoDB, como uma tabela ou um fluxo. Nessa política, você deve especificar permissões para as entidades principais do Identity and Access Management (IAM) que podem realizar ações específicas nesses recursos do DynamoDB. Por exemplo, a política associada a uma tabela conterá permissões para acesso à tabela e seus índices. Como resultado, as políticas baseadas em recursos podem ajudar a simplificar o controle de acesso às tabelas, aos índices e aos fluxos do DynamoDB, definindo permissões em nível de recurso. O tamanho máximo de uma política que pode ser associada a um recurso do DynamoDB é de 20 KB.

Um benefício significativo do uso de políticas baseadas em recursos é simplificar o controle de acesso entre contas para conceder acesso entre contas a entidades principais do IAM em diferentes Contas da AWS. Para obter mais informações, consulte Política baseada em recursos para acesso entre contas.

As políticas baseadas em recursos também comportam integrações com recursos do analisador de acesso externo IAM Access Analyzer e do Bloqueio de Acesso Público (BPA). O IAM Access Analyzer relata o acesso entre contas a entidades externas especificadas em políticas baseadas em recursos. Ele também fornece visibilidade para ajudar a refinar as permissões e se adequar ao princípio de privilégio mínimo. O BPA ajuda a impedir o acesso público às tabelas, aos índices e aos fluxos do DynamoDB e é habilitado automaticamente nos fluxos de trabalho de criação e modificação de políticas baseadas em recursos.

Tópicos

- Criar uma tabela com uma política baseada em recursos
- Associar uma política a uma tabela existente do DynamoDB
- Associar uma política baseada em recurso a um fluxo do DynamoDB
- Remover uma política baseada em recurso de uma tabela do DynamoDB
- Acesso entre contas com políticas baseadas em recurso no DynamoDB
- Bloquear o acesso público com políticas baseadas em recurso no DynamoDB
- Operações de API do DynamoDB aceitas por políticas baseadas em recurso

- Autorização com políticas baseadas em identidade do IAM e políticas baseadas em recursos do DynamoDB
- Exemplos de políticas baseadas em recurso do DynamoDB
- Considerações sobre políticas baseadas em recurso do DynamoDB
- Práticas recomendadas para políticas baseadas em recurso do DynamoDB

Criar uma tabela com uma política baseada em recursos

É possível adicionar uma política baseada em recursos ao criar uma tabela usando o console do DynamoDB, a API CreateTable, a AWS CLI, o SDK da AWS ou um modelo do AWS CloudFormation.

AWS CLI

O exemplo a seguir cria uma tabela chamada *MusicCollection* usando o comando `create-table` da AWS CLI. Esse comando também inclui o parâmetro `resource-policy` que adiciona uma política baseada em recursos à tabela. Essa política possibilita que o usuário *John* realize as ações da API RestoreTableToPointInTime, GetItem e PutItem na tabela.

Lembre-se de substituir o texto em *itálico* pelas informações específicas do recurso.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --resource-policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::123456789012:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:RestoreTableToPointInTime\",
                        \"dynamodb:GetItem\",
                        \"dynamodb:DescribeTable\"
```

```
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS Management Console

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel, selecione Criar tabela.
3. Em Detalhes da tabela, insira o nome da tabela, a chave de partição e os detalhes da chave de classificação.
4. Em Configurações da tabela, selecione Personalizar configurações.
5. (Opcional) Especifique as opções para Classe de tabela, Calculadora de capacidade, Configurações de capacidade de leitura/gravação, Índices secundários, Criptografia em repouso e Proteção contra exclusão.
6. Em Política baseada em recursos, adicione uma política para definir as permissões de acesso para a tabela e seus índices. Nessa política, você especifica quem tem acesso a esses recursos e as ações que podem realizar em cada recurso. Para adicionar uma política, siga um destes procedimentos:

   - Digite ou cole um documento de política JSON. Para ter detalhes sobre a linguagem de política do IAM, consulte Criar políticas usando o editor de JSON no Guia do usuário do IAM.

     > **Tip**
     >
     > Para ver exemplos de políticas baseadas em recursos no Guia do desenvolvedor do Amazon DynamoDB, selecione Exemplos de políticas.

   - Selecione Adicionar nova declaração para adicionar uma nova declaração e insira as informações nos campos fornecidos. Repita esta etapa para todas as instruções que deseja adicionar.

> **Important**
>
> Solucione avisos de segurança, erros ou sugestões antes de salvar a política.

O exemplo de política do IAM a seguir possibilita que o usuário *John* realize as ações da API RestoreTableToPointInTime, GetItem e PutItem na tabela *MusicCollection*.

Lembre-se de substituir o texto em *itálico* pelas informações específicas do recurso.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/username"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (Opcional) Escolha Preview external access (Pré-visualizar o acesso externo) na canto inferior direito para pré-visualizar a forma como sua nova política afetará o acesso público e entre contas ao seu recurso. Antes de salvar sua política, você pode verificar se ela introduz novas descobertas do IAM Access Analyzer ou resolve as descobertas existentes. Se você não vir um analisador ativo, escolha Go to Access Analyzer (Acessar o Access Analyzer) para criar um analisador de contas no IAM Access Analyzer. Para ter mais informações, consulte Acesso de visualização.

8. Escolha Create table.

Modelo AWS CloudFormation

Using the AWS::DynamoDB::Table resource

O modelo do CloudFormation a seguir cria uma tabela com um fluxo usando o recurso AWS::DynamoDB::Table. Esse modelo também inclui políticas baseadas em recursos que são associadas à tabela e ao fluxo.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MusicCollectionTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "Artist",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    }
                ],
                "BillingMode": "PROVISIONED",
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "StreamSpecification": {
                  "StreamViewType": "OLD_IMAGE",
                  "ResourcePolicy": {
                    "PolicyDocument": {
                      "Version": "2012-10-17",
                      "Statement": [
                        {
                            "Principal": {
                                "AWS": "arn:aws:iam::111122223333:user/John"
                            },
```

```
                            "Effect": "Allow",
                            "Action": [
                                "dynamodb:GetRecords",
                                "dynamodb:GetShardIterator",
                                "dynamodb:DescribeStream"
                            ],
                            "Resource": "*"
                        }
                    ]
                }
            }
        },
        "TableName": "MusicCollection",
        "ResourcePolicy": {
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Principal": {
                            "AWS": [
                                "arn:aws:iam::111122223333:user/John"
                            ]
                        },
                        "Effect": "Allow",
                        "Action": "dynamodb:GetItem",
                        "Resource": "*"
                    }
                ]
            }
        }
    }

        }
    }
}
```

Using the AWS::DynamoDB::GlobalTable resource

O modelo do CloudFormation a seguir cria uma tabela com o recurso AWS::DynamoDB::GlobalTable e associa uma política baseada em recursos à tabela e ao seu fluxo.

```
{
```

```
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "GlobalMusicCollection": {
            "Type": "AWS::DynamoDB::GlobalTable",
            "Properties": {
                "TableName": "MusicCollection",
                "AttributeDefinitions": [{
                    "AttributeName": "Artist",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "Artist",
                    "KeyType": "HASH"
                }],
                "BillingMode": "PAY_PER_REQUEST",
                "StreamSpecification": {
                    "StreamViewType": "NEW_AND_OLD_IMAGES"
                },
                "Replicas": [
                    {
                        "Region": "us-east-1",
                        "ResourcePolicy": {
                            "PolicyDocument": {
                                "Version": "2012-10-17",
                                "Statement": [{
                                    "Principal": {
                                        "AWS": [
                                            "arn:aws:iam::111122223333:user/John"
                                        ]
                                    },
                                    "Effect": "Allow",
                                    "Action": "dynamodb:GetItem",
                                    "Resource": "*"
                                }]
                            }
                        },
                        "ReplicaStreamSpecification": {
                            "ResourcePolicy": {
                                "PolicyDocument": {
                                    "Version": "2012-10-17",
                                    "Statement": [{
                                        "Principal": {
                                            "AWS":
"arn:aws:iam::111122223333:user/John"
```

```
                                    },
                                    "Effect": "Allow",
                                    "Action": [
                                        "dynamodb:GetRecords",
                                        "dynamodb:GetShardIterator",
                                        "dynamodb:DescribeStream"
                                    ],
                                    "Resource": "*"
                                }]
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

Associar uma política a uma tabela existente do DynamoDB

É possível associar uma política baseada em recursos a uma tabela existente ou modificar uma política existente usando o console do DynamoDB, a API PutResourcePolicy, a AWS CLI, o SDK da AWS ou um modelo do AWS CloudFormation.

Exemplo de AWS CLI para associar uma nova política

O exemplo de política do IAM a seguir usa o comando `put-resource-policy` da AWS CLI para associar uma política baseada em recursos a uma tabela existente. Este exemplo possibilita que o usuário *John* realize as ações da API GetItem, PutItem, UpdateItem e UpdateTable em uma tabela existente chamada *MusicCollection*.

Lembre-se de substituir o texto em *itálico* pelas informações específicas do recurso.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
```

```
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:PutItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

Exemplo de AWS CLI para atualizar condicionalmente uma política existente

Para atualizar condicionalmente uma política existente baseada em recursos de uma tabela, é possível usar o parâmetro `expected-revision-id` opcional. O exemplo a seguir só atualizará a política se ela existir no DynamoDB e se o ID de revisão atual corresponder ao parâmetro `expected-revision-id` fornecido.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --expected-revision-id 1709841168699 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
```

```
            ]
        }"
```

AWS Management Console

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel, selecione uma tabela existente.
3. Navegue até a guia Permissões e selecione Criar política de tabela.
4. No editor de políticas baseadas em recursos, adicione a política que você gostaria de associar e selecione Criar política.

   O exemplo de política do IAM a seguir possibilita que o usuário *John* realize as ações da API GetItem, PutItem, UpdateItem e UpdateTable em uma tabela existente chamada *MusicCollection*.

   Lembre-se de substituir o texto em *itálico* pelas informações específicas do recurso.

   JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:UpdateTable"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

AWS SDK for Java 2.x

O exemplo de política do IAM a seguir usa o método `putResourcePolicy` para associar uma política baseada em recursos a uma tabela existente. Essa política possibilita que um usuário realize a ação da API GetItem em uma tabela existente.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutResourcePolicyRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * Get started with the AWS SDK for Java 2.x
 */
public class PutResourcePolicy {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableArn> <allowedAWSPrincipal>

                Where:
                    tableArn - The Amazon DynamoDB table ARN to attach the policy to.
 For example, arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection.
                    allowedAWSPrincipal - Allowed AWS principal ARN that the example
 policy will give access to. For example, arn:aws:iam::123456789012:user/John.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableArn = args[0];
        String allowedAWSPrincipal = args[1];
        System.out.println("Attaching a resource-based policy to the Amazon DynamoDB
 table with ARN " +
```

```
                tableArn);
        Region region = Region.US_WEST_2;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        String result = putResourcePolicy(ddb, tableArn, allowedAWSPrincipal);
        System.out.println("Revision ID for the attached policy is " + result);
        ddb.close();
    }

    public static String putResourcePolicy(DynamoDbClient ddb, String tableArn, String
 allowedAWSPrincipal) {
        String policy = generatePolicy(tableArn, allowedAWSPrincipal);
        PutResourcePolicyRequest request = PutResourcePolicyRequest.builder()
                .policy(policy)
                .resourceArn(tableArn)
                .build();

        try {
            return ddb.putResourcePolicy(request).revisionId();
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        return "";
    }

    private static String generatePolicy(String tableArn, String allowedAWSPrincipal) {
        return "{\n" +
                "    \"Version\": \"2012-10-17\",\n" +
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Effect\": \"Allow\",\n" +
                "            \"Principal\": {\"AWS\":\"" + allowedAWSPrincipal + "\"},
\n" +
                "            \"Action\": [\n" +
                "                \"dynamodb:GetItem\"\n" +
                "            ],\n" +
                "            \"Resource\": \"" + tableArn + "\"\n" +
                "        }\n" +
                "    ]\n" +
                "}";
```

```
    }
}
```

Associar uma política baseada em recurso a um fluxo do DynamoDB

É possível associar uma política baseada em recursos ao fluxo de uma tabela existente ou modificar uma política existente usando o console do DynamoDB, a API PutResourcePolicy, a AWS CLI, o SDK da AWS ou um modelo do AWS CloudFormation.

> **Note**
>
> Não é possível associar uma política a um fluxo ao criá-lo usando as APIs CreateTable ou UpdateTable. No entanto, é possível modificar ou excluir uma política após a exclusão de uma tabela. Também é possível modificar ou excluir a política de um fluxo desabilitado.

AWS CLI

O exemplo de política do IAM a seguir usa o comando `put-resource-policy` da AWS CLI para associar uma política baseada em recursos ao fluxo de uma tabela denominada *`MusicCollection`*. Este exemplo possibilita que o usuário *`John`* realize as ações da API GetRecords, GetShardIterator e DescribeStream no fluxo.

Lembre-se de substituir o texto em *`itálico`* pelas informações específicas do recurso.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetRecords\",
                        \"dynamodb:GetShardIterator\",
```

```
                        \"dynamodb:DescribeStream\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492\"
                }
            ]
        }"
```

AWS Management Console

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel do console do DynamoDB, selecione Tabelas e, depois, uma tabela existente.

   Verifique se a tabela selecionada tem fluxos ativados. Para ter informações sobre como ativar fluxos para uma tabela, consulte Habilitar um fluxo.
3. Escolha a aba Permissões.
4. Em Política baseada em recursos para fluxo ativo, selecione Criar política de fluxo.
5. No editor Política baseada em recursos, adicione uma política para definir as permissões de acesso para o fluxo. Nessa política, você deve especificar quem tem acesso ao fluxo e as ações que podem realizar no fluxo. Para adicionar uma política, siga um destes procedimentos:

   - Digite ou cole um documento de política JSON. Para ter detalhes sobre a linguagem de política do IAM, consulte Criar políticas usando o editor de JSON no Guia do usuário do IAM.

     > **Tip**
     >
     > Para ver exemplos de políticas baseadas em recursos no Guia do desenvolvedor do Amazon DynamoDB, selecione Exemplos de políticas.

   - Selecione Adicionar nova declaração para adicionar uma nova declaração e insira as informações nos campos fornecidos. Repita esta etapa para todas as instruções que deseja adicionar.

     > **Important**
     >
     > Solucione avisos de segurança, erros ou sugestões antes de salvar a política.

6. (Opcional) Escolha Preview external access (Pré-visualizar o acesso externo) na canto inferior direito para pré-visualizar a forma como sua nova política afetará o acesso público e entre contas ao seu recurso. Antes de salvar sua política, você pode verificar se ela introduz novas descobertas do IAM Access Analyzer ou resolve as descobertas existentes. Se você não vir um analisador ativo, escolha Go to Access Analyzer (Acessar o Access Analyzer) para criar um analisador de contas no IAM Access Analyzer. Para ter mais informações, consulte Acesso de visualização.
7. Escolha Criar política.

O exemplo de política do IAM a seguir associa uma política baseada em recursos ao fluxo de uma tabela denominada *MusicCollection*. Este exemplo possibilita que o usuário *John* realize as ações da API GetRecords, GetShardIterator e DescribeStream no fluxo.

Lembre-se de substituir o texto em *itálico* pelas informações específicas do recurso.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeStream"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Remover uma política baseada em recurso de uma tabela do DynamoDB

É possível excluir uma política baseada em recursos de uma tabela existente usando o console do DynamoDB, a API DeleteResourcePolicy, a AWS CLI, o SDK da AWS ou um modelo do AWS CloudFormation.

AWS CLI

O exemplo a seguir usa o comando `delete-resource-policy` da AWS CLI para remover uma política baseada em recursos de uma tabela denominada *`MusicCollection`*.

Lembre-se de substituir o texto em *`itálico`* pelas informações específicas do recurso.

```
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS Management Console

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel do console do DynamoDB, selecione Tabelas e, depois, uma tabela existente.
3. Escolha Permissões.
4. No menu suspenso Gerenciar política, selecione Excluir política.
5. Na caixa de diálogo Excluir política baseada em recursos para tabela, digite **`confirm`** para confirmar a ação de exclusão.
6. Escolha Excluir.

Acesso entre contas com políticas baseadas em recurso no DynamoDB

Usando uma política baseada em recursos, é possível fornecer acesso entre contas a recursos disponíveis em diferentes Contas da AWS. Todo o acesso entre contas permitido pelas políticas baseadas em recursos será relatado por meio das descobertas de acesso externo do IAM Access Analyzer, se você tiver um analisador na mesma Região da AWS que o recurso. O IAM Access Analyzer executa verificações de política para validar sua política em relação à gramática das políticas e às práticas recomendadas do IAM. Essas verificações geram descobertas e fornecem recomendações práticas que ajudam a criar políticas que sejam funcionais e estejam em

conformidade com as práticas recomendadas de segurança. É possível ver as descobertas ativas do IAM Access Analyzer na guia Permissões do console do DynamoDB.

Para ter informações sobre a validação de políticas usando o IAM Access Analyzer, consulte Validação de política do IAM Access Analyzer no Guia do usuário do IAM. Para visualizar uma lista de avisos, erros e sugestões retornados pelo IAM Access Analyzer, consulte Referência de verificação de políticas do IAM Access Analyzer.

Para conceder permissão GetItem a um usuário A na conta A para acessar uma tabela B na conta B, realize as seguintes etapas:

1. Associe uma política baseada em recursos à tabela B que conceda permissão ao usuário A para realizar a ação `GetItem`.
2. Associe uma política baseada em recursos ao usuário A que conceda permissão para realizar a ação `GetItem` na tabela B.

Usando a opção Visualizar acesso externo disponível no console do DynamoDB, é possível visualizar como sua nova política afetará o acesso público e entre contas ao seu recurso. Antes de salvar sua política, você pode verificar se ela introduz novas descobertas do IAM Access Analyzer ou resolve as descobertas existentes. Se você não vir um analisador ativo, escolha Go to Access Analyzer (Acessar o Access Analyzer) para criar um analisador de contas no IAM Access Analyzer. Para ter mais informações, consulte Acesso de visualização.

O parâmetro do nome da tabela nas APIs do plano de dados e do ambiente de gerenciamento do DynamoDB aceita o nome do recurso da Amazon (ARN) completo da tabela para comportar operações entre contas. Se você fornecer apenas o parâmetro do nome da tabela em vez de um ARN completo, a operação da API será realizada na tabela da conta à qual o solicitante pertence. Para ter um exemplo de política de use acesso entre contas, consulte Política baseada em recursos para acesso entre contas.

A conta do proprietário do recurso será cobrada mesmo quando uma entidade principal de outra conta estiver lendo ou gravando na tabela do DynamoDB na conta do proprietário. Se a tabela tiver um throughput provisionado, a soma de todas as solicitações das contas do proprietário e dos solicitantes em outras contas determinará se a solicitação terá controle de utilização (se o ajuste de escala automático estiver desabilitado) ou se a escala será reduzida ou aumentada verticalmente se o ajuste de escala automático não estiver habilitado.

As solicitações serão registradas nos logs do CloudTrail das contas do proprietário e do solicitante para que as duas contas possam rastrear qual conta acessou quais dados.

> **Note**
>
> O acesso entre contas das APIs do ambiente de gerenciamento tem um limite menor de transações por segundo (TPS) de quinhentas solicitações.

Bloquear o acesso público com políticas baseadas em recurso no DynamoDB

O Bloqueio de Acesso Público (BPA) é um recurso que identifica e impede a associação de políticas baseadas em recursos que concedem acesso público às tabelas, aos índices ou aos fluxos do DynamoDB em suas contas da Amazon Web Services (AWS). Com o BPA, é possível impedir o acesso público aos recursos do DynamoDB. O BPA realiza verificações durante a criação ou a modificação de uma política baseada em recursos e ajuda a melhorar o procedimento de segurança com o DynamoDB.

O BPA usa raciocínio automatizado para analisar o acesso concedido por sua política baseada em recursos e alerta você se essas permissões forem encontradas no momento da administração de uma política baseada em recursos. A análise verifica o acesso a todas as declarações de políticas baseadas em recursos, ações e ao conjunto de chaves de condição usadas nas políticas.

> **Important**
>
> O BPA ajuda a proteger os recursos impedindo que o acesso público seja concedido por meio de políticas baseadas em recursos que estão diretamente associadas aos recursos do DynamoDB, como tabelas, índices e fluxos. Além de usar o BPA, inspecione com cuidado as seguintes políticas para garantir que elas não concedam acesso público:
>
> - Políticas baseadas em identidade vinculadas a entidades principais da AWS associadas (por exemplo, perfis do IAM).
> - Políticas baseadas em recursos vinculadas a recursos da AWS associados (por exemplo, chaves do AWS Key Management Service).

Você deve garantir que a entidade principal não inclua uma entrada * ou que uma das chaves de condição especificadas restrinja o acesso das entidades principais ao recurso. Se a política baseada em recursos conceder acesso público à tabela, aos índices ou ao fluxo entre Contas da AWS, o

DynamoDB impedirá que você crie ou modifique a política até que a especificação dentro da política seja corrigida e considerada não pública.

É possível tornar uma política não pública especificando uma ou mais entidades principais no bloco `Principal`. O exemplo de política baseada em recursos a seguir bloqueia o acesso público ao especificar duas entidades principais.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
}
```

Políticas que restringem o acesso especificando determinadas chaves de condição também não são consideradas públicas. Junto com a avaliação da entidade principal especificada na política baseada em recursos, as seguintes chaves de condição confiáveis são usadas para concluir a avaliação de uma política baseada em recursos para acesso não público:

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`

- `aws:Ec2InstanceSourceVpc`
- `aws:SourceOrgID`
- `aws:SourceOrgPaths`

Além disso, para que uma política baseada em recursos não seja pública, os valores de nome do recurso da Amazon (ARN) e das chaves de string não devem conter curingas nem variáveis. Se a política baseada em recursos usa a chave `aws:PrincipalIsAWSService`, você deve garantir que tenha definido o valor da chave como verdadeiro.

A seguinte política limita o acesso ao usuário `John` na conta especificada. A condição faz com que a `Principal` seja restrita e não seja considerada pública.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

O exemplo a seguir de uma política baseada em recursos não pública restringe `sourceVPC` usando o operador `StringEquals`.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dynamodb:*",
```

```
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/
MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Operações de API do DynamoDB aceitas por políticas baseadas em recurso

Este tópico lista as operações de API aceitas por políticas baseadas em recursos. No entanto, em relação ao acesso entre contas, só é possível usar determinado conjunto de APIs do DynamoDB por meio de políticas baseadas em recursos. Não é possível associar políticas baseadas em recursos a tipos de recursos, como backups e importações. As ações do IAM, as quais correspondem às APIs que operam nesses tipos de recurso, são excluídas das ações do IAM aceitas nas políticas baseadas em recursos. Como os administradores da tabela definem as configurações internas da tabela na mesma conta, as APIs, como UpdateTimeToLive e DisableKinesisStreamingDestination, não comportam o acesso entre contas por meio de políticas baseadas em recursos.

As APIs do plano de dados e do ambiente de gerenciamento do DynamoDB que comportam o acesso entre contas também aceitam sobrecarga de nomes de tabelas, o que permite especificar o ARN em vez do nome da tabela. É possível especificar o ARN da tabela no parâmetro `TableName` dessas APIs. No entanto, nem todas essas APIs são compatíveis com o acesso entre contas.

Tópicos

- Operações de API do plano de dados
- Operações de API PartiQL
- Operações de API do ambiente de gerenciamento
- Operações de API de tabelas globais da versão 2019.11.21 (atual)
- Operações de API de tabelas globais versão 2017.11.29 (herdada)
- Operação de API de tags

- Operações de API de backup e restauração
- Operações de API de backup contínuo/restauração (PITR)
- Operações de API do Contributor Insights
- Operações API de exportação
- Operações de API de importação
- Operações de API do Amazon Kinesis Data Streams
- Operações de API de políticas baseadas em recursos
- Operações de API de vida útil
- Outras operações de API
- Operações de API do DynamoDB Streams

Operações de API do plano de dados

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API do plano de dados para políticas baseadas em recursos e acesso entre contas.

Plano de dados: APIs de tabelas/índices	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DeleteItem	Sim	Sim
GetItem	Sim	Sim
PutItem	Sim	Sim
Consulta	Sim	Sim
Scan	Sim	Sim
UpdateItem	Sim	Sim
TransactGetItems	Sim	Sim
TransactWriteItems	Sim	Sim
BatchGetItem	Sim	Sim

Plano de dados: APIs de tabelas/índices	Suporte a políticas baseadas em recursos.	Suporte entre contas.
BatchWriteItem	Sim	Sim

Operações de API PartiQL

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API PartiQL para políticas baseadas em recursos e acesso entre contas.

APIs PartiQL	Suporte a políticas baseadas em recursos.	Suporte entre contas.
BatchExecuteStatement	Sim	Não
ExecuteStatement	Sim	Não
ExecuteTransaction	Sim	Não

Operações de API do ambiente de gerenciamento

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API do ambiente de gerenciamento para políticas baseadas em recursos e acesso entre contas.

Ambiente de gerenciamento: APIs de tabelas	Suporte a políticas baseadas em recursos.	Suporte entre contas.
CreateTable	Não	Não
DeleteTable	Sim	Sim
DescribeTable	Sim	Sim
UpdateTable	Sim	Sim

Operações de API de tabelas globais da versão 2019.11.21 (atual)

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API de tabelas globais da versão 2019.11.21 (atual) para políticas baseadas em recursos e acesso entre contas.

APIs de tabelas globais versão 2019.11.21 (atual)	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeTableReplicaAutoScaling	Sim	Não
UpdateTableReplicaAutoScaling	Sim	Não

Operações de API de tabelas globais versão 2017.11.29 (herdada)

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API de tabelas globais da versão 2017.11.29 (herdada) para políticas baseadas em recursos e acesso entre contas.

APIs de tabelas globais versão 2017.11.29 (herdada)	Suporte a políticas baseadas em recursos.	Suporte entre contas.
CreateGlobalTable	Não	Não
DescribeGlobalTable	Não	Não
DescribeGlobalTableSettings	Não	Não
ListGlobalTables	Não	Não
UpdateGlobalTable	Não	Não
UpdateGlobalTableSettings	Não	Não

Operação de API de tags

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API relacionadas a tags para políticas baseadas em recursos e acesso entre contas.

APIs de tags	Suporte a políticas baseadas em recursos.	Suporte entre contas.
ListTagsOfResource	Sim	Sim
TagResource	Sim	Sim
UntagResource	Sim	Sim

Operações de API de backup e restauração

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API relacionadas a backup e restauração para políticas baseadas em recursos e acesso entre contas.

APIs de backup e restauração	Suporte a políticas baseadas em recursos.	Suporte entre contas.
CreateBackup	Sim	Não
DescribeBackup	Não	Não
DeleteBackup	Não	Não
RestoreTableFromBackup	Não	Não

Operações de API de backup contínuo/restauração (PITR)

A tabela a seguir lista o suporte em nível de API fornecido por operações de API relacionadas a backup contínuo/restauração (PITR) para políticas baseadas em recursos e acesso entre contas.

APIs de backup contínuo/ restauração (PITR)	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeContinuousBackups	Sim	Não
RestoreTableToPointInTime	Sim	Não
UpdateContinuousBackups	Sim	Não

Operações de API do Contributor Insights

A tabela a seguir lista o suporte em nível de API fornecido por operações de API relacionadas a backup contínuo/restauração (PITR) para políticas baseadas em recursos e acesso entre contas.

APIs do Contributor Insights	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeContributorInsights	Sim	Não
ListContributorInsights	Não	Não
UpdateContributorInsights	Sim	Não

Operações API de exportação

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API de exportação para políticas baseadas em recursos e acesso entre contas.

APIs de exportação	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeExport	Não	Não
ExportTableToPointInTime	Sim	Não
ListExports	Não	Não

Operações de API de importação

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API de importação para políticas baseadas em recursos e acesso entre contas.

APIs de importação	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeImport	Não	Não

APIs de importação	Suporte a políticas baseadas em recursos.	Suporte entre contas.
ImportTable	Não	Não
ListImports	Não	Não

Operações de API do Amazon Kinesis Data Streams

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API do Kinesis Data Streams para políticas baseadas em recursos e acesso entre contas.

APIs do Kinesis	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeKinesisStreamingDestination	Sim	Não
DisableKinesisStreamingDestination	Sim	Não
EnableKinesisStreamingDestination	Sim	Não
UpdateKinesisStreamingDestination	Sim	Não

Operações de API de políticas baseadas em recursos

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API de políticas baseadas em recursos para políticas baseadas em recursos e acesso entre contas.

APIs de políticas baseadas em recursos	Suporte a políticas baseadas em recursos.	Suporte entre contas.
GetResourcePolicy	Sim	Não

APIs de políticas baseadas em recursos	Suporte a políticas baseadas em recursos.	Suporte entre contas.
PutResourcePolicy	Sim	Não
DeleteResourcePolicy	Sim	Não

Operações de API de vida útil

A tabela a seguir lista o suporte em nível de API fornecido pelas operações de API de vida útil (TTL) para políticas baseadas em recursos e acesso entre contas.

APIs de TTL	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeTimeToLive	Sim	Não
UpdateTimeToLive	Sim	Não

Outras operações de API

A tabela a seguir lista o suporte em nível de API fornecido por outras operações de API variadas para políticas baseadas em recursos e acesso entre contas.

Outras APIs	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeLimits	Não	Não
DescribeEndpoints	Não	Não
ListBackups	Não	Não
ListTables	Não	Não

Operações de API do DynamoDB Streams

A tabela a seguir lista o suporte em nível de API de APIs do DynamoDB Streams para políticas baseadas em recursos e acesso entre contas.

APIs do DynamoDB Streams	Suporte a políticas baseadas em recursos.	Suporte entre contas.
DescribeStream	Sim	Sim
GetRecords	Sim	Sim
GetShardIterator	Sim	Sim
ListStreams	Não	Não

Autorização com políticas baseadas em identidade do IAM e políticas baseadas em recursos do DynamoDB

As políticas baseadas em identidade são associadas a uma identidade, como usuários, grupos de usuários e perfis do IAM. São os documentos da política do IAM que controlam quais ações uma identidade pode realizar, em quais recursos e em que condições. As políticas baseadas em identidade podem ser políticas gerenciadas e em linha.

As políticas baseadas em recursos são documentos de política do IAM que você associa a um recurso, como uma tabela do DynamoDB. Essas políticas concedem permissão ao principal especificado para executar ações específicas nesse recurso e definem em que condições isso se aplica. Por exemplo, a política baseada em recursos para uma tabela do DynamoDB também inclui o índice associado à tabela. As políticas baseadas em recursos são políticas em linha. Não há políticas baseadas em recursos gerenciadas.

Para ter mais informações sobre essas políticas, consulte Políticas baseadas em identidade e em recurso no Guia do usuário do IAM.

Se a entidade principal do IAM for da mesma conta que o proprietário do recurso, uma política baseada em recursos é suficiente para especificar as permissões de acesso ao recurso. Você ainda pode optar por uma política baseada em identidade do IAM além de uma política baseada em recursos. Em relação ao acesso entre contas, é necessário possibilitar o acesso de forma explícita

nas políticas de identidade e de recursos, conforme especificado em Acesso entre contas com políticas baseadas em recurso no DynamoDB. Ao usar os dois tipos de políticas, uma política é avaliada conforme descrito em Determinar se uma solicitação é permitida ou negada em uma conta.

Exemplos de políticas baseadas em recurso do DynamoDB

Ao especificar um ARN no campo `Resource` de uma política baseada em recursos, a política entrará em vigor somente se o ARN especificado corresponder ao ARN do recurso do DynamoDB ao qual está associado.

> **Note**
>
> Lembre-se de substituir o texto em *`itálico`* pelas informações específicas do recurso.

Política baseada em recursos para uma tabela

A política baseada em recursos a seguir, associada a uma tabela do DynamoDB chamada *`MusicCollection`*, oferece aos usuários do IAM *`John`* e *`Jane`* permissão para realizar as ações GetItem e BatchGetItem no recurso *`MusicCollection`*.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/username",
            "arn:aws:iam::111122223333:user/Jane"
          ]
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
```

```
          "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

Política baseada em recursos para um fluxo

A política baseada em recursos a seguir, associada a um fluxo do DynamoDB chamado `2024-02-12T18:57:26.492`, oferece aos usuários do IAM *John* e *Jane* permissão para realizar as ações GetRecords, GetShardIterator e DescribeStream no recurso `2024-02-12T18:57:26.492`.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/username",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Política baseada em recursos para acesso a fim de realizar todas as ações em recursos especificados

Para permitir que um usuário realize todas as ações em uma tabela e em todos os índices associados a uma tabela, é possível usar um caractere curinga (*) para representar as ações e os recursos associados à tabela. O uso de um caractere curinga para os recursos permitirá que o usuário acesse a tabela do DynamoDB e todos os índices associados, incluindo aqueles que ainda não foram criados. Por exemplo, a política a seguir concederá permissão ao usuário *John* para realizar qualquer ação na tabela *MusicCollection* e em todos os seus índices, incluindo quaisquer índices que serão criados no futuro.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/role-name"
      },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
index/index-name"
      ]
    }
  ]
}
```

Política baseada em recursos para acesso entre contas

É possível especificar permissões para uma identidade do IAM entre contas para acessar os recursos do DynamoDB. Por exemplo, talvez seja necessário que um usuário de uma conta confiável tenha acesso para ler o conteúdo da sua tabela, com a condição de que ele acesse somente itens e atributos específicos nesses itens. A política a seguir concede acesso ao usuário *John* por meio de um ID de Conta da AWS confiável *111111111111* para acessar dados de uma tabela na conta *123456789012* usando a API GetItem. A política garante que o usuário possa acessar somente itens com uma chave primária *Jane* e que o usuário só possa recuperar os atributos `Artist` e `SongTitle`, mas nenhum outro atributo.

> **Important**
>
> Se você não especificar a condição `SPECIFIC_ATTRIBUTES`, verá todos os atributos dos itens exibidos.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "CrossAccountTablePolicy",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:user/John"
            },
            "Action": "dynamodb:GetItem",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
```

```
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

Além da política anterior baseada em recursos, a política baseada em identidade associada ao usuário *John* também precisa permitir a ação da API `GetItem` para que o acesso entre contas funcione. Veja a seguir um exemplo de política baseada em identidade que deve ser associada ao usuário *John*.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
```

```
}
```

O usuário John pode fazer uma solicitação `GetItem` especificando o ARN da tabela no parâmetro `table-name` para acessar a tabela *`MusicCollection`* na conta *`123456789012`*.

```
aws dynamodb get-item \
    --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --key '{"Artist": {"S": "Jane"}' \
    --projection-expression 'Artist, SongTitle' \
    --return-consumed-capacity TOTAL
```

Política baseada em recursos com condições de endereço IP

É possível aplicar uma condição para restringir endereços IP de origem, nuvens privadas virtuais (VPCs) e endpoint da VPC (VPCE). É possível especificar permissões com base nos endereços de origem da solicitação de origem. Por exemplo, convém permitir que um usuário acesse os recursos do DynamoDB somente se eles estiverem sendo acessados de uma fonte IP específica, como um endpoint de VPN corporativo. Especifique esses endereços IP na declaração `Condition`.

O exemplo a seguir permite que o usuário *`John`* acesse qualquer recurso do DynamoDB quando os IPs de origem são `54.240.143.0/24` e `2001:DB8:1234:5678::/64`.

JSON

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17"       ,
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:user/username"
      },
      "Action":"dynamodb:*",
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
```

```
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

Também é possível negar todo o acesso aos recursos do DynamoDB, exceto quando a origem é um endpoint da VPC específico, por exemplo, *vpce-1a2b3c4d*.

JSON

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17"       ,
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

Política baseada em recursos usando um perfil do IAM

Você também pode especificar um perfil de serviço do IAM na política baseada em recursos. As entidades do IAM que assumem esse perfil são limitadas pelas ações permitidas especificadas para o perfil e para o conjunto específico de recursos dentro da política baseada em recursos.

O exemplo a seguir permite que uma entidade do IAM realize todas as ações do DynamoDB nos recursos *MusicCollection* e *MusicCollection* do DynamoDB.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/role-name" },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

Considerações sobre políticas baseadas em recurso do DynamoDB

Ao definir políticas baseadas em recursos para os recursos do DynamoDB, são feitas as seguintes considerações:

Considerações gerais

- O tamanho máximo aceito para um documento de política baseado em recursos é de 20 KB. O DynamoDB conta espaços em branco ao calcular o tamanho de uma política em relação a esse limite.
- As atualizações subsequentes de uma política para determinado recurso são bloqueadas por 15 segundos após uma atualização bem-sucedida da política para o mesmo recurso.
- No momento, só é possível associar uma política baseada em recursos a fluxos existentes. Não é possível associar uma política a um fluxo ao criá-la.

Considerações sobre tabelas globais

- As políticas baseadas em recursos não são compatíveis com réplicas da Global Table versão 2017.11.29 (herdada).

- Em uma política baseada em recursos, se a ação de um perfil vinculado ao serviço (SLR) do DynamoDB replicar dados para uma tabela global for negada, a adição ou a exclusão de uma réplica falhará com um erro.
- O recurso AWS::DynamoDB::GlobalTable não comporta a criação de uma réplica e a adição de uma política baseada em recursos a essa réplica na mesma atualização da pilha em regiões diferentes da região onde você implanta a atualização da pilha.

Considerações sobre o acesso entre contas

- O acesso entre contas usando políticas baseadas em recursos não comporta tabelas criptografadas com chaves gerenciadas da AWS porque não é possível conceder acesso entre contas à política do KMS gerenciada da AWS.

Considerações sobre o AWS CloudFormation

- As políticas baseadas em recursos não aceitam a detecção de oscilação. Se você atualizar uma política baseada em recursos fora do modelo de pilha do AWS CloudFormation, precisará atualizar a pilha do CloudFormation com as alterações.
- As políticas baseadas em recursos não comportam mudanças fora da banda. Se você adicionar, atualizar ou excluir uma política fora do modelo do CloudFormation, a alteração não será substituída se não houver alterações na política dentro do modelo.

  Por exemplo, digamos que seu modelo contenha uma política baseada em recursos que você atualize posteriormente fora do modelo. Se você não fizer nenhuma alteração na política no modelo, a política atualizada no DynamoDB não será sincronizada com a política no modelo.

  Por outro lado, digamos que seu modelo não contenha uma política baseada em recursos, mas você adiciona uma política fora do modelo. Essa política não será removida do DynamoDB, desde que você não a adicione ao modelo. Ao adicionar uma política ao modelo e atualizar a pilha, a política existente no DynamoDB será atualizada para corresponder à definida no modelo.

Práticas recomendadas para políticas baseadas em recurso do DynamoDB

Este tópico descreve as práticas recomendadas para definir permissões de acesso para os recursos do DynamoDB e as ações permitidas nesses recursos.

Simplificar o controle de acesso aos recursos do DynamoDB

Se as entidades principais do AWS Identity and Access Management que precisam acessar um recurso do DynamoDB fizerem parte da mesma Conta da AWS que a do proprietário do recurso, uma política baseada em identidade do IAM não será necessária para cada entidade principal. Uma política baseada em recursos associada aos recursos especificados será suficiente. Esse tipo de configuração simplifica o controle de acesso.

Proteger os recursos do DynamoDB com políticas baseadas em recursos

Para todas as tabelas e os fluxos do DynamoDB, crie políticas baseadas em recursos para impor o controle de acesso para esses recursos. As políticas baseadas em recursos permitem centralizar as permissões em nível de recursos, simplificar o controle de acesso às tabelas, aos índices e aos fluxos do DynamoDB e reduzir as despesas indiretas administrativas. Se nenhuma política baseada em recursos for especificada para uma tabela ou um fluxo, o acesso à tabela ou ao fluxo será negado implicitamente, a menos que políticas baseadas em identidade associadas às entidades principais do IAM permitam o acesso.

Aplique permissões de privilégio mínimo

Ao definir permissões com políticas baseadas em recursos para recursos do DynamoDB, conceda apenas as permissões necessárias para a realização de uma ação. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. É possível começar com permissões amplas enquanto explora as permissões necessárias para sua workload ou para seu caso de uso. À medida que seu caso de uso se desenvolve, você pode trabalhar para reduzir as permissões que concede para caminhar em direção ao privilégio mínimo.

Analisar a atividade de acesso entre contas para gerar políticas de privilégio mínimo

O IAM Access Analyzer relata o acesso entre contas a entidades externas especificadas em políticas baseadas em recursos e fornece visibilidade para ajudar você a refinar as permissões e se adequar ao privilégio mínimo. Para obter mais informações sobre a geração de políticas, consulte Geração de políticas do IAM Access Analyzer.

Usar o IAM Access Analyzer para gerar políticas de privilégio mínimo

Para conceder apenas as permissões necessárias para executar uma tarefa, você pode gerar políticas com base em sua atividade de acesso registrada no AWS CloudTrail. O IAM Access Analyzer analisa os serviços e as ações que suas políticas usam.

Usar o controle de acesso por atributo com o DynamoDB

O controle de acesso por atributo (ABAC) é uma estratégia de autorização que define as permissões de acesso com base em condições de tag em suas políticas baseadas em identidade ou em outras políticas da AWS, como políticas baseadas em recursos e políticas do IAM da organização. É possível anexar tags às tabelas do DynamoDB, que são então avaliadas em relação às condições baseadas em tags. Os índices associados a uma tabela herdam as tags adicionadas à tabela. É possível adicionar até cinquenta tags a cada tabela do DynamoDB. O tamanho máximo aceito para todas as tags em uma tabela é de 10 KB. Para ter mais informações sobre a marcação de recursos do DynamoDB e restrições de marcação, consulte Marcar recursos no DynamoDB e the section called “Restrições de marcação no DynamoDB”.

Para ter informações sobre como usar essas tags para controlar o acesso aos recursos da AWS, consulte no Guia do usuário do IAM:

- What is ABAC for AWS
- Controlar o acesso a recursos da AWS usando tags

Com o uso do ABAC, é possível impor diferentes níveis de acesso para que suas equipes e aplicações realizem ações nas tabelas do DynamoDB usando menos políticas. É possível especificar uma tag no elemento de condição de uma política do IAM para controlar o acesso às tabelas ou aos índices do DynamoDB. Essas condições determinam o nível de acesso que uma entidade principal do IAM, um usuário ou um perfil tem às tabelas e aos índices do DynamoDB. Quando uma entidade principal do IAM faz uma solicitação de acesso ao DynamoDB, as tags do recurso e da identidade são avaliadas em relação às condições de tag na política do IAM. Depois disso, a política será efetiva somente se as condições de tag forem atendidas. Isso permite criar uma política do IAM que informe efetivamente uma das seguintes opções:

- Permitir que o usuário gerencie apenas os recursos que têm uma tag com uma chave *X* e um valor *Y*.
- Negar acesso a todos os usuários aos recursos marcados com uma chave *X*.

Por exemplo, é possível criar uma política que permita que os usuários atualizem uma tabela apenas se ela tiver o par de chave-valor de tag: `"environment": "staging"`. É possível usar a chave de condição aws:ResourceTag para permitir ou negar acesso a uma tabela com base nas tags anexadas a essa tabela.

É possível incluir condições baseadas em atributos ao criar a política ou posteriormente usar o AWS Management Console, a API da AWS, a AWS Command Line Interface (AWS CLI), o AWS SDK ou o AWS CloudFormation.

O exemplo a seguir permite a ação UpdateItem em uma tabela chamada `MusicTable` se ela incluir uma chave de tag com o nome `environment` e o valor `production`.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

Tópicos

- Por que devo usar o ABAC?
- Chaves de condição para implementar o ABAC com o DynamoDB
- Considerações sobre o uso do ABAC com o DynamoDB
- Habilitar o ABAC no DynamoDB
- Usar o ABAC com tabelas e índices do DynamoDB
- Exemplos de uso do ABAC com tabelas e índices do DynamoDB
- Solução de problemas comuns de ABAC para tabelas e índices do DynamoDB

Por que devo usar o ABAC?

- Gerenciamento de políticas mais simples: é possível usar menos políticas porque não é necessário criar políticas diferentes para definir o nível de acesso para cada entidade principal do IAM.
- Controle de acesso escalável: ajustar a escala do controle de acesso é mais fácil com o ABAC, pois não é necessário atualizar as políticas ao criar recursos do DynamoDB. É possível usar tags para autorizar o acesso às entidades principais do IAM que contêm tags correspondentes às tags do recurso. É possível integrar novas entidades principais do IAM ou recursos do DynamoDB e aplicar as tags apropriadas para conceder automaticamente as permissões necessárias sem precisar fazer nenhuma alteração na política.
- Gerenciamento de permissões refinado: é prática recomendada conceder o privilégio mínimo ao criar políticas. Com o uso do ABAC, é possível criar tags para a entidade principal do IAM e usá-las para conceder acesso a ações e recursos específicos que correspondam às tags na entidade principal do IAM.
- Alinhamento com o diretório corporativo: é possível associar tags a atributos de funcionários existentes do seu diretório corporativo com o objetivo de alinhar suas políticas de controle de acesso à estrutura organizacional.

Chaves de condição para implementar o ABAC com o DynamoDB

É possível usar as seguintes chaves de condição nas políticas da AWS para controlar o nível de acesso às suas tabelas e índices do DynamoDB:

- aws:ResourceTag/tag-key: controla o acesso com base no fato de o par de chave-valor da tag em uma tabela ou um índice do DynamoDB corresponder ou não à chave e ao valor da tag em uma política. Essa chave de condição é relevante para todas as APIs que operam em uma tabela ou um índice existente.

  As condições `dynamodb:ResourceTag` são avaliadas como se você não tivesse anexado nenhuma tag a um recurso.
- aws:RequestTag/tag-key: permite comparar o par de chave/valor da tag que foi transmitido na solicitação com o par de tags especificado na política. Essa chave de condição é relevante para APIs que contêm tags como parte da carga útil da solicitação. Essas APIs incluem CreateTable e TagResource.

- aws:TagKeys: compara as chaves de tag em uma solicitação com as chaves especificadas na política. Essa chave de condição é relevante para APIs que contêm tags como parte da carga útil da solicitação. Essas APIs incluem `CreateTable`, `TagResource` e `UntagResource`.

Considerações sobre o uso do ABAC com o DynamoDB

Ao usar o ABAC com tabelas ou índices do DynamoDB, as seguintes considerações se aplicam:

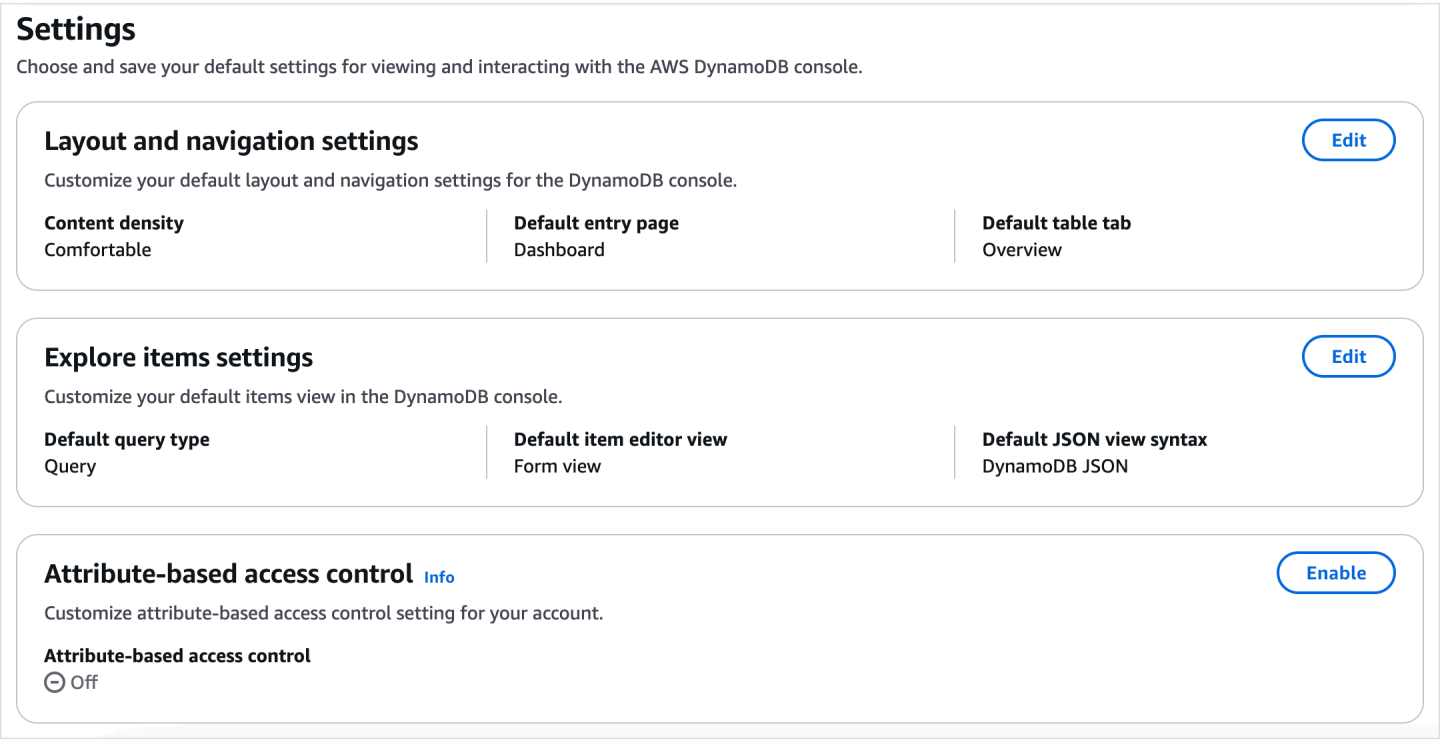
- A marcação e o ABAC não são aceitos no DynamoDB Streams.
- A marcação e o ABAC não são aceitos em backups do DynamoDB. Para usar o ABAC com backups, recomendamos usar o AWS Backup.
- As tags não são preservadas nas tabelas restauradas. É necessário adicionar tags às tabelas restauradas para usar condições baseadas em tags em políticas.

Habilitar o ABAC no DynamoDB

Para a maioria das Contas da AWS, o ABAC é habilitado por padrão. Usando o console do DynamoDB, é possível confirmar se o ABAC está habilitado para sua conta. Para fazer isso, abra o console do DynamoDB com um perfil que tenha a permissão dynamodb:GetAbacStatus. Depois, abra a página Configurações do console do DynamoDB.

Se você não vir o cartão Controle de acesso por atributo ou se o cartão exibir o status Ativado, isso significa que o ABAC está habilitado para sua conta. No entanto, se você vir o cartão Controle de acesso por atributo com o status Desativado, conforme mostrado na imagem a seguir, o ABAC não está habilitado para sua conta.

Controle de acesso por atributo: não habilitado



O ABAC não está habilitado para Contas da AWS para as quais as condições baseadas em tags especificadas nas políticas baseadas em identidade ou em outras políticas ainda precisam ser auditadas. Se o ABAC não estiver habilitado para sua conta, as condições baseadas em tags nas políticas destinadas a atuar nas tabelas ou nos índices do DynamoDB serão avaliadas como se nenhuma tag estivesse presente nos recursos ou nas solicitações de API. Quando o ABAC está habilitado para sua conta, as condições baseadas em tags nas políticas da conta são avaliadas levando em conta as tags anexadas às tabelas ou às solicitações de API.

Para habilitar o ABAC para sua conta, recomendamos primeiro auditar as políticas conforme descrito na seção Auditoria de políticas. Depois, inclua as permissões necessárias para o ABAC na política do IAM. Por fim, siga as etapas descritas em Habilitar o ABAC no console para habilitar o ABAC para sua conta na região atual. Depois de habilitar o ABAC, é possível cancelá-lo nos próximos sete dias corridos após a adesão.

Tópicos

- Auditar as políticas antes de habilitar o ABAC
- Permissões do IAM necessárias para habilitar o ABAC
- Habilitar o ABAC no console

Auditar as políticas antes de habilitar o ABAC

Antes de habilitar o ABAC para sua conta, audite as políticas para confirmar se as condições baseadas em tags que possam existir nas políticas da conta estão configuradas conforme o pretendido. A auditoria das políticas ajudará a evitar surpresas com alterações de autorização nos fluxos de trabalho do DynamoDB após a habilitação do ABAC. Para ver exemplos do uso de condições baseadas em atributos com tags e o comportamento antes e depois da implementação do ABAC, consulte Exemplos de uso do ABAC com tabelas e índices do DynamoDB.

Permissões do IAM necessárias para habilitar o ABAC

É necessário ter a permissão `dynamodb:UpdateAbacStatus` para habilitar o ABAC para sua conta na região atual. Para confirmar se o ABAC está habilitado para a conta, você também deve ter a permissão `dynamodb:GetAbacStatus`. Com essa permissão, é possível visualizar o status do ABAC para uma conta em qualquer região. É necessário ter essas permissões além da permissão necessária para acessar o console do DynamoDB.

A política do IAM a seguir concede a permissão para habilitar o ABAC e visualizar seu status para uma conta na região atual.

```
{
"version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
             ],
            "Resource": "*"
        }
    ]
}
```

Habilitar o ABAC no console

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação superior, escolha a região para a qual você deseja habilitar o ABAC.
3. No painel de navegação esquerdo, escolha Configurações.

4. Na página Configurações, faça o seguinte:

   a. No cartão Controle de acesso por atributo, escolha Habilitar.

   b. Na caixa Confirmar configuração de controle de acesso por atributo, escolha Habilitar para confirmar sua escolha.

      Isso habilita o ABAC para a região atual e o cartão Controle de acesso por atributo mostra o status Ativado.

      Se você quiser cancelar a habilitação do ABAC no console, poderá fazer isso nos próximos sete dias corridos após a habilitação. Para cancelar a habilitação, escolha Desabilitar no cartão Controle de acesso por atributo na página Configurações.

      > **Note**
      >
      > A atualização do status do ABAC é uma operação assíncrona. Se as tags em suas políticas não forem avaliadas imediatamente, talvez seja necessário esperar algum tempo, pois a aplicação das alterações será finalmente consistente.

Usar o ABAC com tabelas e índices do DynamoDB

As etapas a seguir demonstram como configurar permissões usando o ABAC. Neste cenário de exemplo, você adicionará tags a uma tabela do DynamoDB e criará um perfil do IAM com uma política que inclui condições baseadas em tags. Depois, você testará as permissões permitidas na tabela do DynamoDB combinando as condições da tag.

Tópicos

- Etapa 1: adicionar tags a uma tabela do DynamoDB
- Etapa 2: criar um perfil do IAM com uma política que inclui condições baseadas em tags
- Etapa 3: testar permissões

Etapa 1: adicionar tags a uma tabela do DynamoDB

É possível adicionar tags a tabelas novas ou existentes do DynamoDB usando o AWS Management Console, a API da AWS, a AWS Command Line Interface (AWS CLI), o AWS SDK ou o AWS

CloudFormation. Por exemplo, o comando da CLI tag-resource a seguir adiciona uma tag a uma tabela chamada `MusicTable`.

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

Etapa 2: criar um perfil do IAM com uma política que inclui condições baseadas em tags

Crie uma política do IAM usando a chave de condição aws:ResourceTag/tag-key para comparar o par de chave-valor da tag especificado na política do IAM com o par de chave-valor anexado à tabela. O seguinte exemplo de política permite que os usuários coloquem ou atualizem itens em tabelas caso essas tabelas contenham o par de chave-valor de tag: `"environment": "staging"`. Se uma tabela não tiver o par de chave-valor de tag especificado, essas ações serão negadas.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

Etapa 3: testar permissões

1. Anexe a política do IAM a um usuário ou um perfil de teste à sua Conta da AWS. Assegure-se de que a entidade principal do IAM utilizada não tenha acesso à tabela do DynamoDB por meio de uma política diferente.
2. Assegure-se de que sua tabela do DynamoDB contenha a chave de tag `"environment"` com um valor de `"staging"`.
3. Realize as ações `dynamodb:PutItem` e `dynamodb:UpdateItem` na tabela marcada. Essas ações devem ser bem-sucedidas caso o par de chave-valor de tag `"environment": "staging"` esteja presente.

   Se você realizar essas ações em uma tabela que não tenha o par de chave-valor de tag `"environment": "staging"`, sua solicitação falhará com uma `AccessDeniedException`.

Você também pode analisar os outros exemplos de casos de uso descritos na seção a seguir para implementar o ABAC e realizar mais testes.

Exemplos de uso do ABAC com tabelas e índices do DynamoDB

Os exemplos a seguir descrevem alguns casos de uso para implementar condições baseadas em atributos usando tags.

Tópicos

- Exemplo 1: permitir uma ação usando aws:ResourceTag
- Exemplo 2: permitir uma ação usando aws:RequestTag
- Exemplo 3: negar uma ação usando aws:TagKeys

Exemplo 1: permitir uma ação usando aws:ResourceTag

Usando a chave de condição `aws:ResourceTag/tag-key`, é possível comparar o par de chave-valor da tag especificado em uma política do IAM com o par de chave-valor anexado a uma tabela do DynamoDB. Por exemplo, é possível permitir uma ação específica, como PutItem, caso haja correspondência das condições da tag em uma política do IAM e em uma tabela. Para fazer isso, execute as seguintes etapas:

Using the AWS CLI

1. Crie uma tabela. O exemplo a seguir usa o comando create-table da AWS CLI para criar uma tabela chamada `myMusicTable`.

```
aws dynamodb create-table \
  --table-name myMusicTable \
  --attribute-definitions AttributeName=id,AttributeType=S \
  --key-schema AttributeName=id,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --region us-east-1
```

2. Adicione uma tag a essa tabela. O exemplo de comando tag-resource da AWS CLI a seguir adiciona o par de chave-valor de tag `Title: ProductManager` à `myMusicTable`.

```
aws dynamodb tag-resource --region us-east-1 --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/myMusicTable --tags Key=Title,Value=ProductManager
```

3. Crie uma política em linha e adicione-a a um perfil que tenha a política gerenciada AmazonDynamoDBReadOnlyAccess pela AWS anexada a ela, conforme mostrado no exemplo a seguir.

   JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Essa política permite a ação `PutItem` na tabela quando a chave e o valor da tag anexados à tabela correspondem às tags especificadas na política.

4. Assuma o perfil com as políticas descritas na Etapa 3.
5. Use o comando put-item da AWS CLI para inserir um item na `myMusicTable`.

```
aws dynamodb put-item \
    --table-name myMusicTable --region us-east-1 \
    --item '{
        "id": {"S": "2023"},
        "title": {"S": "Happy Day"},
        "info": {"M": {
            "rating": {"N": "9"},
            "Artists": {"L": [{"S": "Acme Band"}, {"S": "No One You Know"}]},
            "release_date": {"S": "2023-07-21"}
        }}
    }'
```

6. Verifique a tabela para saber se o item foi adicionado a ela.

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. Crie uma tabela. O exemplo a seguir usa a API CreateTable para criar uma tabela chamada `myMusicTable`.

```
DynamoDbClient dynamoDB = DynamoDbClient.builder().region(region).build();
CreateTableRequest createTableRequest = CreateTableRequest.builder()
    .attributeDefinitions(
        Arrays.asList(
            AttributeDefinition.builder()
            .attributeName("id")
            .attributeType(ScalarAttributeType.S)
            .build()
        )
    )
    .keySchema(
        Arrays.asList(
            KeySchemaElement.builder()
```

```
            .attributeName("id")
            .keyType(KeyType.HASH)
            .build()
        )
    )
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(5L)
        .writeCapacityUnits(5L)
        .build()
    )
    .tableName("myMusicTable")
    .build();

CreateTableResponse createTableResponse =
 dynamoDB.createTable(createTableRequest);
String tableArn = createTableResponse.tableDescription().tableArn();
String tableName = createTableResponse.tableDescription().tableName();
```

2. Adicione uma tag a essa tabela. A API TagResource no exemplo a seguir adiciona o par de chave-valor de tag `Title: ProductManager` à `myMusicTable`.

```
TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
    .resourceArn(tableArn)
    .tags(
        Arrays.asList(
            Tag.builder()
            .key("Title")
            .value("ProductManager")
            .build()
        )
    )
    .build();
dynamoDB.tagResource(tagResourceRequest);
```

3. Crie uma política em linha e adicione-a a um perfil que tenha a política gerenciada AmazonDynamoDBReadOnlyAccess pela AWS anexada a ela, conforme mostrado no exemplo a seguir.

   JSON

```
{
  "Version": "2012-10-17"       ,
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Essa política permite a ação `PutItem` na tabela quando a chave e o valor da tag anexados à tabela correspondem às tags especificadas na política.

4. Assuma o perfil com as políticas descritas na Etapa 3.
5. Use a API PutItem para colocar um item na `myMusicTable`.

```
HashMap<String, AttributeValue> info = new HashMap<>();
info.put("rating", AttributeValue.builder().s("9").build());
info.put("artists", AttributeValue.builder().ss(List.of("Acme Band","No One You
 Know").build());
info.put("release_date", AttributeValue.builder().s("2023-07-21").build());

HashMap<String, AttributeValue> itemValues = new HashMap<>();
itemValues.put("id", AttributeValue.builder().s("2023").build());
itemValues.put("title", AttributeValue.builder().s("Happy Day").build());
itemValues.put("info", AttributeValue.builder().m(info).build());


PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();
dynamoDB.putItem(putItemRequest);
```

6. Verifique a tabela para saber se o item foi adicionado a ela.

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
```

```
                .build();

ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the AWS SDK para Python (Boto3)

1. Crie uma tabela. O exemplo a seguir usa a API CreateTable para criar uma tabela chamada `myMusicTable`.

```
create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    },
)

table_arn = create_table_response['TableDescription']['TableArn']
```

2. Adicione uma tag a essa tabela. A API TagResource no exemplo a seguir adiciona o par de chave-valor de tag `Title: ProductManager` à `myMusicTable`.

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
```

```
)
```

3. Crie uma política em linha e adicione-a a um perfil que tenha a política gerenciada AmazonDynamoDBReadOnlyAccess pela AWS anexada a ela, conforme mostrado no exemplo a seguir.

   JSON

   ```
   {
       "Version": "2012-10-17"       ,
       "Statement": [
           {
           "Effect": "Allow",
           "Action": "dynamodb:PutItem",
           "Resource": "arn:aws:dynamodb:*:*:table/*",
           "Condition": {
               "StringEquals": {
               "aws:ResourceTag/Title": "ProductManager"
               }
           }
           }
       ]
       }
   ```

   Essa política permite a ação `PutItem` na tabela quando a chave e o valor da tag anexados à tabela correspondem às tags especificadas na política.

4. Assuma o perfil com as políticas descritas na Etapa 3.
5. Use a API PutItem para colocar um item na `myMusicTable`.

   ```
   put_item_response = client.put_item(
       TableName = 'myMusicTable'
       Item = {
           'id': '2023',
           'title': 'Happy Day',
           'info': {
               'rating': '9',
               'artists': ['Acme Band','No One You Know'],
               'release_date': '2023-07-21'
           }
       }
   ```

```
)
```

6. Verifique a tabela para saber se o item foi adicionado a ela.

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

Sem ABAC

Se o ABAC não estiver habilitado para a Conta da AWS, as condições de tag na política do IAM e na tabela do DynamoDB não serão correspondentes. Consequentemente, a ação `PutItem` exibirá uma `AccessDeniedException` devido ao efeito da política `AmazonDynamoDBReadOnlyAccess`.

```
An error occurred (AccessDeniedException) when calling the PutItem operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoDBReadOnlyAccess/Alice is
 not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable because no identity-based policy allows the
 dynamodb:PutItem action.
```

Com ABAC

Se o ABAC estiver habilitado para a Conta da AWS, a ação `put-item` será concluída com êxito e adicionará um novo item à tabela. Isso ocorre porque a política em linha na tabela permite a ação `PutItem` caso as condições de tag na política do IAM e na tabela coincidam.

Exemplo 2: permitir uma ação usando aws:RequestTag

Usando a chave de condição aws:RequestTag/tag-key, você pode comparar o par de chave-valor de tag que é transmitido em sua solicitação com o par de tags especificado na política do IAM. Por exemplo, você pode permitir uma ação específica, como `CreateTable`, usando o `aws:RequestTag` caso as condições de tag não coincidam. Para fazer isso, execute as seguintes etapas:

Using the AWS CLI

1. Crie uma política em linha e adicione-a a um perfil que tenha a política gerenciada AmazonDynamoDBReadOnlyAccess pela AWS anexada a ela, conforme mostrado no exemplo a seguir.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Crie uma tabela que contenha o par de chave-valor de tag `"Owner": "John"`.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Owner,Value=John \
--table-name myMusicTable
```

Using the AWS SDK para Python (Boto3)

1. Crie uma política em linha e adicione-a a um perfil que tenha a política gerenciada AmazonDynamoDBReadOnlyAccess pela AWS anexada a ela, conforme mostrado no exemplo a seguir.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Crie uma tabela que contenha o par de chave-valor de tag `"Owner": "John"`.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
```

```
    },
    Tags=[
        {
            'Key': 'Owner',
            'Value': 'John'
        },
    ],
)
```

Sem ABAC

Se o ABAC não estiver habilitado para a Conta da AWS, as condições de tag na política em linha e na tabela do DynamoDB não vão coincidir. Consequentemente, a solicitação `CreateTable` falhará e a tabela não será criada.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/Admin/John is not authorized to perform:
 dynamodb:CreateTable on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
myMusicTable because no identity-based policy allows the dynamodb:CreateTable action.
```

Com ABAC

Se o ABAC estiver habilitado para a Conta da AWS, sua solicitação de criação de tabela será concluída com êxito. Como o par de chave-valor de tag `"Owner": "John"` está presente na solicitação `CreateTable`, a política em linha permite que o usuário `John` realize a ação `CreateTable`.

Exemplo 3: negar uma ação usando aws:TagKeys

Usando a chave de condição aws:TagKeys, você pode comparar as chaves de tag em uma solicitação com as chaves especificadas na política do IAM. Por exemplo, será possível negar uma ação específica, como `CreateTable`, usando `aws:TagKeys` se uma chave de tag específica não estiver presente na solicitação. Para fazer isso, execute as seguintes etapas:

Using the AWS CLI

1. Adicione uma política gerenciada pelo cliente a um perfil que tenha a política gerenciada AmazonDynamoDBFullAccess pela AWS anexada a ela, conforme mostrado no exemplo a seguir.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Assuma o perfil ao qual a política foi anexada e crie uma tabela com a chave de tag `Title`.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Title,Value=ProductManager \
--table-name myMusicTable
```

Using the AWS SDK para Python (Boto3)

1. Adicione uma política gerenciada pelo cliente a um perfil que tenha a política gerenciada AmazonDynamoDBFullAccess pela AWS anexada a ela, conforme mostrado no exemplo a seguir.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Assuma o perfil ao qual a política foi anexada e crie uma tabela com a chave de tag `Title`.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
```

```
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ],
)
```

Sem ABAC

Se o ABAC não estiver habilitado para a Conta da AWS, o DynamoDB não enviará as chaves de tag no comando `create-table` ao IAM. A condição `Null` garante que a condição seja avaliada como `false` caso não haja tags na solicitação. Como a política `Deny` não coincide, o comando `create-table` é concluído com êxito.

Com ABAC

Se o ABAC estiver habilitado para a Conta da AWS, as chaves de tag transmitidas no comando `create-table` serão enviadas ao IAM. A chave de tag `Title` é avaliada em relação à chave de tag baseada em condições, `CostCenter`, presente na política `Deny`. A chave de tag `Title` não corresponde à chave de tag presente na política `Deny` por causa do operador `StringNotEquals`. Portanto, a ação `CreateTable` falha e sua tabela não é criada. A execução do comando `create-table` exibe uma `AccessDeniedException`.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoFullAccessRole/ProductManager is
 not authorized to perform: dynamodb:CreateTable on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable with an explicit deny in an identity-based
 policy.
```

Solução de problemas comuns de ABAC para tabelas e índices do DynamoDB

Este tópico oferece conselhos para solução de erros e problemas comuns que você pode encontrar ao implementar o ABAC em tabelas ou índices do DynamoDB.

Chaves de condição específicas do serviço em políticas geram um erro

Chaves de condição específicas do serviço não são consideradas chaves de condição válidas. Se você usou essas chaves em suas políticas, elas gerarão um erro. Para corrigir esse problema, você deve substituir as chaves de condição específicas do serviço por uma chave de condição apropriada para implementar o ABAC no DynamoDB.

Por exemplo, digamos que você tenha usado a chave de condição `dynamodb:ResourceTag` em uma política em linha que executa a solicitação PutItem. Imagine que a solicitação falhe com uma `AccessDeniedException`. O exemplo a seguir mostra a política em linha errônea com a chave de condição `dynamodb:ResourceTag`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "dynamodb:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Para corrigir esse problema, substitua a chave de condição `dynamodb:ResourceTag` por `aws:ResourceTag`, conforme mostrado no exemplo a seguir.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Não é possível cancelar o ABAC

Se o ABAC foi habilitado para sua conta por meio do Suporte, você não poderá cancelá-lo pelo console do DynamoDB. Para cancelá-lo, entre em contato com o Suporte.

Você mesmo poderá cancelar o ABAC somente se as seguintes condições forem verdadeiras:

- Você usou o método de autoatendimento para ativar por meio do console do DynamoDB.
- Você está cancelando o recurso em até sete dias corridos após a ativação.

Proteção de dados no DynamoDB

O Amazon DynamoDB fornece uma infraestrutura de armazenamento resiliente projetada para armazenamento de dados essenciais à missão e primários. Os dados são armazenados de maneira redundante em vários dispositivos de diversas instalações em uma região do Amazon DynamoDB.

O DynamoDB protege os dados de usuário armazenados em repouso e também os dados em trânsito entre os clientes on-premises e o Amazon DynamoDB e entre o Amazon DynamoDB e outros recursos da AWS na mesma região da AWS.

Tópicos

- Criptografia em repouso do DynamoDB
- Proteger as conexões do DynamoDB usando endpoints da VPC e políticas do IAM

Criptografia em repouso do DynamoDB

Todos os dados de usuário armazenados no Amazon DynamoDB são totalmente criptografados em repouso. A criptografia em repouso do DynamoDB fornece segurança aprimorada ao criptografar seus dados em repouso usando chaves de criptografia armazenadas no AWS Key Management Service (AWS KMS). Essa funcionalidade ajuda a reduzir a carga e complexidade operacionais necessárias para proteger dados confidenciais. Com a criptografia de dados em repouso, você pode criar aplicativos confidenciais que atendem a requisitos rigorosos de conformidade e regulamentação de criptografia.

A criptografia em repouso do DynamoDB fornece uma camada adicional de proteção de dados sempre protegendo seus dados seguros uma tabela criptografada, incluindo a chave primária, os índices secundários local e global, os fluxos, as tabelas globais, os backups e os clusters do DynamoDB Accelerator (DAX) sempre que os dados são armazenados em mídia durável. Políticas organizacionais, regulamentações setoriais ou governamentais e exigências de conformidade geralmente demandam o uso de criptografia em repouso para aumentar a segurança de dados de seus aplicativos. Para obter mais informações sobre criptografia para aplicações de banco de dados, consulte AWS Database Encryption SDK.

A criptografia em repouso integra-se com o AWS KMS para o gerenciamento da chave de criptografia usada para criptografar suas tabelas. Para obter mais informações sobre os principais tipos e estados, consulte AWS Key Management Service concepts no Guia do desenvolvedor do AWS Key Management Service.

Ao criar uma nova tabela, você pode escolher um dos seguintes tipo de AWS KMS key para criptografar a tabela: É possível alternar entre esses tipos de chave a qualquer momento.

- Chave pertencente à AWS: tipo de criptografia padrão. A chave pertence ao DynamoDB (sem custo adicional).
- Chave gerenciada pela AWS: a chave é armazenada na sua conta e é gerenciada pelo AWS KMS (cobranças do AWS KMS são aplicáveis).
- Chave gerenciada pelo cliente: a chave é armazenada na sua conta e é você que a cria, detém e gerencia. Você tem controle total sobre a chave do KMS (cobranças do AWS KMS são aplicáveis).

Para obter mais informações sobre os tipos de chave, consulte Customer keys and AWS keys.

> **Note**
>
> - Na criação de um novo cluster do DAX com criptografia em repouso habilitada, uma Chave gerenciada pela AWS será usada para criptografar dados em repouso no cluster.
> - Se sua tabela tem uma chave de classificação, algumas dessas chaves que marcam os limites de intervalo são armazenadas em textos simples nos metadados da tabela.

Quando você acessa uma tabela criptografada, o DynamoDB descriptografa os dados da tabela de forma transparente. Você não precisa alterar seu código nem suas aplicações para usar ou gerenciar tabelas criptografadas. O DynamoDB continua a entregar a mesma latência em milissegundos de dígito único que você espera, e todas as consultas do DynamoDB funcionam perfeitamente em seus dados criptografados.

É possível especificar uma chave de criptografia quando você cria uma tabela ou alterna as chaves de criptografia em uma tabela existente usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API do Amazon DynamoDB. Para saber como, consulte Gerenciar tabelas criptografadas no DynamoDB.

A criptografia em repouso usando a Chave pertencente à AWS é oferecida sem custo adicional. No entanto, as cobranças do AWS KMS se aplicam a uma Chave gerenciada pela AWS e a uma chave gerenciada pelo cliente. Para obter mais informações sobre preços, consulte Preços do AWS KMS.

A criptografia em repouso do DynamoDB está disponível em todas as regiões da AWS, incluindo as regiões AWS China (Pequim), AWS China (Ningxia) e AWS GovCloud (EUA). Para obter mais informações, consulte Criptografia em repouso do DynamoDB: como funciona e Notas de uso da criptografia em repouso do DynamoDB.

Criptografia em repouso do DynamoDB: como funciona

A criptografia em repouso do Amazon DynamoDB criptografa seus dados usando Advanced Encryption Standard de 256 bits (AES-256), o que ajuda a proteger seus dados contra acesso não autorizado ao armazenamento subjacente.

A criptografia em repouso integra-se com o AWS Key Management Service (AWS KMS) para o gerenciamento da chave de criptografia usada para criptografar suas tabelas.

> **Note**
>
> Em maio de 2022, o AWS KMS alterou o cronograma de alternância para Chaves gerenciadas pela AWS a cada 3 anos (aproximadamente 1.095 dias) para a cada ano (aproximadamente 365 dias).
> Novas Chaves gerenciadas pela AWS são alternadas automaticamente um ano após serem criadas e aproximadamente a cada ano depois disso.
> As Chaves gerenciadas pela AWS existentes são alternadas automaticamente um ano após sua alternância mais recente e a cada ano depois disso.

Chaves pertencentes à AWS

As Chaves pertencentes à AWS não são armazenadas na sua conta da AWS. Elas fazem parte de um conjunto de chaves do KMS que a AWS detém e gerencia para serem usadas em várias contas da AWS. Os serviços da AWS podem usar Chaves pertencentes à AWS para proteger os dados. As Chaves pertencentes à AWS usadas pelo DynamoDB são alternadas a cada ano (cerca de 365 dias).

Você não pode visualizar, gerenciar ou usar Chaves pertencentes à AWS, nem auditar seu uso. No entanto, você não precisa fazer nenhum trabalho nem alterar nenhum programa para proteger as chaves que criptografam seus dados.

Não é cobrada taxa mensal nem taxa de uso para usar Chaves pertencentes à AWS, e elas não são contabilizadas com base nas cotas do AWS KMS para a sua conta.

Chaves gerenciadas pela AWS

Chaves gerenciadas pela AWS são chaves do KMS em sua conta que são criadas, gerenciadas e usadas em seu nome por um produto da AWS integrado ao AWS KMS. Você pode visualizar as Chaves gerenciadas pela AWS na sua conta, visualizar suas políticas de chaves e auditar o uso delas em logs do AWS CloudTrail. No entanto, não é possível gerenciar essas chaves do KMS nem alterar suas permissões.

A criptografia em repouso integra-se automaticamente com o AWS KMS para o gerenciamento das Chaves gerenciadas pela AWS para o DynamoDB (`aws/dynamodb`), usadas para criptografar suas tabelas. Se uma Chave gerenciada pela AWS não existir quando você criar sua tabela do DynamoDB criptografada, o AWS KMS criará automaticamente uma nova chave para você. Essa chave é usada com tabelas criptografados que são criados no futuro. O AWS KMS integra hardware

e software seguros e altamente disponíveis para oferecer um sistema de gerenciamento de chaves escalonado para a nuvem.

Para obter mais informações sobre como gerenciar as permissões de Chave gerenciada pela AWS, consulte Autorizar uso de Chave gerenciada pela AWS no Guia do Desenvolvedor do AWS Key Management Service.

Chaves gerenciadas pelo cliente

Chaves gerenciadas pelo cliente são chaves do KMS disponíveis na sua conta do AWS que você cria, detém e gerencia. Você tem controle total sobre essas chaves do KMS, inclusive para estabelecer e manter as políticas de chaves, as políticas do IAM e as concessões; habilitar e desabilitar as chaves do KMS; alternar seu material de criptografia; adicionar tags; criar aliases que fazem referência a elas; e programar a exclusão delas. Para ter mais informações sobre como gerenciar as permissões de uma chave gerenciada pelo cliente, consulte Chaves gerenciadas pelo cliente.

Quando você especifica uma chave gerenciada pelo cliente como a chave de criptografia em nível de tabela, a tabela, os índices secundários local e global e os fluxos do DynamoDB são criptografados com a mesma chave gerenciada pelo cliente. Os backups sob demanda são criptografados com a chave de criptografia no nível da tabela que é especificada no momento da criação do backup. A atualização da chave de criptografia no nível da tabela não altera a chave de criptografia associada aos backups sob demanda existentes.

Definir o estado da chave gerenciada pelo cliente como desabilitada ou programá-la para exclusão evita que todos os usuários e o serviço DynamoDB possam criptografar ou descriptografar dados e realizar operações de leitura e gravação na tabela. O DynamoDB deve ter acesso à sua chave de criptografia para garantir que você possa continuar acessando sua tabela e para evitar a perda de dados.

Se você desabilitar a chave gerenciada pelo cliente ou programá-la para exclusão, o status da tabela passará para Inaccessible (Inacessível). Para garantir que você possa continuar trabalhando com a tabela, é necessário conceder ao DynamoDB acesso à chave de criptografia especificada em até sete dias. Assim que o serviço detectar que a chave de criptografia está inacessível, o DynamoDB enviará uma notificação por e-mail para alertar você.

> **Note**
>
> - Se a chave gerenciada pelo cliente permanecer inacessível para o serviço DynamoDB por mais de sete dias, a tabela será arquivada e não poderá mais ser acessada. O DynamoDB criará um backup sob demanda da tabela e você será cobrado por ele. É possível usar esse backup sob demanda a fim de restaurar seus dados para uma nova tabela. Para iniciar a restauração, a última chave gerenciada pelo cliente na tabela deve estar habilitada e o DynamoDB deve ter acesso a ela.
> - Se a chave gerenciada pelo cliente que foi usada para criptografar uma réplica de tabela global estiver inacessível, o DynamoDB removerá essa réplica do grupo de replicação. A réplica não será excluída e a replicação será interrompida de e para essa região 20 horas após detectar que a chave gerenciada pelo cliente está inacessível.

Para obter mais informações, consulte habilitar chaves e excluir chaves.

Observações sobre o uso de Chaves gerenciadas pela AWS

O Amazon DynamoDB não consegue ler os dados da tabela, a menos que tenha acesso à chave do KMS armazenada em sua conta da AWS KMS. O DynamoDB usa criptografia envelopada e hierarquia de chaves para criptografar dados. A chave de criptografia do AWS KMS é usada para criptografar a chave raiz dessa hierarquia de chaves. Para obter mais informações, consulte Criptografia envelopada no Guia do desenvolvedor do AWS Key Management Service.

O DynamoDB não chama AWS KMS para cada operação do DynamoDB. A chave é atualizada uma vez a cada cinco minutos por chamador com tráfego ativo.

Verifique se configurou o SDK para reutilizar conexões. Caso contrário, você experimentará latências do DynamoDB tendo que reestabelecer novas entradas de cache do AWS KMS para cada operação do DynamoDB. Além disso, talvez seja necessário enfrentar custos mais altos do AWS KMS e do CloudTrail. Por exemplo, para fazer isso usando SDK de Node.js, você pode criar um novo agente HTTPS com `keepAlive` ativado. Para obter mais informações, consulte Configurar keepAlive em Node.js no Guia do desenvolvedor do AWS SDK para JavaScript.

Notas de uso da criptografia em repouso do DynamoDB

Considere o seguinte ao usar criptografia em repouso no Amazon DynamoDB:

Todos os dados da tabela são criptografados

A criptografia em repouso no lado do servidor está habilitada em todos os dados de tabelas do DynamoDB e não pode ser desabilitada. Não é possível criptografar apenas um subconjunto de itens em uma tabela.

A criptografia em repouso só criptografa dados enquanto eles estão estáticos (em repouso) em uma mídia de armazenamento persistente. Se a segurança dos dados for motivo de preocupação para dados em trânsito ou dados em uso, talvez seja necessário tomar outras medidas:

- Dados em trânsito: todos os dados no DynamoDB são criptografados em trânsito. Por padrão, as comunicações com o DynamoDB usam o protocolo HTTPS, o qual protege o tráfego de rede usando a criptografia Secure Sockets Layer (SSL)/Transport Layer Security (TLS).
- Dados em uso: proteja seus dados antes de enviá-los ao DynamoDB usando criptografia do lado do cliente. Para obter mais informações, consulte Criptografia do lado do cliente e do lado do servidor no Guia do desenvolvedor do Amazon DynamoDB Encryption Client.

É possível pode usar fluxos com tabelas criptografadas. Os fluxos do DynamoDB são sempre criptografados com uma chave de criptografia em nível de tabela. Para obter mais informações, consulte Capturar dados de alterações para o DynamoDB Streams.

Os backups do DynamoDB são criptografados, e a tabela que é restaurada de um backup também tem a criptografia habilitada. É possível usar a Chave pertencente à AWS, a Chave gerenciada pela AWS ou a chave gerenciada pelo cliente para criptografar seus dados de backup. Para obter mais informações, consulte Backup e restauração para o DynamoDB.

Os índices secundários locais e os índices secundários globais são criptografados usando a mesma chave da tabela-base.

Tipos de criptografia

> **Note**
>
> As chaves gerenciadas pelo cliente não são compatíveis com a tabela global versão 2017. Se quiser usar uma chave gerenciada pelo cliente em uma tabela global do DynamoDB, você deverá atualizar a tabela para a Tabela global versão 2019 e habilitá-la.

No AWS Management Console, o tipo de criptografia é `KMS` quando você usa a Chave gerenciada pela AWS ou a chave gerenciada pelo cliente para criptografar seus dados. O tipo de criptografia é `DEFAULT` quando você usa a Chave pertencente à AWS. Na API do Amazon DynamoDB, o tipo de criptografia é `KMS` quando a Chave gerenciada pela AWS ou a chave gerenciada pelo cliente é usada. Na ausência do tipo de criptografia, seus dados são criptografados usando a Chave pertencente à AWS. É possível alternar entre a Chave pertencente à AWS, a Chave gerenciada pela AWS e a chave gerenciada pelo cliente a qualquer momento. Você pode usar o console, a AWS Command Line Interface (AWS CLI) ou a API do Amazon DynamoDB para alternar as chaves de criptografia.

Observe as seguintes limitações ao usar chaves gerenciadas pelo cliente:

- Não é possível usar uma chave gerenciada pelo cliente com clusters do DynamoDB Accelerator (DAX). Para obter mais informações, consulte Criptografia em repouso do DAX .
- É possível usar uma chave gerenciada pelo cliente para criptografar tabelas que usam transações. No entanto, para garantir a durabilidade da propagação das transações, uma cópia da solicitação da transação é temporariamente armazenada pelo serviço e criptografada usando uma Chave pertencente à AWS. Os dados confirmados em suas tabelas e índices secundários são sempre criptografados em repouso usando a chave gerenciada pelo cliente.
- É possível usar uma chave gerenciada pelo cliente para criptografar tabelas que usam o Contributor Insights. No entanto, os dados transmitidos ao Amazon CloudWatch são criptografados com uma Chave pertencente à AWS.
- Ao fazer a transição para uma nova chave gerenciada pelo cliente, mantenha a chave original ativada até que o processo seja concluído. O AWS ainda precisará da chave original para descriptografar os dados antes de criptografá-los com a nova chave. O processo será concluído quando o Status SSEDescription da tabela estiver HABILITADO e o KMSMasterKeyArn da nova chave gerenciada pelo cliente for exibido. Nesse momento, a chave original pode ser desativada ou programada para exclusão.
- Depois que a nova chave gerenciada pelo cliente é exibida, a tabela e quaisquer novos backups sob demanda são criptografados com a nova chave.
- Todos os backups sob demanda existentes permanecem criptografados com a chave gerenciada pelo cliente que foi usada quando esses backups foram criados. Você precisará dessa mesma chave para restaurar os backups. Você pode identificar a chave para o período em que cada backup foi criado usando a API DescribeBackup para exibir a SSEDescription desse backup.

- Se você desabilitar sua chave gerenciada pelo cliente ou programá-la para exclusão, qualquer dado no DynamoDB Streams ainda estará sujeito a uma vida útil de 24 horas. Qualquer dado de atividade não recuperado é elegível para remoção quando tiver mais de 24 horas.
- Se você desabilitar sua chave gerenciada pelo cliente ou programá-la para exclusão, as exclusões por vida útil (TTL) continuarão por 30 minutos. Essas exclusões do TTL continuarão a ser emitidas ao DynamoDB Streams e estarão sujeitas ao intervalo de retenção/remoção padrão.

  Para obter mais informações, consulte habilitar chaves e excluir chaves.

Usar chaves do KMS e chaves de dados

O recurso de criptografia em repouso do DynamoDB usa uma AWS KMS key e uma hierarquia de chaves de dados para proteger os dados da sua tabela. O DynamoDB usa a mesma hierarquia de chaves para proteger fluxos do DynamoDB, tabelas globais e backups quando eles são gravados em mídia durável.

Recomendamos que você planeje sua estratégia de criptografia antes de implementar sua tabela no DynamoDB. Se você armazenar dados confidenciais no DynamoDB, considere incluir a criptografia do lado do cliente em seu plano. Dessa forma, você poderá criptografar os dados o mais próximo possível de sua origem e garantir sua proteção durante todo o ciclo de vida. Para obter mais informações, consulte a documentação Cliente de criptografia do DynamoDB.

AWS KMS key

A criptografia em repouso protege suas tabelas do DynamoDB em uma AWS KMS key. Por padrão, o DynamoDB usa uma Chave pertencente à AWS, uma chave de criptografia multilocatário que é criada e gerenciada em uma conta de serviço do DynamoDB. Porém, você pode criptografar suas tabelas do DynamoDB em uma chave gerenciada pelo cliente para o DynamoDB (`aws/dynamodb`) na sua Conta da AWS. Você pode selecionar uma chave do KMS diferente para cada tabela. A chave do KMS selecionada para uma tabela também é usada para criptografar índices secundários locais e globais, fluxos e backups.

Você seleciona a chave do KMS para uma tabela ao criar ou atualizar essa tabela. É possível alterar a chave do KMS de uma tabela a qualquer momento, seja no console do DynamoDB ou usando a operação UpdateTable. O processo de alternar chaves é transparente e não exige tempo de inatividade ou serviço de degradação.

> **Important**
>
> O DynamoDB oferece suporte somente para chaves do KMS simétricas. Não é possível usar uma chave do KMS assimétrica para criptografar tabelas do DynamoDB.

Use uma chave gerenciada pelo cliente para obter os seguintes recursos:

- Você cria e gerencia a chave do KMS, incluindo a configuração de políticas de chaves, políticas do IAM e concessões para controlar o acesso à chave do KMS. Você pode habilitar e desabilitar a chave do KMS, habilitar e desabilitar a alternância automática de chaves e excluir a chave do KMS quando ela não estiver mais em uso.
- Você pode usar uma chave gerenciada pelo cliente com material de chave importado ou uma chave gerenciada pelo cliente em um armazenamento de chaves personalizado que você possui e gerencia.
- Você pode auditar a criptografia e a descriptografia da sua tabela do DynamoDB examinando as chamadas de API do DynamoDB para o AWS KMS em logs do AWS CloudTrail.

Use a Chave gerenciada pela AWS se precisar de qualquer um dos seguintes recursos:

- Você pode visualizar a chave do KMS e sua política de chaves. (Você não pode alterar a política de chave.)
- Você pode auditar a criptografia e a descriptografia da sua tabela do DynamoDB examinando as chamadas de API do DynamoDB para o AWS KMS em logs do AWS CloudTrail.

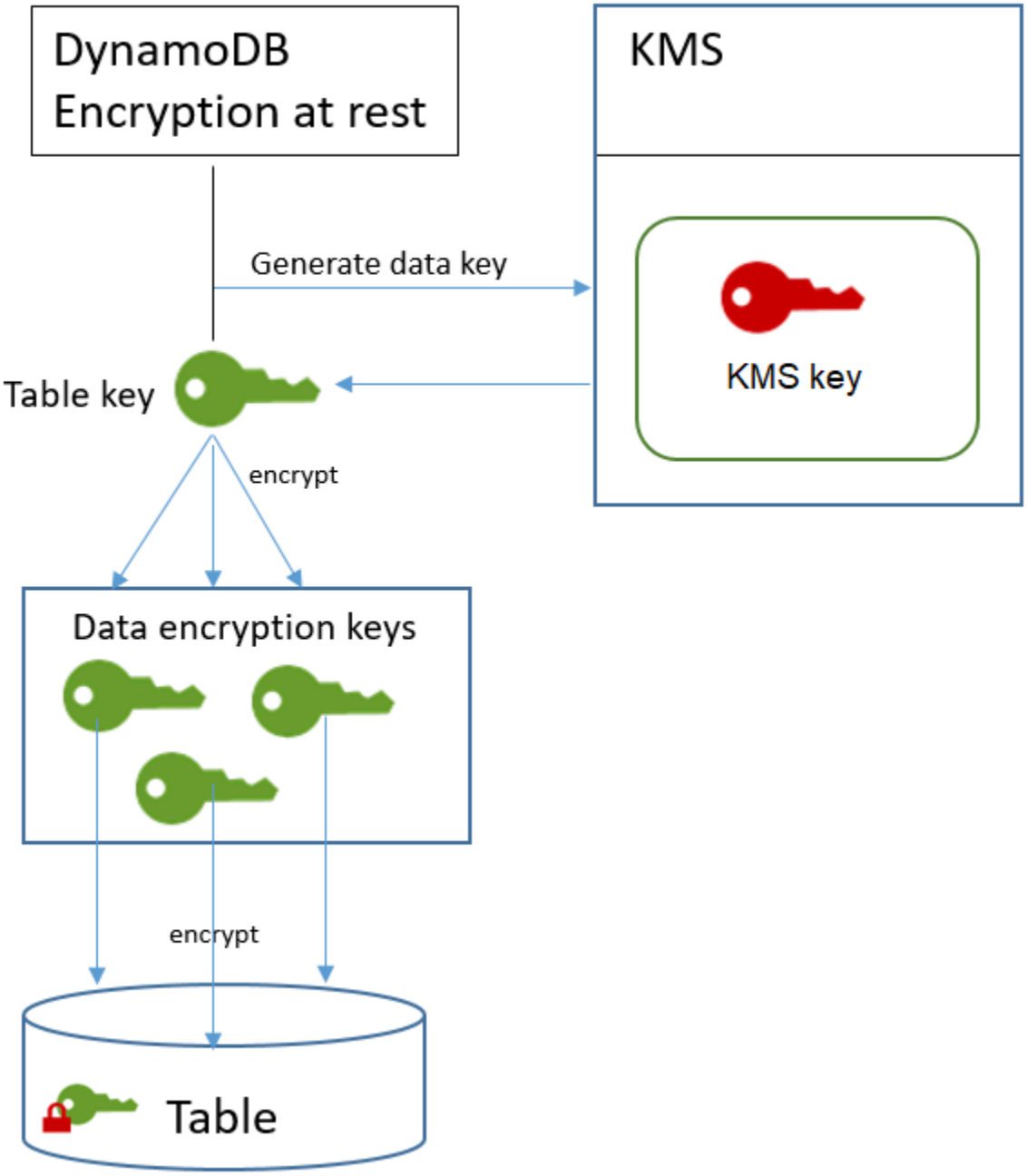
No entanto, a Chave pertencente à AWS é gratuita e seu uso não conta para cotas de solicitações ou de recursos do AWS KMS. As chaves gerenciadas pelo cliente e as Chaves gerenciadas pela AWS geram uma cobrança para cada chamada de API, e as cotas do AWS KMS são aplicáveis a essas chaves do KMS.

Chaves de tabela

O DynamoDB usa a chave do KMS para a tabela para gerar e criptografar uma chave de dados exclusiva para a tabela, conhecida como chave de tabela. A chave de tabela é mantida durante a vida útil da tabela criptografada.

A chave de tabela é usada como uma chave de criptografia de chaves. O DynamoDB usa essa chave de tabela para proteger as chaves de criptografia dos dados usadas para criptografar os dados da tabela. O DynamoDB gera uma chave de criptografia dos dados exclusiva para cada

estrutura subjacente em uma tabela, mas vários itens de tabela podem ser protegidos com a mesma chave de criptografia dos dados.



Quando você acessa uma tabela criptografada pela primeira vez, o DynamoDB envia uma solicitação para o AWS KMS usar a chave do KMS para descriptografar a chave de tabela. Ele usa a chave de tabela de texto simples para descriptografar as chaves de criptografia dos dados

e usa as chaves de criptografia dos dados de texto simples para descriptografar os dados da tabela.

O DynamoDB armazena e usa a chave de tabelas e as chaves de criptografia dos dados fora do AWS KMS. Ele protege todas as chaves com a criptografia Advanced Encryption Standard (AES) e chaves de criptografia de 256 bits. Armazena as chaves criptografadas com os dados criptografados para que estejam disponíveis para descriptografar os dados da tabela sob demanda.

Se você alterar a chave do KMS da tabela, o DynamoDB gerará uma nova chave de tabela. Ele usará a nova chave de tabela para criptografar novamente as chaves de criptografia dos dados.

Armazenamento em cache de chaves de tabela

Para evitar chamar o AWS KMS para cada operação do DynamoDB, o DynamoDB armazena em cache chaves de tabela em texto simples para cada chamador na memória. Quando o DynamoDB recebe uma solicitação para a chave de tabela armazenada em cache após cinco minutos de inatividade, ele envia uma nova solicitação ao AWS KMS para descriptografar a chave de tabela. Essa chamada capturará todas as alterações feitas nas políticas de acesso da chave do KMS no AWS KMS ou no AWS Identity and Access Management (IAM) desde a última solicitação para descriptografar a chave de tabela.

Autorizar o uso da sua chave do KMS

Se você usar uma chave gerenciada pelo cliente ou a Chave gerenciada pela AWS na sua conta para proteger a tabela do DynamoDB, as políticas nessa chave do KMS deverão conceder ao DynamoDB permissão para usá-la em seu nome. O contexto de autorização na Chave gerenciada pela AWS para o DynamoDB inclui sua política de chaves e concessões que delegam permissões para usá-la.

Você tem controle total sobre as políticas e concessões em uma chave gerenciada pelo cliente. Como a Chave gerenciada pela AWS está na sua conta, você pode visualizar suas políticas e concessões. Porém, como ela é gerenciada pela AWS, você não pode alterar as políticas.

O DynamoDB não precisa de autorização adicional para usar a Chave pertencente à AWS padrão para proteger as tabelas do DynamoDB na sua Conta da AWS.

Tópicos

- Política de chaves para uma Chave gerenciada pela AWS
- Política de chaves para uma chave gerenciada pelo cliente

- Usar concessões para autorizar o DynamoDB

Política de chaves para uma Chave gerenciada pela AWS

Quando o DynamoDB usa a Chave gerenciada pela AWS para o DynamoDB (`aws/dynamodb`) em operações de criptografia, ele faz isso em nome do usuário que está acessando o recurso do DynamoDB. A política de chaves na Chave gerenciada pela AWS concede a todos os usuários na conta permissão para usar a Chave gerenciada pela AWS para operações especificadas. Porém, a permissão é concedida somente quando o DynamoDB faz a solicitação em nome do usuário. A condição ViaService na política de chaves não permite que nenhum usuário use a Chave gerenciada pela AWS, a menos que a solicitação seja proveniente do serviço DynamoDB.

Essa política de chaves, como as políticas de todas as Chaves gerenciadas pela AWS, é estabelecida pela AWS. Você não pode alterá-la, mas pode visualizá-la a qualquer momento. Para obter mais detalhes, consulte Visualizar uma política de chaves.

As declarações de política na política de chaves têm os seguintes efeitos:

- Permita que os usuários na conta usem a Chave gerenciada pela AWS para o DynamoDB em operações de criptografia somente quando essa solicitação for proveniente do DynamoDB em seu nome. A política também permite que os usuários criem concessões para a chave do KMS.
- Permite identidades autorizadas do IAM na conta visualizem as propriedades da Chave gerenciada pela AWS para o DynamoDB e revoguem a concessão que permite ao DynamoDB usar a chave do KMS. O DynamoDB usa concessões para operações de manutenção em andamento.
- Permite que o DynamoDB execute operações somente leitura para encontrar a Chave gerenciada pela AWS para o DynamoDB na sua conta.

JSON

```
{
  "Version" : "2012-10-17"       ,
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
```

```
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*",
 "kms:GenerateDataKey*", "kms:CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow DynamoDB Service with service principal name
 dynamodb.amazonaws.com to describe the key directly",
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "dynamodb.amazonaws.com"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
    "Resource" : "*"
  } ]
}
```

Política de chaves para uma chave gerenciada pelo cliente

Ao escolher uma chave gerenciada pelo cliente para proteger uma tabela do DynamoDB, o DynamoDB obtém permissão para usar a chave do KMS em nome da entidade principal que faz a seleção. Essa entidade principal, um usuário ou uma função, deve ter as permissões em uma chave do KMS exigida pelo DynamoDB. É possível fornecer essas permissões em uma política de chaves, em uma política do IAM ou em uma concessão.

No mínimo, o DynamoDB exige as seguintes permissões em uma chave gerenciada pelo cliente:

- kms:Encrypt

- kms:Decrypt
- kms:ReEncrypt* (para kms:ReEncryptFrom e kms:ReEncryptTo)
- kms:GenerateDataKey* (para kms:GenerateDataKey e kms:GenerateDataKeyWithoutPlaintext)
- kms:DescribeKey
- kms:CreateGrant

Por exemplo, o exemplo de política de chaves a seguir fornece somente as permissões necessárias. A política tem os seguintes efeitos:

- Permite que o DynamoDB use a chave do KMS em operações de criptografia e crie concessões, mas somente quando está atuando em nome de entidades principais na conta que tem permissão para usar o DynamoDB. Se as entidades principais especificadas na instrução de política não tiverem permissão para usar o DynamoDB, a chamada falhará, mesmo se vier do serviço do DynamoDB.
- A chave de condição kms:ViaService concede as permissões somente quando a solicitação é proveniente do DynamoDB em nome das entidades principais listadas na instrução da política. Essas entidades principais não podem chamar essas operações diretamente. Observe que o valor de `kms:ViaService`, `dynamodb.*.amazonaws.com`, tem um asterisco (*) na posição da região. O DynamoDB requer a permissão para ser independente de qualquer Região da AWS particular, para que ele possa fazer chamadas entre regiões como suporte a tabelas globais do DynamoDB.
- Concede aos administradores da chave do KMS (usuários que podem assumir a função `db-team`) acesso somente leitura à chave do KMS e permissão para revogar concessões, incluindo as concessões exigidas pelo DynamoDB para proteger a tabela.

Antes de usar um exemplo de política de chaves, substitua o exemplo de entidades principais por entidades principais reais da sua conta da Conta da AWS.

JSON

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17"       ,
  "Statement": [
    {
```

```
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the KMS key and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

Usar concessões para autorizar o DynamoDB

Além de políticas de chaves, o DynamoDB usa concessões para definir permissões em uma chave gerenciada pelo cliente ou na Chave gerenciada pela AWS para o DynamoDB (`aws/dynamodb`). Para visualizar as concessões em uma chave do KMS na sua conta, use a operação ListGrants. O

DynamoDB não precisa de concessões ou permissões adicionais para usar a Chave pertencente à AWS para proteger sua tabela.

O DynamoDB usa as permissões de concessão ao executar manutenção do sistema e tarefas de proteção de dados contínua em segundo plano. Usa também concessões para gerar chaves de tabela.

Cada concessão é específica a uma tabela. Se a conta incluir várias tabelas criptografadas na mesma chave do KMS, haverá uma concessão de cada tipo para cada tabela. A concessão é restrita pelo contexto de criptografia do DynamoDB, que inclui o nome da tabela e o ID da Conta da AWS, bem como a permissão para retirar a concessão caso ela não seja mais necessária.

Para criar as concessões, o DynamoDB deve ter permissão para chamar `CreateGrant` em nome do usuário que criou a tabela criptografada. Para Chaves gerenciadas pela AWS, o DynamoDB recebe a permissão `kms:CreateGrant` da política de chaves, o que permite que os usuários da conta chamem CreateGrant na chave do KMS somente quando o DynamoDB faz a solicitação em nome de um usuário autorizado.

A política de chaves também pode permitir que a conta revogue a concessão na chave do KMS. No entanto, se você revogar a concessão em uma tabela criptografada ativa, o DynamoDB não poderá proteger e manter a tabela.

Contexto de criptografia do DynamoDB

Um contexto de criptografia é um conjunto de pares de chave-valor que contêm dados arbitrários não secretos. Quando você inclui um contexto de criptografia em uma solicitação para criptografar dados, o AWS KMS vincula de forma criptográfica o contexto de criptografia aos dados criptografados. Para descriptografar os dados, você deve passar o mesmo contexto de criptografia.

O DynamoDB usa o mesmo contexto de criptografia em todas as operações de criptografia do AWS KMS. Se você usar uma chave gerenciada pelo cliente ou uma Chave gerenciada pela AWS para proteger a tabela do DynamoDB, poderá usar o contexto de criptografia para identificar o uso da chave do KMS em logs e registros de auditoria. Ele também aparece em texto simples em logs, como o AWS CloudTrail e o Amazon CloudWatch Logs.

O contexto de criptografia também pode ser usado como uma condição para autorização em políticas e concessões. O DynamoDB usa o contexto de criptografia para restringir as concessões que permitem acesso à chave gerenciada pelo cliente ou à Chave gerenciada pela AWS na sua conta e região.

Em suas solicitações para o AWS KMS, o DynamoDB usa um contexto de criptografia com dois pares de chave-valor.

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- Tabela – O primeiro par de chave-valor identifica a tabela que o DynamoDB está criptografando. A chave é `aws:dynamodb:tableName`. O valor é o nome da tabela.

  ```
  "aws:dynamodb:tableName": "<table-name>"
  ```

  Por exemplo:

  ```
  "aws:dynamodb:tableName": "Books"
  ```

- Conta – O segundo par de chave-valor identifica a Conta da AWS. A chave é `aws:dynamodb:subscriberId`. O valor é o ID de conta.

  ```
  "aws:dynamodb:subscriberId": "<account-id>"
  ```

  Por exemplo:

  ```
  "aws:dynamodb:subscriberId": "111122223333"
  ```

Monitoramento da interação do DynamoDB com o AWS KMS

Se você usa uma chave gerenciada pelo cliente ou uma Chave gerenciada pela AWS para proteger suas tabelas do DynamoDB, você poderá usar logs do AWS CloudTrail para rastrear as solicitações que o DynamoDB envia ao AWS KMS em seu nome.

As solicitações `GenerateDataKey`, `Decrypt` e `CreateGrant` são discutidas nesta seção. Além disso, o DynamoDB usa uma operação DescribeKey para determinar se a chave do KMS escolhida existe na conta e na região. Usa também uma operação RetireGrant para remover uma concessão quando você exclui uma tabela.

GenerateDataKey

Quando você habilita a criptografia em repouso em uma tabela, o DynamoDB cria uma chave de tabela exclusiva. Ele envia uma solicitação GenerateDataKey ao AWS KMS que especifica a chave do KMS para a tabela.

O evento que registra a operação `GenerateDataKey` é semelhante ao evento de exemplo a seguir. O usuário é a conta de serviço do DynamoDB. Os parâmetros incluem o Amazon Resource Name (ARN) da chave do KMS, um especificador de chave que requer uma chave de 256 bits e o contexto de criptografia que identifica a tabela e a conta da Conta da AWS.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dynamodb:tableName": "Services",
            "aws:dynamodb:subscriberId": "111122223333"
        },
        "keySpec": "AES_256",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "229386c1-111c-11e8-9e21-c11ed5a52190",
    "eventID": "e3c436e9-ebca-494e-9457-8123a1f5e979",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
```

```
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "bf915fa6-6ceb-4659-8912-e36b69846aad"
}
```

Decrypt

Quando você acessa uma tabela criptografada do DynamoDB, o DynamoDB precisa descriptografar a chave da tabela para que possa descriptografar as chaves abaixo dela na hierarquia. Descriptografa os dados na tabela. Para descriptografar a chave da tabela. O DynamoDB envia uma solicitação Decrypt solicitação ao AWS KMS que especifica a chave do KMS para a tabela.

O evento que registra a operação `Decrypt` é semelhante ao evento de exemplo a seguir. O usuário é a entidade principal na sua Conta da AWS que está acessando a tabela. Os parâmetros incluem a chave de tabela criptografada (como um blob de texto cifrado) e o contexto de criptografia que identifica a tabela e a Conta da AWS. O AWS KMS deriva o ID da chave do KMS do texto cifrado.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T16:42:15Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDT3HGFQZX4RY6RU",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
```

```
    "eventTime": "2018-02-14T16:42:39Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters":
    {
        "encryptionContext":
        {
            "aws:dynamodb:tableName": "Books",
            "aws:dynamodb:subscriberId": "111122223333"
        }
    },
    "responseElements": null,
    "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
    "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

CreateGrant

Quando você usa uma chave gerenciada pelo cliente ou uma Chave gerenciada pela AWS para proteger sua tabela do DynamoDB, o DynamoDB usa concessões para permitir que o serviço execute a proteção de dados e tarefas de manutenção e durabilidade contínuas. Essas concessões não são necessárias em Chave pertencente à AWS.

As concessões que o DynamoDB cria são específicas de uma tabela. A entidade principal na solicitação CreateGrant é o usuário que criou a tabela.

O evento que registra a operação `CreateGrant` é semelhante ao evento de exemplo a seguir. Os parâmetros incluem o Amazon Resource Name (ARN) da chave do KMS para a tabela, a entidade principal favorecida e a entidade principal que está sendo retirada (o serviço

DynamoDB) e as operações que a concessão abrange. Incluem também uma restrição que requer que todas as operações de criptografia usem o contexto de criptografia.


```
{
    "eventVersion": "1.05",
    "userIdentity":
    {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T00:12:02Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:dynamodb:tableName": "Books",
                "aws:dynamodb:subscriberId": "111122223333"
            }
        },
        "granteePrincipal": "dynamodb.us-west-2.amazonaws.com",
```

```
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
        ]
    },
    "responseElements": {
        "grantId":
 "5c5cd4a3d68e65e77795f5ccc2516dff057308172b0cd107c85b5215c6e48bde"
    },
    "requestID": "2192b82a-111c-11e8-a528-f398979205d8",
    "eventID": "a03d65c3-9fee-4111-9816-8bf96b73df01",
    "readOnly": false,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


Gerenciar tabelas criptografadas no DynamoDB

É possível usar o AWS Management Console ou a AWS Command Line Interface (AWS CLI) para especificar a chave de criptografia em novas tabelas e atualizar as chaves de criptografia em tabelas existentes no Amazon DynamoDB.

Tópicos

- Especificar a chave de criptografia para uma nova tabela
- Atualizar uma chave de criptografia

Especificar a chave de criptografia para uma nova tabela

Siga estas etapas para especificar a chave de criptografia em uma nova tabela usando o console do Amazon DynamoDB ou a AWS CLI.

Criar uma tabela criptografada (console)

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Selecione Create Table (Criar tabela). Para o Table name (Nome da tabela), insira **`Music`**. Para a chave primária, insira **`Artist`**. Para a chave de classificação, insira **`SongTitle`**, os dois como strings.
4. Em Settings (Configurações), verifique se Customize Settings (Personalizar configurações) está selecionado.

   > **Note**
   >
   > Se a opção Use default settings (Usar configurações padrão) estiver selecionada, as tabelas serão criptografadas em repouso com Chave pertencente à AWS sem custo adicional.

5. Em Encryption at rest (Criptografia em repouso), escolha um tipo de criptografia: Chave pertencente à AWS, Chave gerenciada pela AWS, ou chave gerenciada pelo cliente.

   - Owned by Amazon DynamoDB (Propriedade do Amazon DynamoDB) chave de propriedade da AWS, pertencente e gerenciada especificamente pelo DynamoDB. Não há custo adicional para usar essa chave.
   - Chave gerenciada pela AWS. Alias da chave: `aws/dynamodb`. A chave é armazenada na sua conta e é gerenciada pelo AWS Key Management Service (AWS KMS). Cobranças do AWS KMS são aplicáveis).
   - Stored in your account, and owned and managed by you. (Armazenada em sua conta, de sua propriedade e gerenciada por você.) Chave gerenciada pelo cliente A chave é armazenada na sua conta e é gerenciada pelo AWS Key Management Service (AWS KMS). Cobranças do AWS KMS são aplicáveis).

> **Note**
>
> Se você optar por ser o proprietário e gerenciar sua própria chave, certifique-se de que a Política de chaves do KMS está definida corretamente. Para obter mais informações, consulte Política de chaves para uma chave gerenciada pelo cliente.

6. Selecione Create (Criar) para criar a tabela criptografada. Para confirmar o tipo de criptografia, selecione os detalhes da tabela na guia Overview (Visão geral) e revise a seção Additional details (Detalhes adicionais).

Criar uma tabela criptografada (AWS CLI)

Use a AWS CLI para criar uma tabela com a Chave pertencente à AWS padrão, com a Chave gerenciada pela AWS ou com a chave gerenciada pelo cliente para o Amazon DynamoDB.

Para criar uma tabela criptografada com a Chave pertencente à AWS padrão

- Crie a tabela `Music` criptografada da seguinte forma:

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5
```

> **Note**
>
> Essa tabela agora está criptografada com a Chave pertencente à AWS padrão na conta de serviço do DynamoDB.

Para criar uma tabela criptografada com a Chave gerenciada pela AWS para o DynamoDB

- Crie a tabela `Music` criptografada da seguinte forma:

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS
```

O status `SSEDescription` da descrição da tabela é definido como `ENABLED`, e o `SSEType` é `KMS`:

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Para criar uma tabela criptografada com uma chave gerenciada pelo cliente para o DynamoDB

- Crie a tabela `Music` criptografada da seguinte forma:

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

O status `SSEDescription` da descrição da tabela é definido como `ENABLED`, e o `SSEType` é `KMS`:

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Atualizar uma chave de criptografia

Também é possível usar o console do DynamoDB ou a AWS CLI para atualizar as chaves de criptografia de uma tabela existente entre uma Chave pertencente à AWS, uma Chave gerenciada pela AWS e uma chave gerenciada pelo cliente a qualquer momento.

Atualizar uma chave de criptografia (console)

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Escolha a tabela que você deseja atualizar.
4. Selecione Actions (Ações) e depois selecione a opção Update settings (Atualizar configurações).
5. Vá para a guia Additional settings (Configurações adicionais).
6. Em Encryption (Criptografia), escolha Manage encryption (Gerenciar criptografia).
7. Escolha um tipo de criptografia:

   - Propriedade do Amazon DynamoDB. A chave AWS KMS pertence e é gerenciada pelo DynamoDB. Não há custo adicional para usar essa chave.
   - Alias de chave da chave gerenciada da AWS: `aws/dynamodb`. A chave é armazenada na sua conta e é gerenciada pelo AWS Key Management Service (AWS KMS). Cobranças do AWS KMS são aplicáveis.
   - Stored in your account, and owned and managed by you(Armazenada em sua conta, de sua propriedade e gerenciada por você). A chave é armazenada na sua conta e é gerenciada pelo AWS Key Management Service (AWS KMS). Cobranças do AWS KMS são aplicáveis.

> **Note**
>
> Se você optar por ser o proprietário e gerenciar sua própria chave, certifique-se de que a Política de chaves do KMS está definida corretamente. Para obter mais informações, consulte Política de chaves para uma chave gerenciada pelo cliente.

Depois escolha Save (Salvar) para atualizar a tabela criptografada. Para confirmar o tipo de criptografia, verifique os detalhes da tabela na guia Overview (Visão geral).

Atualizar uma chave de criptografia (AWS CLI)

Os exemplos a seguir mostram como atualizar uma tabela de criptografia usando a AWS CLI.

Para atualizar uma tabela criptografada com a Chave pertencente à AWS padrão

- Atualize a tabela `Music`, como o exemplo a seguir.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> **Note**
>
> Essa tabela agora está criptografada com a Chave pertencente à AWS padrão na conta de serviço do DynamoDB.

Para atualizar uma tabela criptografada com a Chave gerenciada pela AWS para o DynamoDB

- Atualize a tabela `Music`, como o exemplo a seguir.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

O status `SSEDescription` da descrição da tabela é definido como `ENABLED`, e o `SSEType` é `KMS`:

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Para atualizar uma tabela criptografada com uma chave gerenciada pelo cliente para o DynamoDB

- Atualize a tabela `Music`, como o exemplo a seguir.

  ```
  aws dynamodb update-table \
    --table-name Music \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
  a123-ab1234a1b234
  ```

  O status `SSEDescription` da descrição da tabela é definido como `ENABLED`, e o `SSEType` é `KMS`:

  ```
  "SSEDescription": {
    "SSEType": "KMS",
    "Status": "ENABLED",
    "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
  a123-ab1234a1b234",
  }
  ```

Proteger as conexões do DynamoDB usando endpoints da VPC e políticas do IAM

As conexões são protegidas entre o Amazon DynamoDB e as aplicações on-premises e entre o DynamoDB e outros recursos da AWS na mesma região da AWS.

Política necessária para endpoints

O Amazon DynamoDB fornece uma API DescribeEndpoints que permite enumerar informações de endpoints regionais. Para solicitações aos endpoints públicos do DynamoDB, a API responde independentemente da política do IAM configurada para o DynamoDB, mesmo que haja uma

negação explícita ou implícita na política do IAM ou do endpoint da VPC. Isso ocorre porque o DynamoDB ignora intencionalmente a autorização para a API `DescribeEndpoints`.

Para solicitações de um endpoint da VPC, as políticas de endpoint do IAM e da nuvem privada virtual (VPC) devem autorizar a chamada de API `DescribeEndpoints` para as entidades principais do Identity and Access Management (IAM) solicitantes usando a ação do IAM `dynamodb:DescribeEndpoints`. Caso contrário, o acesso à API `DescribeEndpoints` será negado.

Veja a seguir um exemplo de uma política de endpoints.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

Tráfego entre clientes de serviço e on-premises e as aplicações

Você tem duas opções de conectividade entre sua rede privada e a AWS:

- Uma conexão do AWS Site-to-Site VPN. Para obter mais informações, consulte O que é o AWS Site-to-Site VPN? no Guia do usuário do AWS Site-to-Site VPN.
- Uma conexão do AWS Direct Connect. Para obter mais informações, consulte O que é o AWS Direct Connect? no Guia do usuário do AWS Direct Connect.

O acesso ao DynamoDB via rede é feito por meio de APIs publicadas pela AWS. Os clientes devem ser compatíveis com o Transport Layer Security (TLS) 1.2. Recomendamos o TLS 1.3. Os clientes também devem ter suporte a pacotes de criptografia com sigilo de encaminhamento perfeito (PFS)

como Ephemeral Diffie-Hellman (DHE) ou Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). A maioria dos sistemas modernos como Java 7 e versões posteriores oferece suporte a esses modos. Além disso, você deve assinar solicitações usando um ID da chave de acesso e uma chave de acesso secreta associados a uma entidade principal do IAM. Ou você pode usar o AWS Security Token Service (STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Tráfego entre recursos da AWS na mesma região

Um endpoint da Amazon Virtual Private Cloud (Amazon VPC) para DynamoDB é uma entidade lógica dentro de uma VPC que permite conectividade apenas com o DynamoDB. A Amazon VPC encaminha as solicitações para o DynamoDB e roteia as respostas de volta para a VPC. Para obter mais informações, consulte Endpoints da VPC no Guia do usuário da Amazon VPC. Para obter exemplos de políticas que podem ser usadas para controlar o acesso a partir de endpoints da VPC, consulte Usar políticas do IAM para controlar o acesso ao DynamoDB.

> **Note**
>
> Os endpoints da Amazon VPC não podem ser acessados via AWS Site-to-Site VPN ou AWS Direct Connect.

AWS Identity and Access Management (IAM) e DynamoDB

O AWS Identity and Access Management é um serviço da AWS que ajuda a controlar o acesso aos recursos da AWS de maneira segura. Os administradores controlam quem pode ser autenticado (fazer login) e autorizado (ter permissões) para usar os recursos do Amazon DynamoDB e do DynamoDB Accelerator. Você pode usar o IAM para gerenciar as permissões de acesso e implementar políticas de segurança para o Amazon DynamoDB e o DynamoDB Accelerator. O IAM é um AWS serviço da que pode ser usado sem custo adicional.

Tópicos

- Gerenciamento de identidade e acesso no Amazon DynamoDB
- Uso de condições de política do IAM para controle de acesso refinado

Gerenciamento de identidade e acesso no Amazon DynamoDB

O AWS Identity and Access Management (IAM) é um serviço da AWS service (Serviço da AWS) que ajuda um administrador a controlar com segurança o acesso aos recursos da AWS. Os administradores do IAM controlam quem pode ser autenticado (fazer login) e autorizado (ter permissões) para usar os recursos do DynamoDB. O IAM é um AWS service (Serviço da AWS) que pode ser usado sem custo adicional.

Tópicos

- Público
- Autenticação com identidades
- Gerenciar o acesso usando políticas
- Como o Amazon DynamoDB funciona com o IAM
- Exemplos de políticas baseadas em identidade para o Amazon DynamoDB
- Solução de problemas de identidade e acesso no Amazon DynamoDB
- Política do IAM para evitar a compra de capacidade reservada do DynamoDB

Público

O uso do AWS Identity and Access Management (IAM) varia dependendo do trabalho que for realizado no DynamoDB.

Usuário do serviço: se você usar o serviço DynamoDB para realizar seu trabalho, o administrador fornecerá as credenciais e as permissões necessárias. À medida que usar mais recursos do DynamoDB para realizar seu trabalho, você poderá precisar de permissões adicionais. Entender como o acesso é gerenciado pode ajudar a solicitar as permissões corretas ao administrador. Se não for possível acessar um recurso no DynamoDB, consulte Solução de problemas de identidade e acesso no Amazon DynamoDB.

Administrador do serviço: se você for o responsável pelos recursos do DynamoDB na empresa, provavelmente terá acesso total ao DynamoDB. Cabe a você determinar quais funcionalidades e recursos do DynamoDB os usuários do serviço devem acessar. Envie as solicitações ao administrador do IAM para alterar as permissões dos usuários de serviço. Revise as informações nesta página para compreender os conceitos básicos do IAM. Para saber mais sobre como a empresa pode usar o IAM com o DynamoDB, consulte Como o Amazon DynamoDB funciona com o IAM.

Administrador do IAM: se você for um administrador do IAM, é recomendável conhecer os detalhes sobre como criar políticas para gerenciar o acesso ao DynamoDB. Para visualizar exemplos de

políticas baseadas em identidade do DynamoDB que podem ser usadas no IAM, consulte Exemplos de políticas baseadas em identidade para o Amazon DynamoDB.

Autenticação com identidades

A autenticação é a forma como fazer login na AWS usando suas credenciais de identidade. É necessário ser autenticado (fazer login na AWS) como Usuário raiz da conta da AWS, como usuário do IAM, ou assumindo um perfil do IAM.

É possível fazer login na AWS como uma identidade federada usando credenciais fornecidas por uma fonte de identidades. AWS IAM Identity Center (Centro de Identidade do IAM), a autenticação única da empresa e as suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Quando você acessa a AWS usando a federação, está indiretamente assumindo um perfil.

A depender do tipo de usuário, você pode fazer login no AWS Management Console ou no portal de acesso AWS. Para obter mais informações sobre como fazer login na AWS, consulte Como fazer login na Conta da AWS no Guia do usuário do Início de Sessão da AWS.

Se você acessar a AWS de forma programática, a AWS fornecerá um kit de desenvolvimento de software (SDK) e uma interface de linha de comandos (CLI) para você assinar de forma criptográfica as solicitações usando as suas credenciais. Se você não utilizar as ferramentas da AWS, deverá assinar as solicitações por conta própria. Para obter mais informações sobre como usar o método recomendado para designar solicitações por conta própria, consulte Versão 4 do AWS Signature para solicitações de API no Guia do usuário do IAM.

Independente do método de autenticação usado, também pode ser necessário fornecer informações adicionais de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte Autenticação multifator no Guia do usuário do AWS IAM Identity Center e Usar a autenticação multifator da AWS no IAM no Guia do usuário do IAM.

Usuário-raiz Conta da AWS

Ao criar uma Conta da AWS, você começa com uma identidade de login com acesso completo a todos os Serviços da AWS e recursos na conta. Essa identidade, chamada usuário-raiz da Conta da AWS, é acessada por login com o endereço de e-mail e a senha usada para criar a conta. É altamente recomendável não usar o usuário-raiz para tarefas diárias. Proteja as credenciais do

usuário-raiz e use-as para executar as tarefas que somente ele puder executar. Para obter a lista completa das tarefas que exigem login como usuário-raiz, consulte Tarefas que exigem credenciais de usuário-raiz no Guia do Usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários, inclusive os que precisam de acesso de administrador, usem a federação com um provedor de identidades para acessar os Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário de seu diretório de usuários corporativos, um provedor de identidades da web, o AWS Directory Service, o diretório do Centro de Identidade ou qualquer usuário que acesse os Serviços da AWS usando credenciais fornecidas por meio de uma fonte de identidade. Quando as identidades federadas acessam Contas da AWS, elas assumem perfis que fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, é recomendável usar o AWS IAM Identity Center. É possível criar usuários e grupos no IAM Identity Center ou conectar-se e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todas as suas Contas da AWS e aplicações. Para obter mais informações sobre o Centro de Identidade do IAM, consulte O que é o Centro de Identidade do IAM? no Guia do Usuário do AWS IAM Identity Center.

Usuários e grupos do IAM

Um usuário do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas para uma única pessoa ou aplicação. Sempre que possível, é recomendável contar com credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com usuários do IAM, é recomendável alternar as chaves de acesso. Para obter mais informações, consulte Alternar as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do Usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdmins e conceder a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários

têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Casos de uso para usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Um perfil do IAM é uma identidade na Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. Para assumir temporariamente um perfil do IAM no AWS Management Console, você pode alternar de um usuário para um perfil do IAM (console). É possível presumir um perfil chamando uma operação de API da AWS CLI ou da AWS, ou usando um URL personalizado. Para obter mais informações sobre métodos para usar perfis, consulte Métodos para assumir um perfil no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, é possível criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas por ele. Para ter mais informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidade de terceiros (federação) no Guia do usuário do IAM. Se usar o Centro de Identidade do IAM, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de Identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de Permissões no Guia do Usuário do AWS IAM Identity Center.
- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode presumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, alguns Serviços da AWS permitem anexar uma política diretamente a um recurso (em vez de usar um perfil como proxy). Para conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.
- Acesso entre serviços: alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões da entidade principal da chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.

- Sessões de acesso direto (FAS): qualquer pessoa que utilizar um perfil ou usuário do IAM para executar ações na AWS é considerada uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS usa as permissões da entidade principal chamando um AWS service (Serviço da AWS), bem como o AWS service (Serviço da AWS) solicitante, para fazer solicitações para serviços subsequentes. As solicitações de FAS são feitas somente quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Sessões de acesso direto.
- Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.
- Perfil vinculado a serviço: um perfil vinculado a serviço é um tipo de perfil de serviço vinculado a um AWS service (Serviço da AWS). O serviço pode presumir o perfil para executar uma ação em seu nome. Perfis vinculadas ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados a serviço.

- Aplicações em execução no Amazon EC2: é possível usar um perfil do IAM para gerenciar credenciais temporárias para aplicações em execução em uma instância do EC2 e fazer solicitações da AWS CLI ou da API da AWS. É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir um perfil da AWS a uma instância do EC2 e disponibilizá-la para todas as suas aplicações, crie um perfil de instância que esteja anexado a ela. Um perfil de instância contém o perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte Utilizar um perfil do IAM para conceder permissões a aplicações em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso na AWS criando políticas e anexando-as a identidades ou recursos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou recurso, define suas permissões. A AWS avalia essas políticas quando uma entidade principal (usuário, usuário-raiz ou sessão de perfil) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas é armazenada na AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e perfis não têm permissões. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação independentemente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de perfis do AWS Management Console, da AWS CLI ou da API AWS.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e perfis na Conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. As entidades principais podem incluir contas, usuários, funções, usuários federados ou Serviços da AWS.

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Não é possível usar as políticas gerenciadas pela AWS do IAM em uma política baseada em atributos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou perfis da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

Amazon S3, AWS WAF e Amazon VPC são exemplos de serviços que oferecem compatibilidade com ACLs. Para saber mais sobre ACLs, consulte Visão geral da lista de controle de acesso (ACL) no Guia do Desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

A AWS oferece compatibilidade com tipos de política menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões: um limite de permissões é um recurso avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou o perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs): SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (UO) no AWS Organizations. O AWS Organizations é um serviço que agrupa e gerencia centralmente várias Contas da AWS pertencentes a sua empresa. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. A SCP limita as permissões para entidades em contas de membros, o que inclui cada Usuário raiz da conta da AWS. Para obter mais informações sobre o Organizations e SCPs, consulte Service control policies no Guia do usuário do AWS Organizations.
- Políticas de controle de recursos (RCPs): RCPs são políticas JSON que podem ser usadas para definir o máximo de permissões disponíveis para recursos em suas contas sem atualizar as políticas do IAM anexadas a cada recurso que você possui. A RCP limita as permissões para

recursos nas contas-membro e pode afetar as permissões efetivas para identidades, incluindo o Usuário raiz da conta da AWS, independentemente de pertencerem a sua organização. Consulte mais informações sobre o Organizations e as RCPs, incluindo uma lista de Serviços da AWS compatível com RCPs em Resource control policies (RCPs) no Guia do usuário do AWS Organizations.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recursos. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como a AWS determina se deve permitir ou não uma solicitação quando há vários tipos de política envolvidos, consulte Lógica da avaliação de políticas no Guia do usuário do IAM.

Como o Amazon DynamoDB funciona com o IAM

Antes de usar o IAM para gerenciar o acesso ao DynamoDB, saiba quais recursos do IAM estão disponíveis para uso com o DynamoDB.

Recurso do IAM	Suporte ao DynamoDB
Políticas baseadas em identidade	Sim
Políticas baseadas em atributos	Sim
Ações de políticas	Sim
Recursos de políticas	Sim
Chaves de condição de políticas	Sim

Recurso do IAM	Suporte ao DynamoDB
ACLs	Não
ABAC (tags em políticas)	Sim
Credenciais temporárias	Sim
Permissões de entidade principal	Sim
Perfis de serviço	Sim
Perfis vinculados a serviço	Sim

Para obter uma visão geral de como o Amazon DynamoDB e outros serviços da AWS funcionam com a maioria dos recursos do IAM, consulte Serviços da AWS que funcionam com o IAM no Guia do usuário do IAM.

Políticas baseadas em identidade para o DynamoDB

Compatível com políticas baseadas em identidade: sim

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário do IAM, grupo de usuários ou perfil. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

Com as políticas baseadas em identidade do IAM, é possível especificar ações e recursos permitidos ou negados, assim como as condições sob as quais as ações são permitidas ou negadas. Você não pode especificar a entidade principal em uma política baseada em identidade porque ela se aplica ao usuário ou perfil ao qual ela está anexada. Para saber mais sobre todos os elementos que podem ser usados em uma política JSON, consulte Referência de elemento de política JSON do IAM no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade para o DynamoDB

Para visualizar exemplos de políticas baseadas em identidade do DynamoDB, consulte Exemplos de políticas baseadas em identidade para o Amazon DynamoDB.

Políticas baseadas em recursos no DynamoDB

Compatível com políticas baseadas em recursos: sim

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. As entidades principais podem incluir contas, usuários, perfis, usuários federados ou Serviços da AWS.

Para permitir o acesso entre contas, você pode especificar uma conta inteira ou as entidades do IAM em outra conta como a entidade principal em uma política baseada em recursos. Adicionar uma entidade principal entre contas à política baseada em recurso é apenas metade da tarefa de estabelecimento da relação de confiança. Quando a entidade principal e o recurso estão em diferentes Contas da AWS, um administrador do IAM da conta confiável também deve conceder à entidade principal (usuário ou perfil) permissão para acessar o recurso. Eles concedem permissão ao anexar uma política baseada em identidade para a entidade. No entanto, se uma política baseada em recurso conceder acesso a uma entidade principal na mesma conta, nenhuma política baseada em identidade adicional será necessária. Para obter mais informações, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Ações de políticas para o DynamoDB

Compatível com ações de políticas: sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. As ações de políticas geralmente têm o mesmo nome que a operação de API da AWS associada. Existem algumas exceções, como ações somente de permissão, que não têm uma operação de API correspondente. Algumas operações também exigem várias ações em uma política. Essas ações adicionais são chamadas de ações dependentes.

Incluem ações em uma política para conceder permissões para executar a operação associada.

Para ver uma lista das ações do DynamoDB, consulte Ações definidas pelo Amazon DynamoDB na Referência de autorização do serviço.

As ações de políticas no DynamoDB usam o seguinte prefixo antes da ação:

```
aws
```

Para especificar várias ações em uma única declaração, separe-as com vírgulas.

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Para visualizar exemplos de políticas baseadas em identidade do DynamoDB, consulte Exemplos de políticas baseadas em identidade para o Amazon DynamoDB.

Recursos de políticas para o DynamoDB

Compatível com recursos de políticas: sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou `NotResource`. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon (ARN). Isso pode ser feito para ações que oferecem compatibilidade com um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem compatibilidade com permissões em nível de recurso, como operações de listagem, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

Para ver uma lista dos recursos do DynamoDB e os respectivos ARNs, consulte Tipos de recursos definidos pelo Amazon DynamoDB na Referência de autorização do serviço. Para saber com quais ações você pode especificar o ARN de cada recurso, consulte Ações definidas pelo Amazon DynamoDB.

Para visualizar exemplos de políticas baseadas em identidade do DynamoDB, consulte Exemplos de políticas baseadas em identidade para o Amazon DynamoDB.

Chaves de condição de políticas para o DynamoDB

Compatível com chaves de condição de política específicas de serviço: sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` (ou bloco `Condition`) permite que você especifique condições nas quais uma instrução estiver em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos de `Condition` em uma declaração ou várias chaves em um único elemento de `Condition`, a AWS os avaliará usando uma operação lógica `AND`. Se você especificar vários valores para uma única chave de condição, a AWS avaliará a condição usando uma operação lógica `OR`. Todas as condições devem ser atendidas antes que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar condições. Por exemplo, é possível conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte Elementos da política do IAM: variáveis e tags no Guia do usuário do IAM.

A AWS oferece compatibilidade com chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as chaves de condição globais da AWS, consulte Chaves de contexto de condição globais da AWS no Guia do usuário do IAM.

Para ver uma lista de chaves de condição do DynamoDB, consulte Chaves de condição do Amazon DynamoDB na Referência de autorização do serviço. Para saber com quais ações e recursos você pode usar a chave de condição, consulte Ações definidas pelo Amazon DynamoDB.

Para visualizar exemplos de políticas baseadas em identidade do DynamoDB, consulte Exemplos de políticas baseadas em identidade para o Amazon DynamoDB.

Listas de controle de acesso (ACLs) no DynamoDB

Compatível com ACLs: não

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou perfis da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

Controle de acesso por atributo (ABAC) com o DynamoDB

Compatível com ABAC (tags em políticas): sim

O controle de acesso por atributo (ABAC) é uma estratégia de autorização que define as permissões com base em atributos. Na AWS, esses atributos são chamados de tags. É possível anexar tags a entidades do IAM (usuários ou perfis) e a muitos recursos da AWS. Marcar de entidades e atributos é a primeira etapa do ABAC. Em seguida, você cria políticas de ABAC para permitir operações quando a tag da entidade principal corresponder à tag do recurso que ela estiver tentando acessar.

O ABAC é útil em ambientes que estão crescendo rapidamente e ajuda em situações em que o gerenciamento de políticas se torna um problema.

Para controlar o acesso baseado em tags, forneça informações sobre as tags no elemento de condição de uma política usando as `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* ou chaves de condição `aws:TagKeys`.

Se um serviço for compatível com as três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço for compatível com as três chaves de condição somente para alguns tipos de recursos, o valor será Parcial

Para obter mais informações sobre o ABAC, consulte Definir permissões com autorização do ABAC no Guia do usuário do IAM. Para visualizar um tutorial com etapas para configurar o ABAC, consulte Usar controle de acesso baseado em atributos (ABAC) no Guia do usuário do IAM.

Usar credenciais temporárias com o DynamoDB

Compatível com credenciais temporárias: sim

Alguns Serviços da AWS não funcionam quando você faz login usando credenciais temporárias. Para obter informações adicionais, como quais Serviços da AWS funcionam com credenciais temporárias, consulte Serviços da AWS que funcionem com o IAM no Guia do usuário do IAM.

Você está usando credenciais temporárias se fizer login no AWS Management Console por qualquer método, exceto nome de usuário e uma senha. Por exemplo, quando você acessa a AWS usando o link de autenticação única (SSO) da sua empresa, esse processo cria credenciais temporárias automaticamente. Você também cria automaticamente credenciais temporárias quando faz login

no console como usuário e, em seguida, alterna perfis. Para obter mais informações sobre como alternar funções, consulte Alternar para um perfil do IAM (console) no Guia do usuário do IAM.

É possível criar credenciais temporárias manualmente usando a API AWS CLI ou AWS. Em seguida, você pode usar essas credenciais temporárias para acessar a AWS. A AWS recomenda que você gere credenciais temporárias dinamicamente em vez de usar chaves de acesso de longo prazo. Para obter mais informações, consulte Credenciais de segurança temporárias no IAM.

Permissões de entidade principal entre serviços para o DynamoDB

Compatibilidade com o recurso de encaminhamento de sessões de acesso (FAS): sim

O usuário ou perfil do IAM usado para executar ações na AWS é considerado uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS usa as permissões da entidade principal chamando um AWS service (Serviço da AWS), bem como o AWS service (Serviço da AWS) solicitante, para fazer solicitações para serviços subsequentes. As solicitações de FAS são feitas somente quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Sessões de acesso direto.

Perfis de serviço para o DynamoDB

Compatível com perfis de serviço: sim

O perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.

> **Warning**
>
> Alterar as permissões de um perfil de serviço pode prejudicar a funcionalidade do DynamoDB. Edite perfis de serviço somente quando o DynamoDB fornecer orientação para isso.

Perfis vinculados ao serviço para o DynamoDB

Compatibilidade com perfis vinculados a serviços: sim

Um perfil vinculado a serviço é um tipo de perfil de serviço vinculado a um AWS service (Serviço da AWS). O serviço pode presumir o perfil de executar uma ação em seu nome. Perfis vinculadas ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para funções vinculadas ao serviço.

Para obter detalhes sobre como criar ou gerenciar perfis vinculados a serviços, consulte Serviços da AWS que funcionam com o IAM. Encontre um serviço na tabela que inclua um `Yes` na coluna Perfil vinculado ao serviço. Escolha o link Sim para visualizar a documentação do perfil vinculado a serviço desse serviço.

Perfis vinculados ao serviço aceitos no DynamoDB

Os perfis vinculados ao serviço a seguir são aceitos no DynamoDB.

- O DynamoDB usa o perfil vinculado ao serviço AWSServiceRoleForDynamoDBReplication para replicação de tabelas globais nas Regiões da AWS. Consulte Segurança de tabelas globais do DynamoDB para ter mais informações sobre o perfil vinculado ao serviço AWSServiceRoleForDynamoDBReplication.
- O DynamoDB Accelerator (DAX) usa o perfil vinculado ao serviço AWSServiceRoleForDAX para configurar e manter um cluster do DAX. Consulte the section called “Usar funções vinculadas ao serviço para o DAX” para ter mais informações sobre o perfil vinculado ao serviço AWSServiceRoleForDAX.

Além desses perfis vinculados ao serviço do DynamoDB, a solução usa o serviço Application Auto Scaling para gerenciar automaticamente as configurações de throughput nas tabelas do modo de capacidade provisionada. O serviço Application Auto Scaling usa o perfil vinculado ao serviço AWSServiceRoleForApplicationAutoScaling_DynamoDBTable para gerenciar as configurações de throughput em tabelas do DynamoDB que têm o ajuste de escala automático habilitado. Para ter mais informações, consulte Service-linked roles for Application Auto Scaling.

Exemplos de políticas baseadas em identidade para o Amazon DynamoDB

Por padrão, usuários e perfis não têm permissão para criar ou modificar recursos do DynamoDB. Eles também não podem executar tarefas usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API da AWS. Para conceder aos usuários permissões para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

Para aprender a criar uma política baseada em identidade do IAM ao usar esses documentos de política em JSON de exemplo, consulte Criar políticas do IAM (console) no Guia do usuário do IAM.

Para obter detalhes sobre ações e tipos de recurso definidos pelo DynamoDB, incluindo o formato dos ARNs para cada um dos tipos de recurso, consulte Ações, recursos e chaves de condição do Amazon DynamoDB na Referência de autorização do serviço.

Tópicos

- Práticas recomendadas de política
- Como usar o console do DynamoDB
- Permitir que os usuários visualizem suas próprias permissões
- Usar políticas baseadas em identidade com o Amazon DynamoDB

Práticas recomendadas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos do DynamoDB em sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas gerenciadas pela AWS e avance para as permissões de privilégio mínimo: para começar a conceder permissões a seus usuários e workloads, use as políticas gerenciadas pela AWS, que concedem permissões para muitos casos de uso comuns. Elas estão disponíveis em seus Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente da AWS que são específicas para seus casos de uso. Para obter mais informações, consulte Políticas gerenciadas pela AWS ou Políticas gerenciadas pela AWS para funções de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como usar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.
- Use condições nas políticas do IAM para restringir ainda mais o acesso: você pode adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, você pode escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Também pode usar condições para conceder acesso a ações de serviço, se elas forem usadas por meio de um AWS service (Serviço da AWS) específico, como o AWS CloudFormation.

Para obter mais informações, consulte Elementos da política JSON do IAM: condição no Guia do usuário do IAM.

- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.
- Exigir autenticação multifator (MFA): se houver um cenário que exija usuários do IAM ou um usuário raiz em sua Conta da AWS, ative a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para obter mais informações, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Como usar o console do DynamoDB

Para acessar o console da Amazon DynamoDB, você deve ter um conjunto mínimo de permissões. Essas permissões devem consentir que você liste e visualize detalhes sobre os recursos do DynamoDB em sua Conta da AWS. Caso crie uma política baseada em identidade mais restritiva que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou perfis) com essa política.

Não é necessário conceder permissões mínimas do console para usuários que fazem chamadas somente à AWS CLI ou à API do AWS. Em vez disso, permita o acesso somente a ações que correspondam à operação de API que estiverem tentando executar.

Para garantir que usuários e perfis ainda possam usar o console do DynamoDB, anexe também a política `ConsoleAccess` ou `ReadOnly` do DynamoDB gerenciada pela AWS às entidades. Para obter informações, consulte Adicionar permissões a um usuário no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou de forma programática usando a AWS CLI ou a API da AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Usar políticas baseadas em identidade com o Amazon DynamoDB

Este tópico aborda o uso de políticas baseadas em identidade do AWS Identity and Access Management (IAM) com o Amazon DynamoDB e fornece exemplos. Os exemplos mostram como um administrador de conta pode anexar políticas de permissões a identidades do IAM (usuários, grupos e funções) e, dessa forma, conceder permissões para executar operações em recursos do Amazon DynamoDB.

As seções neste tópico abrangem o seguinte:

- Permissões do IAM necessárias para usar o console do Amazon DynamoDB
- Políticas do IAM (predefinidas) gerenciadas pela AWS para o Amazon DynamoDB
- Exemplos de política gerenciada pelo cliente

Veja a seguir um exemplo de política de permissões.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        }
    ]
}
```

A política anterior tem uma instrução que concede permissões para três ações do DynamoDB (`dynamodb:DescribeTable`, `dynamodb:Query` e `dynamodb:Scan`) em uma tabela na região da AWS `us-west-2` que pertence à conta da AWS especificada por *`account-id`*. O Nome do recurso da Amazon (ARN) no valor `Resource` especifica a tabela à qual as permissões se aplicam.

Permissões do IAM necessárias para usar o console do Amazon DynamoDB

Para trabalhar com o console do DynamoDB, os usuários precisam ter um conjunto mínimo de permissões para utilizar os recursos do DynamoDB de suas contas da AWS. Além dessas permissões do DynamoDB, o console exige permissões:

- Permissões do Amazon CloudWatch para exibir métricas e gráficos.
- Permissões do AWS Data Pipeline para exportar e importar dados do DynamoDB.

- Permissões do AWS Identity and Access Management para acessar funções necessárias para exportações e importações.
- Permissões do Amazon Simple Notification Service para enviar notificações sempre que um alarme do CloudWatch é acionado.
- Permissões do AWS Lambda para processar registros do DynamoDB Streams.

Se você criar uma política do IAM que seja mais restritiva que as permissões mínimas necessárias, o console do não funcionará como pretendido para os usuários com essa política do IAM. Para garantir que esses usuários ainda consigam usar o console do DynamoDB, associe também a política gerenciada pela AWS `AmazonDynamoDBReadOnlyAccess` ao usuário, conforme descrito em Políticas do IAM (predefinidas) gerenciadas pela AWS para o Amazon DynamoDB.

Não é necessário conceder permissões mínimas do console para os usuários que estão fazendo chamadas somente com a AWS CLI ou com a API do Amazon DynamoDB.

> **Note**
>
> Se você fizer referência a um endpoint da VPC, também precisará autorizar a chamada à API DescribeEndpoints para a(s) entidades principal(is) do IAM solicitantes com a ação do IAM (dynamodb:DescribeEndpoints). Para obter mais informações, consulte Política necessária para endpoints.

Políticas do IAM (predefinidas) gerenciadas pela AWS para o Amazon DynamoDB

A AWS aborda alguns casos de uso comuns fornecendo políticas autônomas do IAM que são criadas e administradas pela AWS. Essas políticas gerenciadas pela AWS concedem as permissões necessárias para casos de uso comuns. Assim, você não precisa investigar quais permissões são necessárias. Para obter mais informações, consulte Políticas gerenciadas da AWS no Guia do usuário do IAM.

As seguintes políticas gerenciadas pela AWS, que você pode anexar aos usuários da sua conta, são específicas do DynamoDB e agrupadas por cenário de caso de uso:

- AmazonDynamoDBReadOnlyAccess: concede acesso somente leitura aos recursos do DynamoDB via AWS Management Console.
- AmazonDynamoDBFullAccess: concede acesso total aos recursos do DynamoDB via AWS Management Console.

É possível analisar essas políticas de permissões gerenciadas pela AWS fazendo login no console do IAM e pesquisando políticas específicas.

> **Important**
>
> A prática recomendada é criar políticas personalizadas do IAM que concedam privilégio mínimo aos usuários, perfis ou grupos que precisam dele.

Exemplos de política gerenciada pelo cliente

Nesta seção, você encontrará exemplos de políticas de usuário que concedem permissões para diversas ações do DynamoDB. Essas políticas funcionam quando você está usando os AWS SDKs ou a AWS CLI. Quando você usa o console, precisa conceder permissões adicionais que são específicas do console. Para obter mais informações, consulte Permissões do IAM necessárias para usar o console do Amazon DynamoDB.

> **Note**
>
> Todos os exemplos de política a seguir usam uma das regiões da AWS e contêm IDs de conta e nomes de tabelas fictícios.

Exemplos:

- Política do IAM para conceder permissões a todas as ações do DynamoDB em uma tabela
- Política do IAM para conceder permissões somente leitura em itens de uma tabela do DynamoDB
- Política do IAM para conceder acesso a uma tabela específica do DynamoDB e seus índices
- Política do IAM para ler, gravar, atualizar e excluir o acesso em uma tabela do DynamoDB
- Política do IAM para separar ambientes do DynamoDB na mesma conta da AWS
- Política do IAM para evitar a compra de capacidade reservada do DynamoDB
- Política do IAM para conceder acesso de leitura somente para um fluxo do DynamoDB (não para a tabela)
- Política do IAM para permitir que uma função AWS Lambda acesse registros de fluxo do DynamoDB
- Política do IAM para acesso de leitura e gravação a um cluster do DynamoDB Accelerator (DAX)

O Guia do usuário do IAM inclui três exemplos adicionais do DynamoDB:

- Amazon DynamoDB: permite acesso a uma tabela específica
- Amazon DynamoDB: permite acesso a colunas específicas
- Amazon DynamoDB: permite acesso por linha ao DynamoDB com base em um ID do Amazon Cognito

Política do IAM para conceder permissões a todas as ações do DynamoDB em uma tabela

A política de permissões a seguir concede permissões para todas as ações do DynamoDB em uma tabela chamada `Books`. O ARN do recurso especificado em `Resource` identifica uma tabela em uma região da AWS específica. Se você substituir o nome da tabela `Books` no ARN do `Resource` por um caractere curinga (*), permitirá todas as ações do DynamoDB em todas as tabelas da conta. Considere cuidadosamente as possíveis implicações de segurança antes de usar um caractere curinga nesta ou em qualquer política do IAM.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> **Note**
>
> Este é um exemplo de uso de um caractere curinga (*) para permitir todas as ações, incluindo administração, operações de dados, monitoramento e aquisição de capacidade reservada do DynamoDB. Em vez disso, é uma prática recomendada especificar explicitamente cada ação a ser concedida e apenas o que esse usuário, função ou grupo precisa.

Política do IAM para conceder permissões somente leitura em itens de uma tabela do DynamoDB

A política de permissões a seguir concede permissões somente para as ações `GetItem`, `BatchGetItem`, `Scan`, `Query` e `ConditionCheckItem` do DynamoDB e, portanto, define o acesso somente leitura à tabela `Books`.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "ReadOnlyAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Política do IAM para conceder acesso a uma tabela específica do DynamoDB e seus índices

A política a seguir concede permissões para ações de modificação de dados em uma tabela do DynamoDB chamada `Books` e todos os índices dessa tabela. Para obter mais informações sobre como os índices funcionam, consulte Melhorar o acesso aos dados com índices secundários no DynamoDB.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AccessTableAllIndexesOnBooks",
```

```
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

Política do IAM para ler, gravar, atualizar e excluir o acesso em uma tabela do DynamoDB

Use essa política se precisar permitir que a aplicação crie, leia, atualize e exclua dados em tabelas, índices e streams do Amazon DynamoDB. Substitua o nome da região da AWS, ID da conta e o nome da tabela ou caractere curinga (*), quando apropriado.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "DynamoDBIndexAndStreamAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:ListStreams"
```

```
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/stream/*"
            ]
        },
        {
            "Sid": "DynamoDBTableAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:DescribeTable",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        },
        {
            "Sid": "DynamoDBDescribeLimitsAccess",
            "Effect": "Allow",
            "Action": "dynamodb:DescribeLimits",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

Para expandir essa política a fim de abranger todas as tabelas do DynamoDB em todas as regiões da AWS para esta conta, use um curinga (*) para a região e o nome da tabela. Por exemplo:

```
"Resource":[
              "arn:aws:dynamodb:*:123456789012:table/*",
              "arn:aws:dynamodb:*:123456789012:table/*/index/*"
              ]
```

Política do IAM para separar ambientes do DynamoDB na mesma conta da AWS

Suponha que você tenha ambientes separados, onde cada ambiente mantém sua própria versão de uma tabela chamada `ProductCatalog`. Se você criar duas tabelas `ProductCatalog` na mesma conta da AWS, o trabalho realizado em um ambiente poderá afetar o outro ambiente devido à forma como as permissões são configuradas. Por exemplo, as cotas no número de operações simultâneas de ambiente de gerenciamento (como `CreateTable`) são definidas por cada conta da AWS.

Como resultado, cada ação em um ambiente reduz o número de operações disponíveis no outro ambiente. Há também um risco do código em seu ambiente de teste poder acessar acidentalmente tabelas no ambiente de produção.

> **Note**
>
> Se preferir separar as workloads de produção e teste para ajudar a controlar o "raio de explosão" de um evento, a prática recomendada é criar contas da AWS separadas para workloads de teste e produção. Para obter mais informações, consulte Gerenciamento e separação de contas da AWS.

Suponha também que você tenha dois desenvolvedores, Amit e Alice, que estão testando a tabela `ProductCatalog`. Em vez de cada desenvolvedor ter uma conta da AWS os desenvolvedores podem compartilhar a mesma conta de teste da AWS. Nesta conta de teste, você pode criar uma cópia da mesma tabela para cada desenvolvedor trabalhar, como `Alice_ProductCatalog` e `Amit_ProductCatalog`. Nesse caso, é possível criar os usuários Alice e Amit na conta da AWS que você criou para o ambiente de teste. Em seguida, você pode conceder permissões para esses usuários executarem ações do DynamoDB nas tabelas que eles possuem.

Para conceder essas permissões de usuário do IAM, você pode executar uma das seguintes ações:

- Crie uma política separada para cada usuário e, em seguida, anexe cada política ao seu usuário separadamente. Por exemplo, você pode anexar a política a seguir ao usuário Alice para permitir que ela acesse todas as ações do DynamoDB na tabela `Alice_ProductCatalog`:

  JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
```

```
            "Sid": "AllAPIActionsOnAliceTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:DeleteItem",
              "dynamodb:DescribeContributorInsights",
              "dynamodb:RestoreTableToPointInTime",
              "dynamodb:ListTagsOfResource",
              "dynamodb:CreateTableReplica",
              "dynamodb:UpdateContributorInsights",
              "dynamodb:CreateBackup",
              "dynamodb:DeleteTable",
              "dynamodb:UpdateTableReplicaAutoScaling",
              "dynamodb:UpdateContinuousBackups",
              "dynamodb:TagResource",
              "dynamodb:DescribeTable",
              "dynamodb:GetItem",
              "dynamodb:DescribeContinuousBackups",
              "dynamodb:BatchGetItem",
              "dynamodb:UpdateTimeToLive",
              "dynamodb:BatchWriteItem",
              "dynamodb:ConditionCheckItem",
              "dynamodb:UntagResource",
              "dynamodb:PutItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteTableReplica",
              "dynamodb:DescribeTimeToLive",
              "dynamodb:RestoreTableFromBackup",
              "dynamodb:UpdateTable",
              "dynamodb:DescribeTableReplicaAutoScaling",
              "dynamodb:GetShardIterator",
              "dynamodb:DescribeStream",
              "dynamodb:GetRecords",
              "dynamodb:DescribeLimits",
              "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
Alice_ProductCatalog/*"
        }
    ]
}
```

Em seguida, você pode criar uma política semelhante com um recurso diferente (a tabela `Amit_ProductCatalog`) para o usuário Amit.

- Em vez de anexar políticas a usuários individuais, você pode usar variáveis de política do IAM para gravar uma única política e anexá-la a um grupo. Você precisa criar um grupo e, no caso deste exemplo, adicionar ambos os usuários Alice e Amit ao grupo. O exemplo a seguir concede permissões para executar todas as ações do DynamoDB na tabela `${aws:username}_ProductCatalog`. A variável de política `${aws:username}` é substituída pelo nome de usuário do solicitante quando a política é avaliada. Por exemplo, se Alice envia uma solicitação para adicionar um item, a ação é permitida apenas se Alice estiver adicionando itens à tabela `Alice_ProductCatalog`.

  JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "ActionsOnUserSpecificTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
${aws:username}_ProductCatalog"
        },
        {
            "Sid": "AdditionalPrivileges",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContributorInsights"
            ],
```

```
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
        }
    ]
}
```

> **Note**
>
> Ao usar variáveis de política do IAM, você deve especificar explicitamente a versão `2012-10-17` da linguagem da política do IAM na política. A versão padrão da linguagem da política do IAM (`2008-10-17`) não é compatível com as variáveis da política.

Em vez de identificar uma tabela específica como um recurso, conforme normalmente faria, você pode usar um caractere curinga (*) para conceder permissões em todas as tabelas em que o nome da tabela tem como prefixo o usuário que está fazendo a solicitação, conforme mostrado a seguir.

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

Política do IAM para evitar a compra de capacidade reservada do DynamoDB

Com a capacidade reservada do Amazon DynamoDB, você paga uma taxa única antecipada e se compromete a pagar por um nível mínimo de utilização, com uma economia significativa, ao longo de um período. Você pode usar o AWS Management Console para visualizar e comprar capacidade reservada. No entanto, talvez você não queira que todos os usuários em sua organização possam adquirir capacidade reservada. Para obter mais informações sobre capacidade reservada, consulte Preços do Amazon DynamoDB.

O DynamoDB fornece as seguintes operações de API para controlar o acesso ao gerenciamento de capacidade reservada:

- `dynamodb:DescribeReservedCapacity`: retorna as compras de capacidade reservada que estão em vigor no momento.
- `dynamodb:DescribeReservedCapacityOfferings`: retorna detalhes sobre os planos de capacidade reservada que são oferecidos no momento pela AWS.
- `dynamodb:PurchaseReservedCapacityOfferings`: executa uma compra real de capacidade reservada.

O AWS Management Console usa essas ações da API para exibir as informações de capacidade reservada e para fazer compras. Você não pode chamar essas operações de um programa de aplicação, pois elas podem ser acessadas somente pelo console. No entanto, é possível permitir ou negar o acesso a essas operações em uma política de permissões do IAM.

A política a seguir permite que os usuários visualizem ofertas de capacidade reservada e ofertas atuais usando o AWS Management Console, mas novas compras são negadas.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

Observe que esta política usa o caractere curinga (*) para permitir permissões de descrição para todos, e para negar a compra de capacidade reservada do DynamoDB para todos.

Política do IAM para conceder acesso de leitura somente para um fluxo do DynamoDB (não para a tabela)

Quando você ativa o DynamoDB Streams em uma tabela, informações são capturadas sobre cada modificação em itens de dados na tabela. Para obter mais informações, consulte Capturar dados de alterações para o DynamoDB Streams.

Em alguns casos, talvez você queira evitar que uma aplicação leia dados de uma tabela do DynamoDB, mas ainda permitir o acesso aos fluxos dessa tabela. Por exemplo, você pode configurar o AWS Lambda para sondar um fluxo e invocar uma função do Lambda quando as atualizações de itens forem detectadas e, em seguida, executar processamento adicional.

As ações a seguir estão disponíveis para controlar o acesso a fluxos do DynamoDB:

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

O exemplo de política a seguir concede aos usuários permissões para acessar os streams em uma tabela chamada `GameScores`. O último caractere curinga (*) no ARN corresponde a qualquer ID de fluxo associado a essa tabela.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AccessGameScoresStreamOnly",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Observe que essa política concede acesso aos fluxos da tabela `GameScores`, mas não à tabela em si.

Política do IAM para permitir que uma função AWS Lambda acesse registros de fluxo do DynamoDB

Para que determinadas ações sejam executadas com base em novos eventos em um stream do DynamoDB, é possível escrever uma função do AWS Lambda que seja acionada por esses eventos. Uma função do Lambda como essa precisa de permissões para ler dados de um fluxo do DynamoDB. Para obter mais informações sobre como usar o Lambda com DynamoDB Streams, consulte DynamoDB Streams e acionadores do AWS Lambda.

Para conceder permissões ao Lambda, use a política de permissões que está associada ao perfil do IAM do Lambda (também conhecida como função de execução). Especifique essa política ao criar a função do Lambda.

Por exemplo, você pode associar a política de permissões a seguir a uma função de execução para conceder permissões ao Lambda para realizar as ações do DynamoDB Streams listadas.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "APIAccessForDynamoDBStreams",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Para obter mais informações, consulte Permissões do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

Política do IAM para acesso de leitura e gravação a um cluster do DynamoDB Accelerator (DAX)

A política a seguir permite o acesso para leitura, gravação, atualização e exclusão em um cluster do DynamoDB Accelerator (DAX), mas não na tabela do DynamoDB associada. Para usar essa política, substitua o nome da região da AWS, o ID da conta e o nome do cluster do DAX.

> **Note**
>
> Essa política dá acesso ao cluster do DAX, mas não à tabela associada do DynamoDB. Seu cluster do DAX deve dispor da política correta para executar por você essas mesmas operações na tabela do DynamoDB.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AmazonDynamoDBDAXDataOperations",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:PutItem",
                "dax:ConditionCheckItem",
                "dax:BatchGetItem",
                "dax:BatchWriteItem",
                "dax:DeleteItem",
                "dax:Query",
                "dax:UpdateItem",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:eu-west-1:123456789012:cache/MyDAXCluster"
        }
    ]
}
```

Para expandir essa política de modo a dar ao DAX acesso a todas as regiões da AWS de uma conta, use um caractere curinga (*) para o nome da região.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Solução de problemas de identidade e acesso no Amazon DynamoDB

Use as informações a seguir para ajudar a diagnosticar e corrigir problemas comuns que podem ser encontrados ao trabalhar com o DynamoDB e o IAM.

Tópicos

- Não tenho autorização para executar uma ação no DynamoDB
- Não estou autorizado a executar iam:PassRole
- Quero permitir que as pessoas fora da minha Conta da AWS acessem meus recursos do DynamoDB

Não tenho autorização para executar uma ação no DynamoDB

Se o AWS Management Console informar que você não foi autorizado a executar uma ação, você deve entrar em contato com o administrador para obter assistência. O administrador é a pessoa que forneceu o seu nome de usuário e senha.

O erro do exemplo a seguir ocorre quando o usuário `mateojackson` tenta usar o console para visualizar detalhes sobre um recurso do *`my-example-widget`* fictício, mas não tem as permissões fictícias do `aws:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

Neste caso, Mateo pede ao administrador para atualizar suas políticas para permitir a ele o acesso ao recurso *`my-example-widget`* usando a ação `aws:`*`GetWidget`*.

Não estou autorizado a executar iam:PassRole

Se você receber uma mensagem de erro informando que não tem autorização para executar a ação `iam:PassRole`, você precisará atualizar as políticas para poder passar um perfil para o DynamoDB.

Alguns Serviços da AWS permitem que você passe um perfil existente para o serviço, em vez de criar um perfil de serviço ou perfil vinculado ao serviço. Para fazer isso, é preciso ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta usar o console para executar uma ação no DynamoDB. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se você precisar de ajuda, entre em contato com seu administrador AWS. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que as pessoas fora da minha Conta da AWS acessem meus recursos do DynamoDB

Você pode criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem compatibilidade com políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se o DynamoDB oferece compatibilidade para esses recursos, consulte Como o Amazon DynamoDB funciona com o IAM.
- Para saber como conceder acesso a seus recursos em todas as Contas da AWS pertencentes a você, consulte Fornecimento de acesso a um usuário do IAM em outra Conta da AWS pertencente a você no Guia de usuário do IAM.
- Para saber como conceder acesso a seus recursos para Contas da AWS de terceiros, consulte Fornecimento de acesso a Contas da AWS pertencentes a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para saber a diferença entre o uso de perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Política do IAM para evitar a compra de capacidade reservada do DynamoDB

Com a capacidade reservada do Amazon DynamoDB, você paga uma taxa única antecipada e se compromete a pagar por um nível mínimo de utilização, com uma economia significativa, ao longo de um período. Você pode usar o AWS Management Console para visualizar e comprar capacidade reservada. No entanto, talvez você não queira que todos os usuários em sua organização possam adquirir capacidade reservada. Para obter mais informações sobre capacidade reservada, consulte Preços do Amazon DynamoDB.

O DynamoDB fornece as seguintes operações de API para controlar o acesso ao gerenciamento de capacidade reservada:

- `dynamodb:DescribeReservedCapacity`: retorna as compras de capacidade reservada que estão em vigor no momento.
- `dynamodb:DescribeReservedCapacityOfferings`: retorna detalhes sobre os planos de capacidade reservada que são oferecidos no momento pela AWS.
- `dynamodb:PurchaseReservedCapacityOfferings`: executa uma compra real de capacidade reservada.

O AWS Management Console usa essas ações da API para exibir as informações de capacidade reservada e para fazer compras. Você não pode chamar essas operações de um programa de aplicação, pois elas podem ser acessadas somente pelo console. No entanto, é possível permitir ou negar o acesso a essas operações em uma política de permissões do IAM.

A política a seguir permite que os usuários visualizem ofertas de capacidade reservada e ofertas atuais usando o AWS Management Console, mas novas compras são negadas.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
```

```
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

Observe que esta política usa o caractere curinga (*) para permitir permissões de descrição para todos, e para negar a compra de capacidade reservada do DynamoDB para todos.

Uso de condições de política do IAM para controle de acesso refinado

Ao conceder permissões no DynamoDB, você pode especificar as condições que determinam como uma política de permissões entra em vigor.

Visão geral

No DynamoDB, você tem a opção de especificar condições ao conceder permissões usando uma política do IAM (consulte Gerenciamento de identidade e acesso no Amazon DynamoDB). Por exemplo, é possível:

- Conceder permissões a fim de permitir acesso somente leitura aos usuários para determinados itens e atributos em uma tabela ou um índice secundário.
- Conceder permissões para permitir somente acesso de gravação aos usuários para determinados atributos em uma tabela com base na identidade desse usuário.

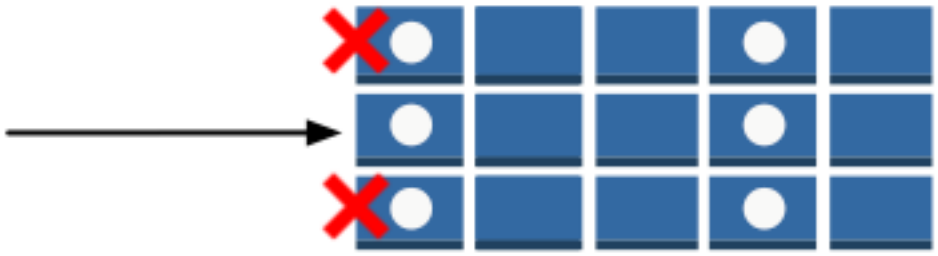
No DynamoDB, você pode especificar as condições em uma política do IAM usando chaves de condição, como ilustrado no caso de uso na seção a seguir.

Caso de uso de permissões

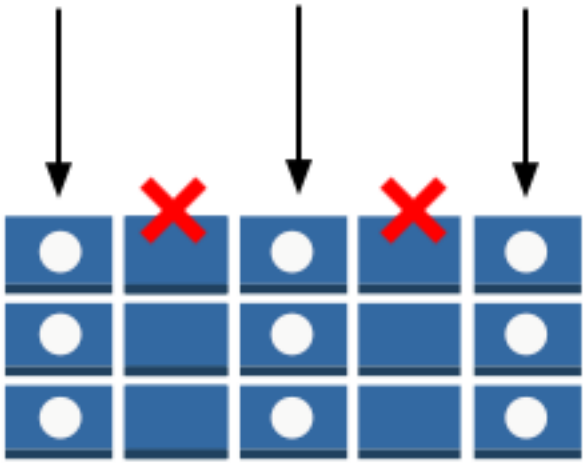
Além de controlar o acesso a ações de API do DynamoDB, você também pode controlar o acesso a itens de dados e atributos individuais. Por exemplo, você pode fazer o seguinte:

- Conceder permissões em uma tabela, mas restringir o acesso a itens específicos dessa tabela com base em determinados valores de chave primária. Um exemplo pode ser uma aplicação

de rede social para jogos, onde todos os dados de jogos salvos dos usuários são armazenados em uma única tabela, mas nenhum usuário pode acessar itens de dados que não possui, como mostrado na ilustração a seguir:



- Oculte informações para que apenas um subconjunto de atributos fique visível para o usuário. Um exemplo pode ser uma aplicação que exibe os dados de voo para aeroportos próximos, com base na localização do usuário. Nomes de companhias aéreas e horas de partida, além de números de voo, são exibidos. No entanto, atributos como nomes de piloto ou número de passageiros são ocultos, como mostrado na ilustração a seguir:



Para implementar esse tipo de controle de acesso refinado, grave uma política de permissões do IAM que especifica condições para acessar as credenciais de segurança e as permissões associadas. Em seguida, aplique a política aos usuários, grupos ou funções do que você criar usando o console do IAM. Sua política do IAM pode restringir o acesso a cada item de uma tabela, aos atributos desses itens ou a ambos ao mesmo tempo.

Se preferir, você pode usar federação de identidades na web para controlar o acesso dos usuários que serão autenticados por meio do Login with Amazon, Facebook ou Google. Para obter mais informações, consulte Usar federação de identidades na Web.

Use o elemento `Condition` do IAM para implementar uma política de controle de acesso refinada. Ao adicionar um elemento `Condition` a uma política de permissões, você pode permitir ou negar o acesso a itens e atributos em tabelas e índices do DynamoDB com base em seus requisitos comerciais específicos.

Como um exemplo, considere um aplicativo de jogos móveis que permite que os jogadores selecionem e joguem uma variedade de jogos diferentes. A aplicação usa uma tabela do DynamoDB chamada `GameScores` para manter o controle das pontuações máximas e outros dados do usuário. Cada item na tabela é exclusivamente identificado por um ID de usuário e o nome do jogo que o usuário jogou. A tabela `GameScores` tem uma chave primária que consiste em uma chave de partição (`UserId`) e a chave de classificação (`GameTitle`). Os usuários só têm acesso aos dados de jogos associados ao seu ID de usuário. Um usuário que deseja jogar um jogo deve pertencer a um perfil do IAM chamada `GameRole`, a qual tem uma política de segurança anexada a ela.

Para gerenciar permissões de usuário neste aplicativo, grave uma política de permissões, como a seguinte:

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
```

```
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

Além de conceder permissões para ações específicas do DynamoDB (elemento `Action`) na tabela `GameScores` (elemento `Resource`), o elemento `Condition`usa as chaves de condição a seguir específicas do DynamoDB que limitam as permissões, da seguinte forma:

- `dynamodb:LeadingKeys`: esta chave de condição permite que os usuários acessem apenas os itens nos quais o valor de chave de partição coincide com seu ID de usuário. Este ID, `${www.amazon.com:user_id}`, é uma variável de substituição. Para obter mais informações sobre variáveis de substituição, consulte Usar federação de identidades na Web.
- `dynamodb:Attributes`: esta chave de condição limita o acesso aos atributos especificados para que apenas as ações listadas na política de permissões possam retornar valores para esses atributos. Além disso, a cláusula `StringEqualsIfExists` garante que o aplicativo sempre deve fornecer uma lista de atributos específicos nos quais atuar e que o aplicativo não pode solicitar todos os atributos.

Quando uma política do IAM é avaliada, o resultado é sempre verdadeiro (o acesso é permitido) ou falso (o acesso é negado). Se qualquer parte do elemento `Condition` é falsa, toda a política é avaliada como falsa e o acesso é negado.

> **Important**
>
> Caso você use `dynamodb:Attributes`, deve especificar os nomes de todos os atributos de chave de índice e chave primária da tabela, além de quaisquer índices secundários que estejam listados na política. Caso contrário, o DynamoDB não poderá usar esses atributos de chave para executar a ação solicitada.

Os documentos de política do IAM podem conter apenas os seguintes caracteres Unicode: guia horizontal (U+0009), linefeeds (U+000A), retorno de carro (U+000D) e caracteres no intervalo U +0020 a U+00FF.

Especificar condições: usar chaves de condição

A AWS fornece um conjunto de chaves de condição predefinidas (chaves de condição no âmbito da AWS) para todos os serviços da AWS que oferecem suporte ao IAM para controle de acesso. Por exemplo, você pode usar a condição de chave `aws:SourceIp` para verificar o endereço IP do solicitante antes de permitir que uma ação seja executada. Para obter mais informações e uma lista das chaves no âmbito da AWS, consulte Chaves disponíveis para condições no Guia do usuário do IAM.

A tabela a seguir mostra as chaves de condição específicas do serviço DynamoDB que se aplicam ao DynamoDB.

Chave de condição do DynamoDB	Descrição
`dynamodb:LeadingKeys`	Representa o primeiro atributo de chave de uma tabela – em outras palavras, a chave de partição. O nome da chave `LeadingKeys` é no plural, mesmo se ela for usada com ações de um único item. Além disso, você deve usar o modificador `ForAllValues` ao utilizar `LeadingKeys` em uma condição.
`dynamodb:Select`	Representa o parâmetro `Select` de uma `Query` ou solicitação `Scan`. `Select` pode ter qualquer um dos seguintes valores: • `ALL_ATTRIBUTES` • `ALL_PROJECTED_ATTRIBUTES`

Chave de condição do DynamoDB	Descrição
	• SPECIFIC_ATTRIBUTES • COUNT
dynamodb:Attributes	Representa uma lista dos nomes de atributos em uma solicitação, ou os atributos que são retornados de uma solicitação. Os valores de Attributes são nomeados da mesma forma e têm o mesmo significado que os parâmetros de determinadas ações da API do DynamoDB, conforme mostrado a seguir: • AttributesToGet Usado por: BatchGetItem, GetItem, Query, Scan • AttributeUpdates Usado por: UpdateItem • Expected Usado por: DeleteItem, PutItem, UpdateItem • Item Usado por: PutItem • ScanFilter Usado por: Scan
dynamodb:ReturnValues	Representa o parâmetro ReturnValues de uma solicitação. ReturnValues pode ter qualquer um dos seguintes valores: • ALL_OLD • UPDATED_OLD • ALL_NEW • UPDATED_NEW • NONE

Chave de condição do DynamoDB	Descrição
`dynamodb:ReturnConsumedCapacity`	Representa o parâmetro `ReturnConsumedCapacity` de uma solicitação. `ReturnConsumedCapacity` pode ter um dos seguintes valores: • `TOTAL` • `NONE`

Limite de acesso de usuário

Muitas políticas de permissões do IAM permitem que os usuários acessem esses itens em uma tabela, na qual o valor de chave de partição coincide com o identificador do usuário. Por exemplo, o aplicativo de jogo anterior limita o acesso dessa forma, para que os usuários possam acessar somente os dados do jogo que estão associados ao ID de usuário deles. As variáveis de substituição do IAM `${www.amazon.com:user_id}`, `${graph.facebook.com:id}` e `${accounts.google.com:sub}` contêm identificadores do usuário para o Login with Amazon, no Facebook e no Google. Para saber como um aplicativo se conecta a um desses provedores de identidade e obtém esses identificadores, consulte Usar federação de identidades na Web.

Important

Não é possível usar o controle de acesso refinado para restringir a replicação de tabelas globais. A aplicação de condições de política para controle de acesso refinado a entidades principais do serviço ou perfis vinculados ao serviço do DynamoDB usados para replicação de tabelas globais pode interromper a replicação em uma tabela global.

Note

Cada um dos exemplos na seção a seguir define a cláusula `Effect` como `Allow` e especifica apenas as ações, os recursos e os parâmetros permitidos. O acesso é permitido apenas para o que está listado explicitamente na política do IAM.
Em alguns casos, é possível reescrever essas políticas para que elas sejam baseadas em negação (ou seja, definindo a cláusula `Effect` como `Deny` e invertendo toda a lógica na política). No entanto, recomendamos que você evite usar políticas baseadas em negação

com o DynamoDB porque é difícil escrevê-las corretamente em comparação às políticas baseadas em permissão. Além disso, futuras alterações na API do DynamoDB (ou alterações nas entradas existentes da API) podem tornar ineficaz uma política baseada em negação.

Políticas de exemplo: usar condições para controle de acesso refinado

Esta seção mostra várias políticas para a implementação de um controle de acesso refinado em tabelas e índices do DynamoDB.

> **Note**
>
> Todos os exemplos usam a Região us-west-2 e contêm IDs de conta fictícios.

O vídeo a seguir explica o controle de acesso refinado no DynamoDB usando condições de política do IAM.

1: conceder permissões que limitam o acesso a itens com um valor específico de chave de partição

A política de permissões a seguir concede permissões que permitem um conjunto de ações do DynamoDB na tabela `GamesScore`. Ela usa a chave de condição `dynamodb:LeadingKeys` para limitar as ações do usuário apenas aos itens cujo valor da chave de partição `UserID` coincida com o ID do usuário único do Login with Amazon desse aplicativo.

> **Important**
>
> A lista de ações não inclui permissões para `Scan` porque `Scan` retorna todos os itens, independentemente das principais chaves.

JSON

```
{
   "Version":"2012-10-17"      ,
   "Statement":[
      {
         "Sid":"FullAccessToUserItems",
         "Effect":"Allow",
         "Action":[
```

```
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> Ao usar variáveis de política, você deve especificar explicitamente a versão `2012-10-17` na política. A versão padrão da linguagem de políticas de acesso, `2008-10-17`, não é compatível com as variáveis de política.

Para implementar o acesso somente leitura, você pode remover as ações que podem modificar os dados. Na política a seguir, somente as ações que fornecem acesso somente leitura são incluídas na condição.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Sid":"ReadOnlyAccessToUserItems",
```

```
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Important**
>
> Caso você use `dynamodb:Attributes`, deve especificar os nomes de todos os atributos de chave de índice e chave primária da tabela, além de quaisquer índices secundários que estejam listados na política. Caso contrário, o DynamoDB não poderá usar esses atributos de chave para executar a ação solicitada.

2: conceder permissões que limitam o acesso a determinados atributos em uma tabela

A política de permissões a seguir permite o acesso apenas a dois atributos em uma tabela, adicionando a chave de condição `dynamodb:Attributes`. Esses atributos podem ser lidos, gravados ou avaliados em um filtro de verificação ou gravação condicional.

JSON

```
{
   "Version":"2012-10-17"      ,
   "Statement":[
      {
```

```
         "Sid":"LimitAccessToSpecificAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "UserId",
                  "TopScore"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> A política usa uma abordagem de lista de permissões, o que permite o acesso a um conjunto nomeado de atributos. Mas você pode escrever uma política equivalente que nega o acesso a outros atributos. Não recomendamos essa abordagem com lista de proibições. Os usuários podem determinar os nomes desses atributos negados seguindo o princípio do privilégio mínimo, conforme explicado na Wikipedia em http://en.wikipedia.org/wiki/

Principle_of_least_privilege e usar uma abordagem de lista de permissões para enumerar todos os valores permitidos, em vez de especificar os atributos negados.

Essa política não permite `PutItem`, `DeleteItem` ou `BatchWriteItem`. Essas ações sempre substituem o item anterior inteiro, o que permitira aos usuários excluir os valores anteriores que eles não tem permissão para acessar.

A cláusula `StringEqualsIfExists` na política de permissões garante o seguinte:

- Se o usuário especifica o parâmetro `Select`, seu valor deve ser `SPECIFIC_ATTRIBUTES`. Essa exigência impede que a ação da API retorne quaisquer atributos que não sejam permitidos, tal como de uma projeção do índice.
- Se o usuário especifica o parâmetro `ReturnValues`, então seu valor deve ser `NONE`, `UPDATED_OLD` ou `UPDATED_NEW`. Isso é necessário porque a ação `UpdateItem` também executa operações de leitura implícitas para verificar se um item existe antes de substituí-lo, e de forma que os valores de atributo anteriores possam ser retornados, se solicitado. Restringir `ReturnValues` dessa forma garante que os usuários possam apenas ler ou gravar atributos permitidos.
- A cláusula `StringEqualsIfExists` garante que apenas um desses parâmetros – `Select` ou `ReturnValues` – pode ser usado por solicitação, no contexto das ações permitidas.

Veja a seguir algumas variações desta política:

- Para permitir apenas as ações de leitura, você pode remover `UpdateItem` da lista de ações permitidas. Como nenhuma das ações restantes aceitam `ReturnValues`, você pode remover `ReturnValues` da condição. Também é possível alterar `StringEqualsIfExists` para `StringEquals`, pois o parâmetro `Select` sempre tem um valor (`ALL_ATTRIBUTES`, a menos que especificado de outra forma).
- Para permitir apenas as ações de gravação, você pode remover tudo menos `UpdateItem` da lista de ações permitidas. Como `UpdateItem` não usa o parâmetro `Select`, você pode remover `Select` da condição. Você também tem que alterar `StringEqualsIfExists` para `StringEquals` porque o parâmetro `ReturnValues` sempre tem um valor (`NONE`, a menos que especificado de outra forma).
- Para permitir todos os atributos cujo nome corresponde a um padrão, use `StringLike` em vez de `StringEquals`, e use um caractere curinga de correspondência de padrão multicaractere (*).

3: conceder permissões para evitar atualizações em determinadas atributos

A política de permissões a seguir limita o acesso do usuário à atualização apenas dos atributos identificados pela chave de condição `dynamodb:Attributes`. A condição `StringNotLike` impede que um aplicativo atualize os atributos especificados usando a condição de chave `dynamodb:Attributes`.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Sid":"PreventUpdatesOnCertainAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
         "Condition":{
            "ForAllValues:StringNotLike":{
               "dynamodb:Attributes":[
                  "FreeGamesAvailable",
                  "BossLevelUnlocked"
               ]
            },
            "StringEquals":{
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

Observe o seguinte:

- A ação `UpdateItem`, como outras ações de gravação, exige acesso de leitura aos itens para que possa retornar valores antes e depois da atualização. Na política, é possível limitar a ação para acessar somente os atributos que podem ser atualizados, especificando a chave de condição `dynamodb:ReturnValues`. A chave de condição restringe `ReturnValues` na solicitação para especificar apenas `NONE`, `UPDATED_OLD` ou `UPDATED_NEW` e não inclui `ALL_OLD` ou `ALL_NEW`.
- As ações `PutItem` e `DeleteItem` substituem um item inteiro e, assim, permite que os aplicativos modifiquem quaisquer atributos. Portanto, ao limitar um aplicativo para atualizar somente atributos específicos, você não deve conceder permissão para essas APIs.

4: conceder permissões para consultar somente atributos projetados em um índice

A`TopScoreDateTimeIndex` política de permissões a seguir permite consultas em um índice secundário () usando a chave de condição `dynamodb:Attributes`. A política também limita as consultas para solicitar somente atributos específicos que foram projetados no índice.

Para solicitar que a aplicação especifique uma lista de atributos na consulta, a política também especifica a chave de condição `dynamodb:Select` para exigir que o parâmetro `Select` da ação `Query` do DynamoDB seja `SPECIFIC_ATTRIBUTES`. A lista de atributos é limitada a uma lista específica que é fornecida por meio da chave de condição `dynamodb:Attributes`.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Sid":"QueryOnlyProjectedIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "TopScoreDateTime",
                  "GameTitle",
```

```
                "Wins",
                "Losses",
                "Attempts"
            ]
         },
         "StringEquals":{
            "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
         }
      }
   }
]
}
```

A política de permissões a seguir é semelhante, mas a consulta deve solicitar todos os atributos que foram projetados no índice.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Sid":"QueryAllIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:Select":"ALL_PROJECTED_ATTRIBUTES"
            }
         }
      }
   ]
}
```

5: conceder permissões para limitar o acesso a determinados atributos e valores de chave de partição

A política de permissões a seguir permite ações específicas do DynamoDB (especificadas no elemento `Action`) em uma tabela e em um índice de tabela (especificadas no elemento `Resource`). A política usa a chave de condição `dynamodb:LeadingKeys` para restringir as permissões apenas aos itens cujo valor de chave de partição corresponde ao ID do usuário no Facebook.

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Sid":"LimitAccessToCertainAttributesAndKeyValues",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${graph.facebook.com:id}"
               ],
               "dynamodb:Attributes":[
                  "attribute-A",
                  "attribute-B"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
```

```
                "UPDATED_NEW"
            ]
        }
    }
  ]
}
```

Observe o seguinte:

- As ações de gravação permitidas pela política (`UpdateItem`) só podem modificar `attribute-A` ou `attribute-B`.
- Como a política permite `UpdateItem`, um aplicativo pode inserir novos itens, e os atributos ocultos serão nulos em novos itens. Se esses atributos estiverem projetados em `TopScoreDateTimeIndex`, a política terá o benefício adicional de impedir consultas que causem buscas na tabela.
- Os aplicativos não podem ler quaisquer atributos que não estejam listados em `dynamodb:Attributes`. Com essa política em vigor, uma aplicação deve definir o parâmetro `Select` como `SPECIFIC_ATTRIBUTES` em solicitações de leitura, e apenas os atributos da lista de permissões podem ser solicitados. Para solicitações de gravação, o aplicativo não pode definir `ReturnValues` como `ALL_OLD` ou `ALL_NEW` e ele não pode executar operações de gravação condicional com base em outros atributos.

Tópicos relacionados

- Gerenciamento de identidade e acesso no Amazon DynamoDB
- Permissões da API do DynamoDB: referência de ações, recursos e condições

Usar federação de identidades na Web

Se você estiver criando um aplicativo direcionado a um grande número de usuários, você poderá, opcionalmente, usar a federação de identidades da web para autenticação e autorização. A federação de identidades da web exclui a necessidade de criar usuários do individuais. Em vez disso, os usuários podem fazer login em um provedor de identidades e, então, obter credenciais de segurança temporárias do AWS Security Token Service (AWS STS). A aplicação pode então usar essas credenciais para acessar serviços da AWS.

A federação de identidades na web é compatível com os seguintes provedores de identidade:

- Login with Amazon
- Facebook
- Google

Recursos adicionais para federação de identidades na Web

Os recursos a seguir podem ajudá-lo a saber mais sobre a federação de identidades na web:

- A postagem Web Identity Federation using the AWS SDK para .NET no blog de desenvolvedores da AWS ensina a usar a federação de identidades na Web com o Facebook. Isso inclui snippets de código em C# que mostram como assumir um perfil do IAM com identidade da Web e usar credenciais de segurança temporárias para acessar um recurso da AWS.
- O AWS Mobile SDK for iOS e o AWS Mobile SDK for Androidcontêm exemplos de aplicações. Eles incluem código que mostra como chamar os provedores de identidade e, depois, como usar as informações desses provedores para obter e usar credenciais de segurança temporárias.
- O artigo Web Identity Federation with Mobile Applications (Federação de identidades na Web com aplicativos móveis) discute a federação de identidades na web e mostra um exemplo de como usar a federação de identidades na web para acessar um recurso da AWS.

Exemplo de política para federação de identidades na Web

Para mostrar como você pode usar a federação de identidades na Web com o DynamoDB, revise a tabela GameScores que foi apresentada em Uso de condições de política do IAM para controle de acesso refinado. Esta é a chave primária para GameScores:

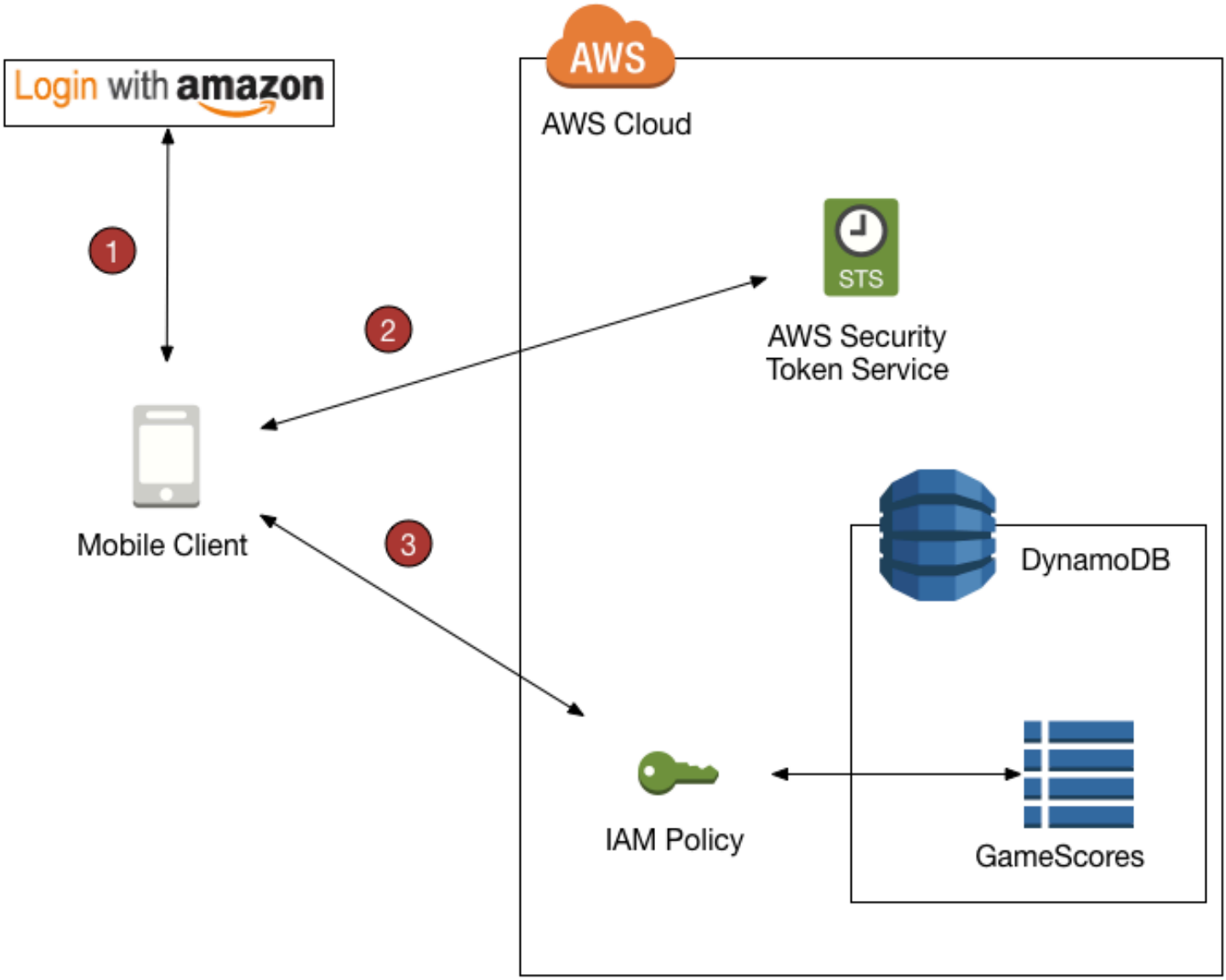
Nome da tabela	Tipo de chave primária	Nome e tipo de chave de partição	Nome e tipo de chave de classificação
GameScores (UserId, GameTitle, ...)	Composto	Nome do atributo: UserId Tipo: String	Nome do atributo: GameTitle Tipo: String

Agora, suponha que um aplicativo de jogos móveis use essa tabela, e que o aplicativo precise oferecer suporte a milhares, ou até mesmo milhões, de usuários. Nesta escala, torna-se muito difícil gerenciar usuários e aplicativos individuais e garantir que cada usuário possa acessar somente seus próprios dados na tabela GameScores. Felizmente, muitos usuários já têm contas com um provedor de identidade de terceiros, como o Facebook, o Google ou o Login with Amazon. Então faz sentido usar um desses provedores nas tarefas de autenticação.

Para fazer isso usando a federação de identidades na web, o desenvolvedor do aplicativo deve registrar o aplicativo com um provedor de identidade (como o Login with Amazon) e obter um ID de aplicativo exclusivo. Em seguida, o desenvolvedor precisa criar um perfil do IAM. (Por exemplo, essa perfil chama GameRole.) O perfil deve ter um documento de política do IAM anexado a ele especificando as condições sob as quais o aplicativo pode acessar a tabela GameScores.

Quando um usuário deseja jogar, ele faz login em sua conta do Login with Amazon de dentro do aplicativo do jogo. O aplicativo então chama o AWS Security Token Service (AWS STS) fornecendo o ID do aplicativo Login with Amazon e solicitando a associação em GameRole. O AWS STS retorna as credenciais temporárias da AWS para o aplicativo e permite que ele acesse a tabela GameScores de acordo com o documento de política de GameRole.

O diagrama a seguir mostra como essas partes se encaixam.



Visão geral da federação de identidades da .Web

1. O aplicativo chama um provedor de identidade de terceiros para autenticar o usuário e o aplicativo. O provedor de identidade retorna um token de identidade da web para o aplicativo.
2. O aplicativo chama AWS STS e passa o token de identidade da web como entrada. O AWS STS autoriza o aplicativo e fornece credenciais de acesso da AWS temporárias. O aplicativo tem permissão para assumir um perfil do IAM (GameRole) e acessar os recursos da AWS de acordo com a política de segurança do perfil.

3. O aplicativo chama o DynamoDB para acessar a tabela GameScores. Como ele assumiu a GameRole, o aplicativo está sujeito à política de segurança associada a esse perfil. O documento de política impede o aplicativo de acessar dados que não pertencem ao usuário.

Mais uma vez, essa é a política de segurança de GameRole que foi mostrada em Uso de condições de política do IAM para controle de acesso refinado:

JSON

```
{
   "Version":"2012-10-17"       ,
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
```

```
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

A cláusula `Condition` determina quais itens em GameScores são visíveis para o aplicativo. Isso é feito ao comparar o ID do Login with Amazon com os valores de chave de partição `UserId` em `GameScores`. Somente os itens que pertencem ao usuário atual podem ser processados usando uma das ações do DynamoDB que estão listadas nessa política. Outros itens na tabela não podem ser acessados. Além disso, somente os atributos específicos listados na política podem ser acessados.

Preparar para usar a federação de identidades na Web

Se você é um desenvolvedor de aplicativo e deseja usar a federação de identidades na web para seu aplicativo, siga estas etapas:

1. Cadastre-se como desenvolvedor em um provedor de identidades de terceiros. Os links externos a seguir fornecem informações sobre como se cadastrar em provedores de identidades compatíveis:
   - Centro do desenvolvedor do Login with Amazon
   - Registro no site do Facebook
   - Uso do OAuth 2.0 para acessar APIs do Google no site do Google
2. Registre seu aplicativo com o provedor de identidades. Ao fazer isso, o provedor fornece um ID exclusivo para o seu aplicativo. Se você deseja que o seu aplicativo funcione com vários provedores de identidade, é necessário obter um ID do aplicativo de cada fornecedor.
3. Crie uma ou mais perfis do IAM. Você precisa de uma função para cada provedor de identidade de cada aplicativo. Por exemplo, você pode criar um perfil que possa ser assumido por um aplicativo em que o usuário se conectou usando Login with Amazon, um segundo perfil para o mesmo aplicativo em que o usuário se conectou usando o Facebook, e um terceiro perfil para o aplicativo em que os usuários fazem login usando o Google.

Como parte do processo de criação do perfil, você precisa anexar uma política do IAM ao perfil. Seu documento de política deve definir os recursos do DynamoDB exigidos por sua aplicação, e as permissões para acessar esses recursos.

Para obter mais informações, consulte Sobre a federação de identidades na Web no Guia do usuário do IAM.

> **Note**
>
> Como alternativa ao AWS Security Token Service, você pode usar o Amazon Cognito. O Amazon Cognito é o serviço preferencial para o gerenciamento de credenciais temporárias para aplicações móveis. Para obter mais informações, consulte Obter credenciais no Guia do desenvolvedor do Amazon Cognito.

Gerar uma política do IAM usando o console do DynamoDB

O console do DynamoDB pode ajudar a criar uma política do IAM para usar com a federação de identidades na Web. Para fazer isso, você escolhe uma tabela do DynamoDB e especifica o provedor de identidade, ações e atributos a serem incluídos na política. O console do DynamoDB gera uma política que você pode anexar a um perfil do IAM.

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, selecione Tables (Tabelas).
3. Na lista de tabelas, escolha a tabela para a qual você deseja criar a política do IAM.
4. Selecione o botão Ações e escolha Criar política de controle de acesso.
5. Escolha o provedor de identidade, as ações e os atributos para a política.

   Quando as configurações estiverem como você deseja, selecione Gerar política. A política gerada aparece.
6. Selecione Consulte a documentação e siga as etapas necessárias para anexar a política gerada a um perfil do IAM.

Criar aplicação para usar a federação de identidades na Web

Para usar a federação de identidades na Web, sua aplicação precisa assumir o perfil do IAM que você criou. A partir daí, o aplicativo honra a política de acesso que você anexou ao perfil.

No tempo de execução, se o seu aplicativo usa a federação de identidades na web, ele deverá seguir estas etapas:

1. Autentique-se com um provedor de identidade de terceiros. Seu aplicativo deve chamar o provedor de identidade usando uma interface fornecida por ele. A maneira exata em que você autentica o usuário depende do provedor e em que plataforma seu aplicativo está em execução. Normalmente, se o usuário ainda não estiver conectado, o provedor de identidade exibe uma página de login para esse provedor.

   Depois de autenticar o usuário, o provedor de identidade retorna um token de identidade da web para seu aplicativo. O formato desse token depende do fornecedor, mas geralmente é uma string muito longa.
2. Obtenha credenciais de segurança temporárias da AWS. Para fazer isso, sua aplicação envia uma solicitação `AssumeRoleWithWebIdentity` para o AWS Security Token Service (AWS STS). Essa solicitação contém o seguinte:
   - O token de identidade da web da etapa anterior
   - O ID do aplicativo do provedor de identidade
   - O nome do recurso da Amazon (ARN) do perfil do IAM que você criou junto a este provedor de identidade para esta aplicação

   O AWS STS retorna um conjunto de credenciais de segurança da AWS que expira depois de um determinado período de tempo (por padrão, 3.600 segundos).

   Veja a seguir uma solicitação e resposta de exemplo de uma ação `AssumeRoleWithWebIdentity` no AWS STS. O token de identidade da web foi obtido do provedor de identidade Login with Amazon.

```
GET / HTTP/1.1
Host: sts.amazonaws.com
Content-Type: application/json; charset=utf-8
URL: https://sts.amazonaws.com/?ProviderId=www.amazon.com
&DurationSeconds=900&Action=AssumeRoleWithWebIdentity
&Version=2011-06-15&RoleSessionName=web-identity-federation
&RoleArn=arn:aws:iam::123456789012:role/GameRole
```

```
&WebIdentityToken=Atza|IQEBLjAsAhQluyKqyBiYZ8-kclvGTYM81e...(remaining characters
 omitted)
```

```
<AssumeRoleWithWebIdentityResponse
  xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
  <AssumeRoleWithWebIdentityResult>
    <SubjectFromWebIdentityToken>amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE</
SubjectFromWebIdentityToken>
    <Credentials>
      <SessionToken>AQoDYXdzEMf//////////wEa8AP6nNDwcSLnf+cHupC...(remaining
 characters omitted)</SessionToken>
      <SecretAccessKey>8Jhi60+EWUUbbUShTEsjTxqQtM8UKvsM6XAjdA==</SecretAccessKey>
      <Expiration>2013-10-01T22:14:35Z</Expiration>
      <AccessKeyId>06198791C436IEXAMPLE</AccessKeyId>
    </Credentials>
    <AssumedRoleUser>
      <Arn>arn:aws:sts::123456789012:assumed-role/GameRole/web-identity-federation</
Arn>
      <AssumedRoleId>AROAJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
    </AssumedRoleUser>
  </AssumeRoleWithWebIdentityResult>
  <ResponseMetadata>
    <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
  </ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>
```

3. Acesse os recursos da AWS. A resposta do AWS STS contém informações de que seu aplicativo precisa para acessar recursos do DynamoDB:
   - Os campos `AccessKeyID`, `SecretAccessKey` e `SessionToken` contêm credenciais de segurança válidas apenas para este usuário e este aplicativo.
   - O campo `Expiration` significa o limite de tempo dessas credenciais. Depois disso, elas não são mais válidas.
   - O campo `AssumedRoleId` contém o nome de um perfil do IAM específico da sessão que foi assumida pela aplicação. A aplicação honra os controles de acesso do documento de política do IAM durante esta sessão.
   - O campo `SubjectFromWebIdentityToken` contém o ID exclusivo que aparece em uma variável de política do IAM para esse provedor de identidade específico. Veja a seguir as variáveis de política do IAM para provedores compatíveis, e alguns valores de exemplo para elas:

Variável de política	Valor de exemplo
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`12345678901234567890 1`

Por obter políticas do IAM de exemplo nas quais essas variáveis de política são usadas, consulte Políticas de exemplo: usar condições para controle de acesso refinado.

Para obter mais informações sobre como o AWS STS gera credenciais de acesso temporárias, consulte Solicitação de credenciais de segurança temporárias no Guia do usuário do IAM.

Permissões da API do DynamoDB: referência de ações, recursos e condições

Ao configurar o Gerenciamento de identidade e acesso no Amazon DynamoDB e criar uma política de permissões que pode ser anexada a uma identidade do IAM (políticas baseadas em identidade), você poderá usar a lista de Ações, recursos e chaves de condição para Amazon DynamoDB no Guia do usuário do IAM como referência. A página a seguir lista cada operação de API do DynamoDB, as ações correspondentes às quais você pode conceder permissões para executar a ação e o recurso da AWS ao qual você pode conceder as permissões. Você especifica as ações no campo `Action` da política e o valor do recurso no campo `Resource` da política.

Você pode usar as chaves de condição usadas por toda a AWS em suas políticas do DynamoDB para expressar condições. Para obter uma lista completa de chaves válidas no âmbito da AWS, consulte a Referência de elementos de políticas JSON do IAM no Guia do usuário do IAM.

Além das chaves de condição no âmbito da AWS, o DynamoDB tem suas próximas chaves específicas que você pode usar nas condições. Para obter mais informações, consulte Uso de condições de política do IAM para controle de acesso refinado.

Tópicos relacionados

- Gerenciamento de identidade e acesso no Amazon DynamoDB
- Uso de condições de política do IAM para controle de acesso refinado

Validação de conformidade do DynamoDB por setor

Para saber se um AWS service (Serviço da AWS) está no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo por programa de conformidade e selecione o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de Conformidade da AWS.

É possível baixar relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte Baixar relatórios no AWS Artifact.

Sua responsabilidade de conformidade ao usar o Serviços da AWS é determinada pela confidencialidade dos seus dados, pelos objetivos de conformidade da sua empresa e pelos regulamentos e leis aplicáveis. A AWS fornece os recursos a seguir para ajudar com a conformidade:

- Governança e conformidade de segurança: esses guias de implementação de solução abordam considerações sobre a arquitetura e fornecem etapas para implantar recursos de segurança e conformidade.
- Referência de serviços qualificados para HIPAA: lista os serviços qualificados para HIPAA. Nem todos os Serviços da AWS estão qualificados pela HIPAA.
- Recursos de Conformidade da AWS: essa coleção de manuais e guias pode ser aplicada ao seu setor e local.
- Guias de conformidade do cliente da AWS: entenda o modelo de responsabilidade compartilhada sob a ótica da conformidade. Os guias resumem as práticas recomendadas para proteção de Serviços da AWS e mapeiam as diretrizes para controles de segurança em várias estruturas (incluindo o Instituto Nacional de Padrões e Tecnologia (NIST), o Conselho de Padrões de Segurança do Setor de Cartões de Pagamento (PCI) e a Organização Internacional de Padronização (ISO)).
- Avaliar recursos com regras no Guia do desenvolvedor do AWS Config: o serviço AWS Config avalia como as configurações de recursos estão em conformidade com práticas internas, diretrizes do setor e regulamentos.
- AWS Security Hub: este AWS service (Serviço da AWS) fornece uma visão abrangente do seu estado de segurança na AWS. O Security Hub usa controles de segurança para avaliar os recursos da AWS e verificar a conformidade com os padrões e as práticas recomendadas do setor de segurança. Para obter uma lista dos serviços e controles aceitos, consulte a Referência de controles do Security Hub.
- Amazon GuardDuty: este AWS service (Serviço da AWS) detecta possíveis ameaças às suas Contas da AWS, workloads, contêineres e dados ao monitorar o ambiente em busca de atividades

suspeitas e maliciosas. O GuardDuty pode ajudar você a atender a diversos requisitos de conformidade, como o PCI DSS, com o cumprimento dos requisitos de detecção de intrusões requeridos por determinadas estruturas de conformidade.

- AWS Audit Manager: este AWS service (Serviço da AWS) ajuda você a auditar continuamente o seu uso da AWS para simplificar o modo como você gerencia os riscos e a conformidade com regulamentos e padrões do setor.

Resiliência e recuperação de desastres no Amazon DynamoDB

A infraestrutura global da AWS é criada com base em regiões da AWS e zonas de disponibilidade. As regiões da AWS As regiões fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, as quais são conectadas com baixa latência, alto throughput e redes altamente redundantes. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que executam o failover automaticamente entre as zonas de disponibilidade sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de datacenter tradicionais.

Se você precisa replicar seus dados ou aplicações para distâncias geográficas maiores, use as regiões locais da AWS. Uma região local da AWS é um único data center projetado para complementar um região da AWS existente. Como todas as regiões da AWS, as regiões locais da AWS são completamente isoladas de outras regiões da AWS.

Para obter mais informações sobre AWS Regiões e Zonas de Disponibilidade, consulte AWS global infrastructure.

Além da infraestrutura global da AWS, o Amazon DynamoDB conta com vários recursos para oferecer suporte às suas necessidades de resiliência e backup de dados.

Backup e restauração sob demanda

O DynamoDB oferece o recurso de backup sob demanda. Ele permite que você crie backups completos das suas tabelas para retenção e arquivamento de longo prazo. Para obter mais informações, consulte Backup e restauração sob demanda para o DynamoDB.

Recuperação pontual

A recuperação em um ponto anterior no tempo ajuda a proteger as tabelas do DynamoDB contra operações acidentais de gravação ou exclusão. Com a recuperação point-in-time, você não

precisa se preocupar com a criação, a manutenção ou a programação de backups sob demanda. Para obter mais informações, consulte Recuperação pontual para o DynamoDB.

Tabelas globais sincronizadas entre regiões da AWS

O DynamoDB distribui automaticamente os dados e o tráfego para as tabelas por um número suficiente de servidores para lidar com seus requisitos de throughput e armazenamento sem deixar de manter uma performance consistente e rápida. Todos os dados são armazenados em discos de estado sólido (SSDs) e automaticamente replicados entre várias zonas de disponibilidade em uma região da AWS, o que oferece alta durabilidade de dados e disponibilidade integradas. Você pode usar tabelas globais para manter as tabelas do DynamoDB sincronizadas em regiões da AWS.

Segurança da infraestrutura no Amazon DynamoDB

Como um serviço gerenciado, o Amazon DynamoDB é protegido pelos procedimentos de segurança de redes globais da AWS que são descritos na proteção de segurança localizada no AWS Well-Architected Framework.

Você usa as chamadas de API publicadas da AWS para acessar o DynamoDB via rede. Os clientes podem usar TLS (Transport Layer Security) versão 1.2 ou 1.3. Os clientes também devem ter suporte a pacotes de criptografia com sigilo de encaminhamento perfeito (PFS) como Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Diffie-Hellman Encaminhamento (ECDHE). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos. Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Você também pode usar um endpoint da VPC (nuvem privada virtual) para DynamoDB para permitir que instâncias do Amazon EC2 em sua VPC usem seus endereços IP privados para acessar o DynamoDB sem se expor à Internet pública. Para obter mais informações, consulte Usar endpoints da Amazon VPC para acessar o DynamoDB.

Usar endpoints da Amazon VPC para acessar o DynamoDB

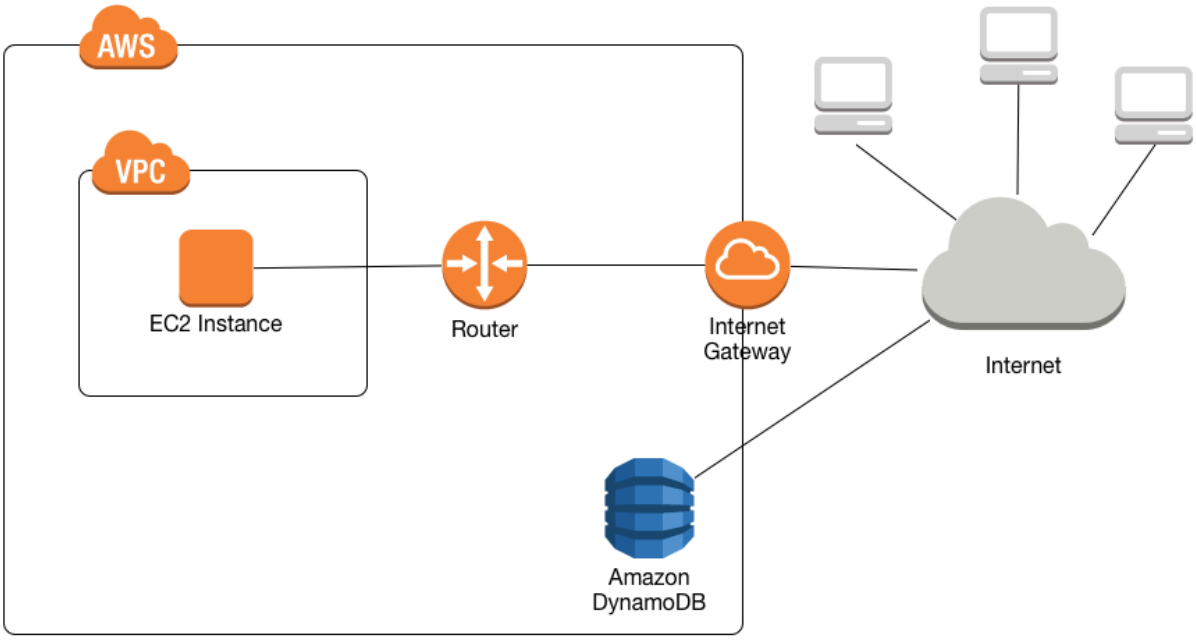
Por motivos de segurança, muitos clientes da AWS executam suas aplicações em um ambiente da Amazon Virtual Private Cloud (Amazon VPC). Com a Amazon VPC, você pode executar instâncias do Amazon EC2 em uma nuvem privada virtual, a qual é isolada logicamente de outras redes, incluindo a Internet pública. Com uma Amazon VPC, você tem controle sobre o intervalo de

endereços IP, as sub-redes, as tabelas de roteamento, os gateways de rede e as configurações de segurança.

> **Note**
>
> Se você criou a Conta da AWS após 4 de dezembro de 2013, você já tem uma VPC padrão em cada Região da AWS. Uma VPC padrão é fornecida pronta para ser usada. Você pode começar a usar sua VPC padrão imediatamente, sem precisar executar quaisquer etapas de configuração adicionais.
> Para obter mais informações, consulte VPC padrão e sub-redes padrão no Guia do usuário da Amazon VPC.

Para acessar a Internet pública, a VPC deverá ter um gateway da Internet, um roteador virtual que conecta a VPC à Internet. Isso permite que as aplicações em execução no Amazon EC2 em sua VPC acessem recursos da Internet, como o Amazon DynamoDB.

Por padrão, as comunicações de e para o DynamoDB usam o protocolo HTTPS, o qual protege o tráfego de rede usando a criptografia SSL/TLS. O diagrama a seguir mostra uma instância do Amazon EC2 em uma VPC acessando o DynamoDB ao fazer com que o DynamoDB use um gateway da internet em vez de endpoints da VPC.



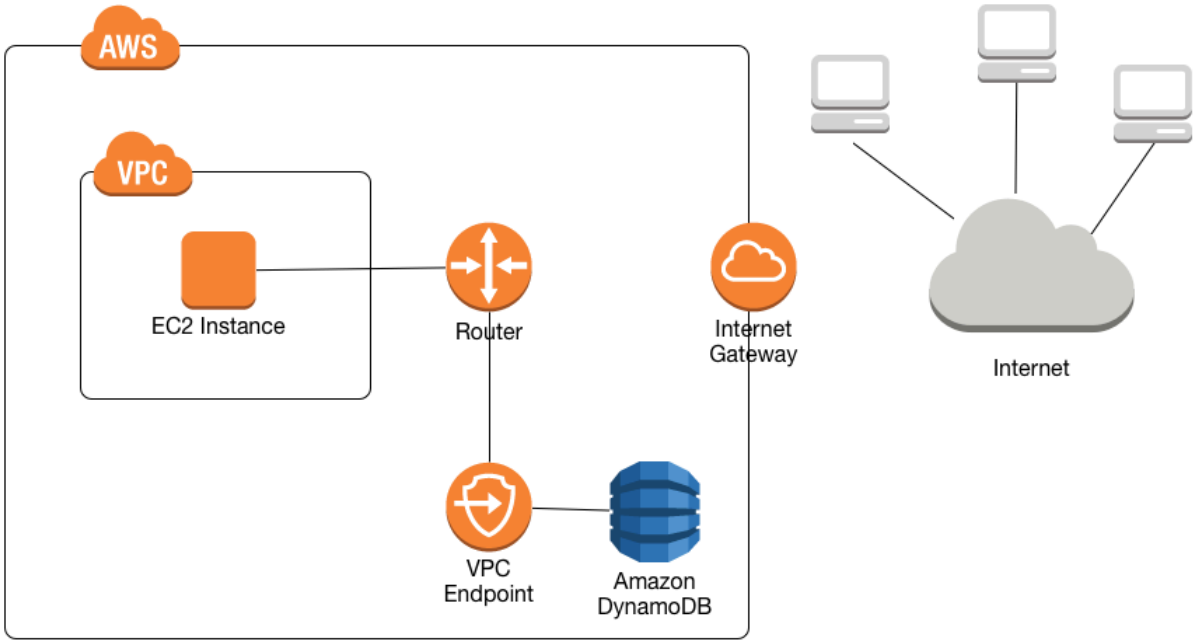
Muitos clientes têm preocupações legítimas com a segurança e a privacidade sobre o envio e o recebimento de dados pela Internet pública. Você pode abordar essas preocupações usando uma rede privada virtual (VPN) para rotear todo o tráfego de rede do DynamoDB por meio da sua infraestrutura de rede corporativa. No entanto, essa abordagem pode introduzir desafios de largura de banda e disponibilidade.

Os endpoints da VPC para DynamoDB podem minimizar esses desafios. Um endpoint da VPC para DynamoDB permite que instâncias do Amazon EC2 em sua VPC usem seus endereços IP privados para acessar o DynamoDB sem se expor à Internet pública. Suas instâncias do EC2 não precisam de endereços IP públicos, e você não precisa de um gateway da Internet, um dispositivo NAT ou um gateway privado virtual em sua VPC. Use as políticas de endpoint para controlar o acesso ao DynamoDB. O tráfego entre sua VPC e o serviço da AWS não sai da rede da Amazon.

> **Note**
>
> Mesmo ao usar endereços IP públicos, todas as comunicações de VPC entre instâncias e serviços hospedados na AWS são mantidas privadas na rede da AWS. Os pacotes originados da rede da AWS com um destino na rede da AWS permanecem na rede global da AWS, exceto o tráfego de ou para as regiões da AWS da China.

Quando você cria um endpoint da VPC para DynamoDB, quaisquer solicitações para um endpoint do DynamoDB dentro da região (por exemplo, dynamodb.us-west-2.amazonaws.com) são encaminhadas para um endpoint do DynamoDB privado dentro da rede da Amazon. Você não precisa modificar suas aplicações executadas em instâncias do EC2 em sua VPC. O nome do endpoint permanece o mesmo, mas a rota para o DynamoDB permanece inteiramente dentro da rede da Amazon, sem acesso à Internet pública.

O diagrama a seguir mostra como uma instância do EC2 em uma VPC pode usar um endpoint da VPC para acessar o DynamoDB.



Para obter mais informações, consulte the section called “Tutorial: usar um endpoint da VPC para o DynamoDB”.

Compartilhar endpoints do Amazon VPC e do DynamoDB

Para permitir o acesso ao serviço DynamoDB por meio do endpoint do gateway de uma sub-rede da VPC, você deve ter permissões de conta de proprietário para essa sub-rede da VPC.

Depois que o endpoint do gateway da sub-rede da VPC tiver acesso ao DynamoDB, qualquer conta da AWS conta com acesso a essa sub-rede poderá usar o DynamoDB. Isso significa que todos os usuários da conta na sub-rede da VPC podem usar qualquer tabela do DynamoDB à qual tenham acesso. Isso inclui tabelas do DynamoDB associadas a uma conta diferente da sub-rede da VPC. O proprietário da sub-rede da VPC ainda pode impedir que qualquer usuário específico dentro da sub-rede use o serviço DynamoDB por meio do endpoint do gateway, se assim ele decidir.

Tutorial: usar um endpoint da VPC para o DynamoDB

Esta seção mostra como configurar e usar um endpoint da VPC para DynamoDB.

Tópicos

- Etapa 1: iniciar uma instância do Amazon EC2

- Etapa 2: configurar a instância do Amazon EC2
- Etapa 3: criar um endpoint da VPC para o DynamoDB
- Etapa 4: limpar (opcional)

Etapa 1: iniciar uma instância do Amazon EC2

Nesta etapa, você inicia uma instância do Amazon EC2 em sua Amazon VPC padrão. Você então pode criar e usar um endpoint da VPC para DynamoDB.

1. Abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. Escolha Iniciar instância e faça o seguinte:

   Etapa 1: escolher uma Imagem de máquina da Amazon (AMI)

   - Na parte superior da lista de AMIs, vá para Amazon Linux AMI (AMI do Amazon Linux) e escolha Select (Selecionar).

   Etapa 2: escolher um tipo de instância

   - Na parte superior da lista de tipos de instância, escolha t2.micro.
   - Escolha Next: Configure Instance Details (Próximo: configurar detalhes da instância).

   Etapa 3: configurar detalhes da instância

   - Vá para Network (Rede) e escolha sua VPC padrão.

     Escolha Next: Add Storage (Próximo: adicionar armazenamento).

   Etapa 4: adicionar armazenamento

   - Ignore esta etapa escolhendo Próximo: instância de tag.

   Etapa 5: instância de tag

   - Ignore esta etapa escolhendo Próximo: configurar o grupo de segurança.

   Etapa 6: configurar o grupo de segurança

- Escolha Select an existing security group (Selecionar um grupo de segurança existente).
- Na lista de grupos de segurança, escolha default (padrão). Este é o grupo de segurança padrão para sua VPC.
- Escolha Próximo: revisar e executar.

Etapa 7: revisar a execução da instância

- Escolha Launch (Executar).

3. Na janela Select an existing key pair or create a new key pair (Selecionar um par de chaves existente ou criar um novo par de chaves), siga um destes procedimentos:

   - Se você não tiver um par de chaves do Amazon EC2, escolha Create a new key pair (Criar um novo par de chaves) e siga as instruções. Você será solicitado a fazer download de um arquivo de chave privada (arquivo .pem), o qual será necessário no momento de fazer login na sua instância do Amazon EC2.
   - Se você já tiver um par de chaves existente do Amazon EC2, vá para Select a key pair (Selecionar um par de chaves) e escolha o seu par de chaves na lista. Você já deverá ter o arquivo de chave privada (arquivo .pem) disponível para fazer login na instância do Amazon EC2.

4. Após configurar o par de chaves, escolha Iniciar instâncias.
5. Retorne à página inicial do console do Amazon EC2 e escolha a instância que você iniciou. No painel inferior, na guia Description (Descrição), localize o Public DNS (DNS público) para sua instância. Por exemplo: `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`.

   Anote esse nome DNS público, pois você precisará dele na próxima etapa deste tutorial (Etapa 2: configurar a instância do Amazon EC2).

> **Note**
>
> Serão necessários alguns minutos para que a sua instância do Amazon EC2 se torne disponível. Antes de ir para a próxima etapa, verifique se o Estado da instância é `running` e se todas as suas Verificações de status foram aprovadas.

Etapa 2: configurar a instância do Amazon EC2

Quando sua instância do Amazon EC2 estiver disponível, você poderá fazer login e prepará-la para ser usada pela primeira vez.

> **Note**
>
> As etapas a seguir assumem que você está se conectando à sua instância do Amazon EC2 de um computador que executa o Linux. Para conferir outras maneiras de se conectar, consulte Conecte-se à sua instância do Linux no Guia do usuário do Amazon EC2.

1. É necessário autorizar o tráfego SSH de entrada para a sua instância do Amazon EC2. Para fazer isso, crie um novo security group EC2 e, em seguida, atribua o security group à sua instância do EC2.

   a. No painel de navegação, escolha Grupos de segurança.

   b. Escolha Criar grupo de segurança. Na janela Security group, faça o seguinte:

      - Nome do security group: digite um nome para seu grupo de segurança. Por exemplo: `my-ssh-access`
      - Descrição – digite uma breve descrição para o security group.
      - VPC – escolha a VPC padrão.
      - Na seção Regras do security group, escolha Adicionar regra e faça o seguinte:
        - Tipo – escolha SSH.
        - Origem – escolha My IP.

      Quando estiver satisfeito com as configurações, escolha Create (Criar).

   c. No painel de navegação, escolha Instâncias.

   d. Escolha a instância do Amazon EC2 que você iniciou em Etapa 1: iniciar uma instância do Amazon EC2.

   e. Escolha Ações --> Rede --> Alterar security groups.

   f. Em Change Security Groups (Alterar grupos de segurança), selecione o grupo de segurança que você criou anteriormente neste procedimento (por exemplo, `my-ssh-access`). O grupo de segurança `default` existente também deve ser selecionado. Quando estiver satisfeito com as configurações, escolha Atribuir security group.

2. Use o comando `ssh` para fazer login na sua instância do Amazon EC2 conforme o exemplo a seguir.

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

   Você precisará especificar seu arquivo de chave privada (arquivo .pem) e o nome DNS público da sua instância. (Consulte Etapa 1: iniciar uma instância do Amazon EC2.)

   O ID de login é `ec2-user`. Nenhuma senha é necessária.

3. Configure as credenciais da AWS, conforme mostrado no exemplo a seguir. Insira o ID da chave de acesso da AWS, chave secreta e nome de região padrão quando solicitado.

```
aws configure
```

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]:
```

Agora, você está pronto para criar um endpoint da VPC para DynamoDB.

Etapa 3: criar um endpoint da VPC para o DynamoDB

Nesta etapa, você criará um endpoint da VPC para DynamoDB e o testará para garantir que ele funcione.

1. Antes de começar, verifique se você pode se comunicar com o DynamoDB usando seu endpoint público:

```
aws dynamodb list-tables
```

   A saída mostrará uma lista de tabelas do DynamoDB que você possui no momento. (Se você não tiver tabelas, a lista estará vazia).

2. Verifique se o DynamoDB é um serviço disponível para a criação de endpoints da VPC na região da AWS atual. (O comando é mostrada em negrito, seguido pelo exemplo de saída).

```
aws ec2 describe-vpc-endpoint-services
```

```
{
    "ServiceNames": [
        "com.amazonaws.us-east-1.s3",
        "com.amazonaws.us-east-1.dynamodb"
    ]
}
```

No exemplo de saída, o DynamoDB é um dos serviços disponíveis, portanto, você pode prosseguir com a criação de um endpoint da VPC para ele.

3. Determine o identificador da VPC.

```
aws ec2 describe-vpcs
```

```
{
    "Vpcs": [
        {
            "VpcId": "vpc-0bbc736e",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

No exemplo de saída, o ID da VPC é `vpc-0bbc736e`.

4. Crie o endpoint da VPC. Para o parâmetro `--vpc-id`, especifique o ID da VPC da etapa anterior. Use o parâmetro `--route-table-ids` para associar o endpoint às tabelas de rotas.

```
aws ec2 create-vpc-endpoint --vpc-id vpc-0bbc736e --service-name com.amazonaws.us-
east-1.dynamodb --route-table-ids rtb-11aa22bb
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
```

```
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [
            "rtb-11aa22bb"
        ],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

5. Verifique se você consegue acessar o DynamoDB por meio do endpoint da VPC.

```
aws dynamodb list-tables
```

Se desejar, experimente alguns outros comandos da AWS CLI para o DynamoDB. Para obter mais informações, consulte Referência de comandos da AWS CLI.

Etapa 4: limpar (opcional)

Para excluir os recursos que você criou neste tutorial, siga estes procedimentos:

Para remover seu endpoint da VPC para DynamoDB

1. Faça login em sua instância do Amazon EC2.
2. Determine o ID do endpoint da VPC.

```
aws ec2 describe-vpc-endpoints
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

No exemplo de saída, o ID do VPC endpoint é `vpce-9b15e2f2`.

3. Exclua o endpoint da VPC.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
}
```

A matriz vazia `[]` indica o sucesso (não há solicitações malsucedidas).

Para terminar sua instância do Amazon EC2

1. Abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. No painel de navegação, escolha Instances (Instâncias).
3. Escolha a instância do Amazon EC2.
4. Escolha Actions (Ações), Instance State (Estado da instância), Terminate (Encerrar).
5. Na janela de confirmação, clique em Sim, encerrar.

AWS PrivateLink para DynamoDB

Com o AWS PrivateLink para o DynamoDB, é possível provisionar endpoints da Amazon VPC de interface (endpoints de interface) em sua nuvem privada virtual (Amazon VPC). Esses endpoints podem ser acessados diretamente por meio da VPN e do AWS Direct Connect pelas aplicações que estão no ambiente on-premises ou por emparelhamento da Amazon VPC pelas aplicações que estão em uma Região da AWS diferente. Usando endpoints de interface e o AWS PrivateLink, é possível simplificar a conectividade de rede privada das aplicações com o DynamoDB.

As aplicações na VPC não necessitam que endereços IP públicos se comuniquem com o DynamoDB usando endpoints de interface da VPC para operações do DynamoDB. Os endpoints de interface são representados por uma ou mais interfaces de rede elástica (ENIs) que recebem endereços IP privados de sub-redes na Amazon VPC. As solicitações para o DynamoDB por meio de endpoints de interface permanecem na rede da Amazon. Também é possível acessar endpoints de interface na Amazon VPC via aplicações on-premises por meio do AWS Direct Connect ou do AWS Virtual Private Network (AWS VPN). Para ter mais informações sobre como conectar a Amazon VPC à rede

on-premises, consulte o Guia do usuário do AWS Direct Connect e o Guia do usuário do AWS Site-to-Site VPN.

Para ter informações gerais sobre endpoints de interface, consulte Interface Amazon VPC endpoints (AWS PrivateLink) no Guia do AWS PrivateLink. O AWS PrivateLink também comporta endpoints do Amazon DynamoDB Streams. Para obter mais informações, consulte the section called “AWS PrivateLink para DynamoDB Streams”.

Tópicos

- Tipos de endpoint da Amazon VPC para o Amazon DynamoDB
- Considerações ao usar o AWS PrivateLink para Amazon DynamoDB
- Criar um Amazon VPC endpoint
- Acessar os endpoints de interface do Amazon DynamoDB
- Acessar tabelas do DynamoDB e operações de API de controle por meio dos endpoints da interface do DynamoDB
- Atualizar uma configuração de DNS on-premises
- Criar uma política de endpoint da Amazon VPC para o DynamoDB
- AWS PrivateLink para DynamoDB Streams

Tipos de endpoint da Amazon VPC para o Amazon DynamoDB

É possível usar dois tipos de endpoints da VPC para acessar o Amazon DynamoDB: endpoints de gateway e endpoints de interface (usando o AWS PrivateLink). Endpoint de gateway é um gateway especificado na tabela de rotas para acessar o DynamoDB por meio da Amazon VPC pela rede da AWS. Os endpoints de interface estendem a funcionalidade dos endpoints de gateway usando endereços IP privados para rotear solicitações para o DynamoDB de dentro da Amazon VPC, do ambiente on-premises ou de uma Amazon VPC em outra Região da AWS usando emparelhamento da VPC ou o AWS Transit Gateway. Para ter mais informações, consulte What is Amazon VPC peering? em Transit Gateway vs Amazon VPC peering.

Os endpoints de interface são compatíveis com os endpoints de gateway. Se você tiver um endpoint de gateway na Amazon VPC, poderá usar os dois tipos de endpoint na mesma Amazon VPC.

Endpoints de gateway para o DynamoDB	Endpoints de interface para o DynamoDB
Em ambos os casos, o tráfego de rede permanece na rede AWS.	

Endpoints de gateway para o DynamoDB	Endpoints de interface para o DynamoDB
Usar endereços IP públicos do Amazon DynamoDB	Usar endereços IP privados da Amazon VPC para acessar o Amazon DynamoDB
Não permita o acesso pelo ambiente on-premises	Permitir acesso desde on-premises
Não permita o acesso por outra Região da AWS	Permitir acesso de um endpoint da Amazon VPC em outra Região da AWS usando emparelhamento da Amazon VPC ou o AWS Transit Gateway
Não faturado	Faturado

Para ter mais informações sobre endpoints de gateway, consulte Gateway Amazon VPC endpoints no Guia do AWS PrivateLink.

Considerações ao usar o AWS PrivateLink para Amazon DynamoDB

As considerações sobre a Amazon VPC se aplicam ao AWS PrivateLink para Amazon DynamoDB. Para obter mais informações, consulte Considerações sobre o endpoint de interface e Cotas do AWS PrivateLink no Guia do usuário do AWS PrivateLink. Além disso, aplicam-se as restrições a seguir.

O AWS PrivateLink para Amazon DynamoDB não é compatível com:

- Transport Layer Security (TLS) 1.1
- Serviços de Sistema de Nomes de Domínio (DNS) privados e híbridos

É possível enviar até 50 mil solicitações por segundo para cada endpoint do AWS PrivateLink habilitado.

> **Note**
>
> Os tempos limite de conectividade de rede com os endpoints do AWS PrivateLink não estão dentro do escopo das respostas de erro do DynamoDB e precisam ser tratados adequadamente pelas aplicações que se conectam aos endpoints do PrivateLink.

Criar um Amazon VPC endpoint

Para criar um endpoint de interface da Amazon VPC, consulte Create an Amazon VPC endpoint no Guia do AWS PrivateLink.

Acessar os endpoints de interface do Amazon DynamoDB

Quando você cria um endpoint de interface, o DynamoDB gera dois tipos de nome de DNS do DynamoDB específicos do endpoint: regional e zonal.

- Os nomes de DNS regionais incluem um ID de endpoint da Amazon VPC exclusivo, um identificador de serviço, a Região da AWS e `vpce.amazonaws.com` no respectivo nome. Por exemplo, para o ID de endpoint da Amazon VPC *`vpce-1a2b3c4d`*, o nome de DNS gerado pode ser semelhante a *`vpce-1a2b3c4d-5e6f`*`.dynamodb.us-east-1.vpce.amazonaws.com`.
- Os nomes DNS zonais incluem a zona de disponibilidade. Por exemplo, *`vpce-1a2b3c4d-5e6f`***`-us-east-1a.`**`dynamodb.us-east-1.vpce.amazonaws.com`. Você pode usar essa opção se sua arquitetura isola zonas de disponibilidade. Por exemplo, você pode usar para contenção de falhas ou para reduzir os custos regionais de transferência de dados.

> **Note**
>
> Para ter a confiabilidade ideal, recomendamos implantar o serviço em três zonas de disponibilidade, no mínimo.

Acessar tabelas do DynamoDB e operações de API de controle por meio dos endpoints da interface do DynamoDB

É possível usar a AWS CLI ou os SDKs da AWS para acessar as tabelas do DynamoDB e controlar as operações da API por meio dos endpoints de interface do DynamoDB.

Exemplos da AWS CLI

Para acessar as tabelas do DynamoDB ou as operações da API de controle do DynamoDB por meio dos endpoints de interface do DynamoDB em comandos da AWS CLI, use os parâmetros `--region` e `--endpoint-url`.

Exemplo: Criar um endpoint da VPC

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Exemplo: Modificar um endpoint da VPC

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Exemplo: Listar tabelas usando um URL de endpoint

No exemplo a seguir, substitua a região `us-east-1` e o nome de DNS do ID de endpoint da VPC `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` por suas próprias informações.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

Exemplos de AWS SDK

Para acessar tabelas do DynamoDB ou operações de API de controle do DynamoDB por meio de endpoints de interface do DynamoDB ao usar os SDKs da AWS, atualize os SDKs para a versão mais recente. Depois, configure os clientes para usar um URL de endpoint para acessar uma tabela ou uma operação de API de controle do DynamoDB por meio de endpoints de interface do DynamoDB.

SDK for Python (Boto3)

Exemplo: Usar um URL de endpoint para acessar uma tabela do DynamoDB

No exemplo a seguir, substitua a região `us-east-1` e o ID de endpoint da VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` por suas próprias informações.

```
ddb_client = session.client(
service_name='dynamodb',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Exemplo: Usar um URL de endpoint para acessar uma tabela do DynamoDB

No exemplo a seguir, substitua a região `us-east-1` e o ID de endpoint da VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` por suas próprias informações.

```
//client build with endpoint config
final AmazonDynamoDB dynamodb =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

Exemplo: Usar um URL de endpoint para acessar uma tabela do DynamoDB

No exemplo a seguir, substitua a região us-east-1 e o ID de endpoint da VPC https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com por suas próprias informações.

```
Region region = Region.US_EAST_1;
dynamoDbClient = DynamoDbClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

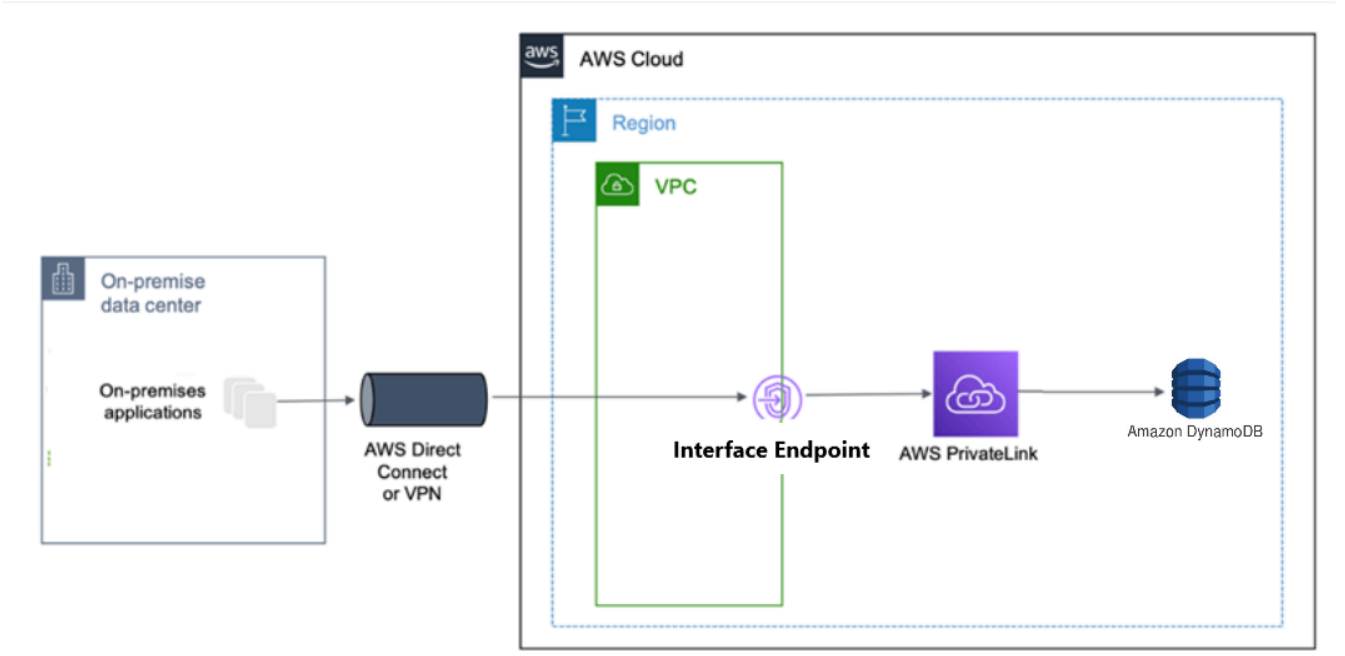
Atualizar uma configuração de DNS on-premises

Ao usar nomes de DNS específicos do endpoint para acessar os endpoints de interface do DynamoDB, não é necessário atualizar seu resolvedor de DNS on-premises. É possível resolver

o nome de DNS específico do endpoint com o endereço IP privado do endpoint de interface pelo domínio de DNS público do DynamoDB.

Usar endpoints de interface para acessar o DynamoDB sem um endpoint de gateway ou um gateway da internet na Amazon VPC

Os endpoints de interface na Amazon VPC podem rotear aplicações na Amazon VPC e aplicações on-premises para o DynamoDB pela rede da Amazon, conforme ilustrado no diagrama a seguir.

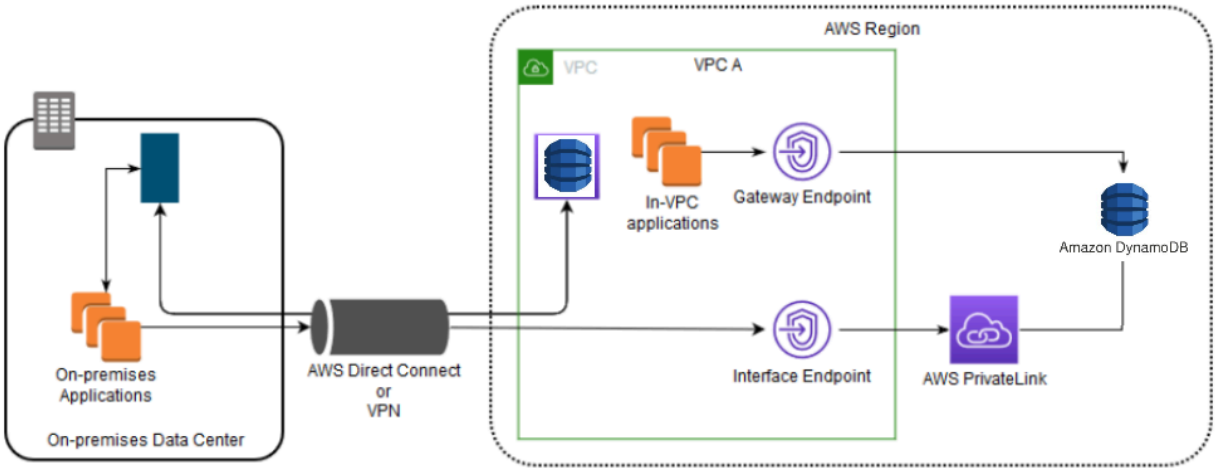


O diagrama ilustra o seguinte:

- Sua rede on-premises usa o AWS Direct Connect ou o AWS VPN para se conectar à Amazon VPC A.
- Suas aplicações on-premises e na Amazon VPC A usam nomes de DNS específicos do endpoint para acessar o DynamoDB por meio do endpoint de interface do DynamoDB.
- As aplicações on-premises enviam dados ao endpoint de interface na Amazon VPC por meio do AWS Direct Connect (ou do AWS VPN). O AWS PrivateLink move os dados do endpoint de interface para o DynamoDB por meio da rede da AWS.
- As aplicações na Amazon VPC também enviam o tráfego ao endpoint de interface. O AWS PrivateLink move os dados do endpoint de interface para o DynamoDB por meio da rede da AWS.

Usar endpoints de gateway e de interface juntos na mesma Amazon VPC para acessar o DynamoDB

É possível criar endpoints de interface e reter o endpoint de gateway existente na mesma Amazon VPC, como mostra o diagrama a seguir. Ao adotar essa abordagem, você permite que as aplicações na Amazon VPC continuem acessando o DynamoDB por meio do endpoint de gateway, que não é cobrado. Depois, apenas as aplicações on-premises usariam endpoints de interface para acessar o DynamoDB. Para acessar o DynamoDB dessa maneira, é necessário atualizar as aplicações on-premises para usar nomes de DNS específicos do endpoint para DynamoDB.



O diagrama ilustra o seguinte:

- As aplicações on-premises usam nomes de DNS específicos do endpoint para enviar dados ao endpoint de interface dentro da Amazon VPC por meio do AWS Direct Connect (ou do AWS VPN). O AWS PrivateLink move os dados do endpoint de interface para o DynamoDB por meio da rede da AWS.
- Usando nomes regionais padrão do DynamoDB, as aplicações na Amazon VPC enviam dados ao endpoint de gateway que se conecta ao DynamoDB pela rede da AWS.

Para ter mais informações sobre endpoints de gateway, consulte Gateway Amazon VPC endpoints no Guia do usuário da Amazon VPC.

Criar uma política de endpoint da Amazon VPC para o DynamoDB

É possível vincular uma política de endpoint ao endpoint da Amazon VPC que controla o acesso ao DynamoDB. Essa política especifica as seguintes informações:

- A entidade principal do AWS Identity and Access Management (IAM) que pode executar ações
- As ações que podem ser executadas
- Os recursos nos quais as ações podem ser executadas

Tópicos

- Exemplo: Restringir o acesso a uma tabela específica por meio de um endpoint da Amazon VPC

Exemplo: Restringir o acesso a uma tabela específica por meio de um endpoint da Amazon VPC

É possível criar uma política de endpoint que restrinja o acesso somente a tabelas específicas do DynamoDB. Esse tipo de política será útil se houver outros Serviços da AWS na Amazon VPC que usem tabelas. A política de tabela a seguir restringe o acesso somente a *DOC-EXAMPLE-TABLE*. Para usar essa política de endpoint, substitua *DOC-EXAMPLE-TABLE* pelo nome da tabela.

JSON

```
{
"Version": "2012-10-17"       ,
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-table-only",
      "Principal": "*",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE",
                   "arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE/*"]
    }
  ]
```

```
}
```

AWS PrivateLink para DynamoDB Streams

Com o AWS PrivateLink para Amazon DynamoDB Streams, é possível provisionar endpoints de interface da Amazon VPC (endpoints de interface) em sua nuvem privada virtual (Amazon VPC). Esses endpoints podem ser acessados diretamente por meio da VPN e do AWS Direct Connect pelas aplicações que estão no ambiente on-premises ou por emparelhamento da Amazon VPC pelas aplicações que estão em uma Região da AWS diferente. Usando o AWS PrivateLink e endpoints de interface, é possível simplificar a conectividade de rede privada das aplicações com o DynamoDB Streams.

As aplicações na Amazon VPC não necessitam que endereços IP públicos se comuniquem com o DynamoDB Streams usando endpoints da interface da Amazon VPC para operações do DynamoDB Streams. Os endpoints de interface são representados por uma ou mais interfaces de rede elástica (ENIs) que recebem endereços IP privados de sub-redes na Amazon VPC. As solicitações ao DynamoDB Streams por meio de endpoints de interface permanecem na rede da Amazon. Também é possível acessar endpoints de interface na Amazon VPC via aplicações on-premises por meio do AWS Direct Connect ou do AWS Virtual Private Network (AWS VPN). Para ter mais informações sobre como conectar sua AWS Virtual Private Network à rede on-premises, consulte o Guia do usuário do AWS Direct Connect e o Guia do usuário do AWS Site-to-Site VPN.

Para ter informações gerais sobre endpoints de interface, consulte Interface Amazon VPC endpoints (AWS PrivateLink).

> **Note**
>
> Somente endpoints de interface são aceitos no DynamoDB Streams. Endpoints de gateway não são aceitos.

Tópicos

- Considerações ao usar o AWS PrivateLink para Amazon DynamoDB Streams
- Criar um Amazon VPC endpoint
- Acessar os endpoints de interface do Amazon DynamoDB Streams
- Acessar operações de API do DynamoDB Streams por meio de endpoints da interface do DynamoDB Streams

- Exemplos de AWS SDK
- Criar uma política de endpoint da Amazon VPC para o DynamoDB Streams

Considerações ao usar o AWS PrivateLink para Amazon DynamoDB Streams

As considerações sobre a Amazon VPC se aplicam ao AWS PrivateLink para Amazon DynamoDB Streams. Para ter mais informações, consulte Interface endpoint considerations e AWS PrivateLink quotas. As restrições a seguir são aplicáveis.

O AWS PrivateLink para Amazon DynamoDB Streams não é compatível com o seguinte:

- Transport Layer Security (TLS) 1.1
- Serviços de Sistema de Nomes de Domínio (DNS) privados e híbridos

> **Note**
>
> Os tempos limite de conectividade de rede com os endpoints do AWS PrivateLink não estão dentro do escopo das respostas de erro do DynamoDB e precisam ser tratados adequadamente pelas aplicações que se conectam aos endpoints do AWS PrivateLink.

Criar um Amazon VPC endpoint

Para criar um endpoint de interface da Amazon VPC, consulte Create an Amazon VPC endpoint no Guia do AWS PrivateLink.

Acessar os endpoints de interface do Amazon DynamoDB Streams

Ao criar um endpoint de interface, o DynamoDB gera dois tipos de nome de DNS do DynamoDB Streams específicos do endpoint: regional e zonal.

- Os nomes de DNS regionais incluem um ID de endpoint da Amazon VPC exclusivo, um identificador de serviço, a Região da AWS e `vpce.amazonaws.com` no respectivo nome. Por exemplo, para o ID de endpoint da Amazon VPC *`vpce-1a2b3c4d`*, o nome de DNS gerado pode ser semelhante a *`vpce-1a2b3c4d-5e6f`*`.streams.dynamodb.us-east-1.vpce.amazonaws.com`.
- Os nomes DNS zonais incluem a zona de disponibilidade. Por exemplo, *`vpce-1a2b3c4d-5e6f`*`-us-east-1a.streams.dynamodb.us-`

`east-1.vpce.amazonaws.com`. Você pode usar essa opção se sua arquitetura isola zonas de disponibilidade. Por exemplo, você pode usar para contenção de falhas ou para reduzir os custos regionais de transferência de dados.

Acessar operações de API do DynamoDB Streams por meio de endpoints da interface do DynamoDB Streams

É possível usar a AWS CLI ou os SDKs da AWS para acessar as operações de API do DynamoDB Streams por meio dos endpoints de interface do DynamoDB Streams.

Exemplos da AWS CLI

Para acessar as operações de API ou do DynamoDB Streams por meio dos endpoints de interface do DynamoDB Streams em comandos da AWS CLI, use os parâmetros `--region` e `--endpoint-url`.

Exemplo: Criar um endpoint da VPC

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb-streams \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Exemplo: Modificar um endpoint da VPC

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Exemplo: listar tabelas usando um URL de endpoint

No exemplo a seguir, substitua a região `us-east-1` e o nome de DNS do ID de endpoint da VPC `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` por suas próprias informações.

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

Exemplos de AWS SDK

Para acessar operações de API do Amazon DynamoDB Streams por meio de endpoints de interface do DynamoDB Streams ao usar os SDKs da AWS, atualize os SDKs para a versão mais recente. Depois, configure os clientes para usar um URL de endpoint para uma operação de API do DynamoDB Streams por meio de endpoints de interface do DynamoDB Streams.

SDK for Python (Boto3)

Exemplo: usar um URL de endpoint para acessar um fluxo do DynamoDB

No exemplo a seguir, substitua a região `us-east-1` e o ID de endpoint da VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` por suas próprias informações.

```
ddb_streams_client = session.client(
service_name='dynamodbstreams',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Exemplo: usar um URL de endpoint para acessar um fluxo do DynamoDB

No exemplo a seguir, substitua a região `us-east-1` e o ID de endpoint da VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` por suas próprias informações.

```
//client build with endpoint config
final AmazonDynamoDBStreams dynamodbstreams =
 AmazonDynamoDBStreamsClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
```

```
        )
).build();
```

SDK for Java 2.x

Exemplo: usar um URL de endpoint para acessar um fluxo do DynamoDB

No exemplo a seguir, substitua a região `us-east-1` e o ID de endpoint da VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` por suas próprias informações.

```
Region region = Region.US_EAST_1;
dynamoDbStreamsClient = DynamoDbStreamsClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

Criar uma política de endpoint da Amazon VPC para o DynamoDB Streams

É possível anexar uma política de endpoint ao endpoint da Amazon VPC que controle o acesso ao DynamoDB Streams. Essa política especifica as seguintes informações:

- A entidade principal do AWS Identity and Access Management (IAM) que pode executar ações
- As ações que podem ser executadas
- Os recursos nos quais as ações podem ser executadas

Tópicos

- Exemplo: restringir o acesso a um fluxo específico por meio de um endpoint da Amazon VPC

Exemplo: restringir o acesso a um fluxo específico por meio de um endpoint da Amazon VPC

É possível criar uma política de endpoint que restrinja o acesso somente a tabelas específicas do DynamoDB Streams. Esse tipo de política será útil se houver outros Serviços da AWS na Amazon VPC que usem o DynamoDB Streams. A política de fluxos a seguir restringe o acesso somente ao fluxo *2025-02-20T11:22:33.444* anexado a *DOC-EXAMPLE-TABLE*. Para usar essa política de endpoint, substitua *DOC-EXAMPLE-TABLE* pelo nome da tabela e *2025-02-20T11:22:33.444* pelo rótulo do fluxo.

JSON

```
{
"Version": "2012-10-17"       ,
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-stream-only",
      "Principal": "*",
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/table-name/
stream/2025-02-20T11:22:33.444"]
    }
  ]
}
```

> **Note**
>
> Endpoints de gateway não são aceitos no DynamoDB Streams.

Análise de configuração e vulnerabilidade no Amazon DynamoDB

A AWS se encarrega das tarefas básicas de segurança, como aplicação de patches a bancos de dados e sistemas operacionais (SOs) convidados, configuração de firewalls e recuperação de desastres. Esses procedimentos foram revisados e certificados por terceiros certificados. Para obter mais detalhes, consulte os seguintes recursos da :

- Validação de conformidade para o Amazon DynamoDB
- Modelo de responsabilidade compartilhada
- Amazon Web Services: Overview of security processes (Amazon Web Services: visão geral do processo de segurança) (whitepaper)

As seguintes práticas recomendadas de segurança também abordam a análise de configuração e vulnerabilidade no Amazon DynamoDB:

- Monitorar a compatibilidade do DynamoDB com o Regras do AWS Config
- Monitorar a configuração do DynamoDB com o AWS Config

Práticas recomendadas de segurança para o Amazon DynamoDB

O Amazon DynamoDB fornece uma série de recursos de segurança a serem considerados no desenvolvimento e na implementação das suas próprias políticas de segurança. As práticas recomendadas a seguir são diretrizes gerais e não representam uma solução completa de segurança. Como essas práticas recomendadas podem não ser adequadas ou suficientes para o seu ambiente, trate-as como considerações úteis em vez de prescrições.

Tópicos

- Práticas recomendadas de segurança preventiva do DynamoDB
- Práticas recomendadas de segurança de detecção do DynamoDB

Práticas recomendadas de segurança preventiva do DynamoDB

As seguintes práticas recomendadas podem ajudar a antecipar e evitar incidentes de segurança no Amazon DynamoDB.

Criptografia em repouso

O DynamoDB criptografa em repouso todos os dados de usuário gravados em tabelas, índices, streams e backups, usando chaves de criptografia armazenadas no AWS Key Management Service (AWS KMS). Isso oferece uma camada de proteção de dados adicional ao proteger seus dados contra acesso não autorizado ao armazenamento subjacente.

Você pode especificar se o DynamoDB deve usar um Chave pertencente à AWS (tipo de criptografia padrão), um Chave gerenciada pela AWS ou uma chave gerenciada pelo cliente para criptografar os dados do usuário. Para obter mais informações, consulte Criptografia em repouso do Amazon DynamoDB.

Usar perfis do IAM para autenticar o acesso ao DynamoDB

Para que usuários, aplicações e outros serviços da AWS possam acessar o DynamoDB, eles devem incluir credenciais da AWS válidas em suas solicitações à AWS API. Você não deve

armazenar credenciais da AWS diretamente na aplicação ou na instância do EC2. Essas são credenciais de longo prazo que não são automaticamente alternadas e, portanto, podem ter impacto comercial significativo se forem comprometidas. Um perfil do IAM permite obter chaves de acesso temporárias que podem ser usadas para acessar os serviços e recursos da AWS.

Para obter mais informações, consulte Gerenciamento de identidade e acesso no Amazon DynamoDB.

Usar políticas o IAM para autorizações de base do DynamoDB

Ao conceder permissões, você decide quem as recebe, a quais APIs do DynamoDB as permissões se referem e as ações específicas que deseja permitir nesses recursos. A implementação do privilégio mínimo é fundamental para reduzir o risco de segurança e o impacto que pode resultar de erros ou usuários mal-intencionados.

Anexe políticas de permissões para identidades do IAM (ou seja, usuários, grupos e perfis) e, assim, dê permissões para eles executarem operações nos recursos do DynamoDB.

Para isso, você pode usar o seguinte:

- Políticas gerenciadas (predefinidas) do AWS
- Políticas gerenciadas pelo cliente

Uso de condições de política do IAM para controle de acesso refinado

Ao conceder permissões no DynamoDB, você pode especificar as condições que determinam como uma política de permissões entra em vigor. A implementação do privilégio mínimo é fundamental para reduzir o risco de segurança e o impacto que pode resultar de erros ou usuários mal-intencionados.

É possível especificar as condições ao conceder permissões usando uma política do IAM. Por exemplo, você pode fazer o seguinte:

- Conceder permissões a fim de permitir acesso somente leitura aos usuários para determinados itens e atributos em uma tabela ou um índice secundário.
- Conceder permissões para permitir somente acesso de gravação aos usuários para determinados atributos em uma tabela com base na identidade desse usuário.

Para obter mais informações, consulte Como usar condições de política do IAM para um controle de acesso refinado.

Usar um endpoint da VPC e políticas para acessar o DynamoDB

Se você só precisa de acesso ao DynamoDB a partir de uma nuvem privada virtual (VPC), use um endpoint da VPC para limitar o acesso somente a partir da VPC requerida. Fazendo isso, você evita que o tráfego atravesse para a Internet pública e esteja sujeito a esse ambiente.

Com um endpoint da VPC para o DynamoDB, você tem controle e limita o acesso usando o seguinte:

- Políticas de endpoint da VPC: essas políticas são aplicadas ao endpoint da VPC do DynamoDB. Elas permitem controle e limite de acesso da API à tabela do DynamoDB.
- Politicas do IAM: ao usar a condição `aws:sourceVpce` nas políticas associadas a usuários, grupos ou perfis, você pode reforçar que todo o acesso à tabela do DynamoDB é feito por meio do endpoint da VPC especificado.

Para obter mais informações, consulte Endpoints para o Amazon DynamoDB.

Considere utilizar a criptografia do lado do cliente

Recomendamos que você planeje sua estratégia de criptografia antes de implementar sua tabela no DynamoDB. Se você armazenar dados confidenciais no DynamoDB, considere incluir a criptografia do lado do cliente em seu plano. Dessa forma, você poderá criptografar os dados o mais próximo possível de sua origem e garantir sua proteção durante todo o ciclo de vida. A criptografia dos seus dados confidenciais em trânsito e em repouso ajuda você a garantir que os dados em texto simples não estejam disponíveis a terceiros.

O SDK de criptografia de banco de dados da AWS para o DynamoDB é uma biblioteca de software que ajuda a proteger os dados de tabela antes que eles sejam enviados ao DynamoDB. Ele criptografa, assina, verifica e descriptografa os itens da tabela do DynamoDB. Você controla quais atributos são criptografados e assinados.

Considerações sobre chave primária

Não use nomes confidenciais nem dados confidenciais em texto simples na chave primária para a tabela e os índices secundários globais. Os nomes das chaves aparecerão na definição da tabela. Por exemplo, os nomes de chave primária podem ser acessados por qualquer pessoa com permissão para chamar DescribeTable. Os valores das chaves podem aparecer no AWS CloudTrail e em outros logs. Além disso, o DynamoDB usa os valores das chaves para distribuir dados e direcionar solicitações, e os administradores da AWS podem observar os valores para manter a integridade do serviço.

Se você precisar usar dados confidenciais na tabela ou valores de chave de GSI, recomendamos usar criptografia de cliente de ponta a ponta. Isso permite que você faça referências de chave-valor aos dados e garanta que eles nunca apareçam sem criptografia nos logs relacionados do DynamoDB. Uma forma de fazer isso é usar o SDK de criptografia de banco de dados da AWS para DynamoDB, mas isso não é obrigatório. Se você usa sua própria solução, sempre é necessário utilizar um algoritmo de criptografia suficientemente seguro. Você não deve usar opções não criptográficas, como um hash, pois, na maioria das situações, elas não são consideradas suficientemente seguras.

Se os nomes de chave primária forem confidenciais, recomendamos usar `pk` e `sk` em vez disso. Essa é uma prática recomendada geral que flexibiliza o design da chave de partição.

Sempre consulte especialistas em segurança ou a equipe de contas da AWS se não souber ao certo qual seria a escolha certa.

Práticas recomendadas de segurança de detecção do DynamoDB

As práticas recomendadas a seguir para o Amazon DynamoDB podem ajudar a detectar pontos fracos e incidentes potenciais de segurança.

Usar o AWS CloudTrail para monitorar o uso de chaves do KMS gerenciadas pela AWS

Se você estiver usando uma Chave gerenciada pela AWS para criptografia em repouso, o uso dessa chave é registrado em log no AWS CloudTrail. O CloudTrail fornece visibilidade da atividade do usuário ao registrar ações realizadas em sua conta. O CloudTrail registra informações importantes sobre cada ação, incluindo quem fez a solicitação, os serviços usados, as ações realizadas, os parâmetros das ações e os elementos de resposta retornados pelo serviço da AWS. Essas informações ajudam você a rastrear as alterações feitas em seus recursos da AWS e solucionar problemas operacionais. O CloudTrail facilita garantir a conformidade com as políticas internas e os padrões regulatórios.

Use o CloudTrail para auditar o uso de chaves. O CloudTrail cria arquivos de log que contêm um histórico de chamadas da AWS API e eventos relacionados da sua conta. Esses arquivos de log incluem todas as solicitações da API do AWS KMS feitas com o AWS Management Console, AWS SDKs e ferramentas da linha de comando, como também pelos serviços integrados da AWS. Você pode usar esses arquivos de log para obter informações sobre quando a chave do KMS foi usada, a operação solicitada, a identidade do solicitante, o endereço IP de origem da

solicitação e assim por diante. Para obter mais informações, consulte Como registrar chamadas de API do AWS KMS com o AWS CloudTrail no Guia do usuário do AWS CloudTrail.

Monitorar operações do DynamoDB usando o CloudTrail

O CloudTrail pode monitorar eventos de ambiente de gerenciamento e eventos de plano de dados. As operações do ambiente de gerenciamento permitem criar e gerenciar tabelas do DynamoDB. Elas também permitem que você trabalhe com índices, fluxos e outros objetos que são dependentes de tabelas. As operações do plano de dados permitem criar, ler, atualizar e excluir (também chamadas de CRUD) nos dados de uma tabela. Algumas das operações de plano de dados também permitem que você leia dados de um índice secundário. Para ativar o registro de eventos de plano de dados no CloudTrail, você precisará ativar o registro da atividade da API do plano de dados no CloudTrail. Para obter mais informações, consulte Log de eventos de dados para trilhas.

Quando uma atividade ocorre no DynamoDB, essa atividade é registrada em um evento do CloudTrail com outros eventos de serviços da AWS no histórico de eventos. Para obter mais informações, consulte Log de operações do DynamoDB usando o AWS CloudTrail. É possível visualizar, pesquisar e baixar eventos recentes em sua conta da AWS. Para obter mais informações, consulte Visualizar eventos com o histórico de eventos CloudTrail no Guia do usuário do AWS CloudTrail.

Para obter um registro contínuo de eventos na sua conta da AWS, incluindo eventos do DynamoDB, crie uma trilha. Uma trilha permite que o CloudTrail entregue arquivos de log a um bucket do Amazon Simple Storage Service (Amazon S3). Por padrão, quando você cria uma trilha no console, ela é aplicada a todas as regiões do AWS. A trilha registra eventos de todas as regiões na partição da AWS e fornece os arquivos de log ao bucket do S3 que você especificar. Além disso, é possível configurar outros serviços da AWS para analisar e agir melhor com base nos dados de eventos coletados nos logs do CloudTrail.

Use o DynamoDB Streams para monitorar operações de plano de dados

O DynamoDB é integrado ao AWS Lambda para que você possa criar acionadores (pedaços de código que respondem automaticamente a eventos no DynamoDB Streams). Com os acionadores, você pode criar aplicações que reagem às modificações de dados em tabelas do DynamoDB.

Se você habilitar o DynamoDB Streams em uma tabela, poderá associar o nome do recurso da Amazon (ARN) do fluxo a uma função do Lambda escrita por você. Imediatamente após um item da tabela ser modificado, um novo registro aparece no fluxo da tabela. O AWS Lambda

pesquisa o fluxo e invoca a função do Lambda de forma síncrona ao detectar novos registros de fluxo. A função do Lambda pode realizar qualquer ação que você especificar, como enviar uma notificação ou iniciar um fluxo de trabalho.

Para ver um exemplo, consulte Tutorial: Como usar o AWS Lambda com o Amazon DynamoDB Streams. Esse exemplo recebe uma entrada de evento do DynamoDB, processa as mensagens que ele contém e grava alguns dados de evento de entrada no Amazon CloudWatch Logs.

Monitorar a configuração do DynamoDB com o AWS Config

Usando o AWS Config, você pode monitorar e gravar alterações de configuração de forma contínua dos seus recursos da AWS. Você também pode usar o AWS Config para fazer o inventário dos seus recursos da AWS. Quando uma alteração em um estado anterior é detectada, uma notificação do Amazon Simple Notification Service (Amazon SNS) pode ser entregue para que você revise e adote as medidas necessárias. Siga as orientações em Configurar uma AWS Config com o console, garantindo que os tipos de recursos do DynamoDB estejam incluídos.

Você pode configurar o AWS Config para transmitir as alterações de configuração e notificações para um tópico do Amazon SNS. Por exemplo, quando um recurso é atualizado, você pode receber uma notificação em seu e-mail, para que possa visualizar as alterações. Você também pode ser notificado quando o AWS Config avaliar suas regras gerenciadas ou personalizadas em relação aos seus recursos.

Para ver um exemplo, consulte Notificações enviadas pelo AWS Config para um tópico do Amazon SNS no Guia do desenvolvedor do AWS Config.

Monitorar a compatibilidade do DynamoDB com regras do AWS Config

O AWS Config rastreia continuamente as alterações de configuração que ocorrem entre seus recursos. Ele verifica se as mudanças violam qualquer condição nas regras. Se um recurso viola uma regra, o AWS Config sinaliza o recurso e a regra como não compatíveis.

Ao usar o AWS Config para avaliar suas configurações de recursos, você pode avaliar se suas configurações de recursos atendem adequadamente a práticas internas e a diretrizes e regulamentações da indústria. O AWS Config oferece regras gerenciadas pela AWS, que são regras predefinidas e personalizáveis que o AWS Config usa para avaliar se seus recursos da AWS estão em conformidade com as práticas recomendadas mais comuns.

Marcar seus recursos do DynamoDB para identificação e automação

Você pode atribuir metadados aos seus recursos da AWS na forma de tags. Cada tag é um rótulo simples que consiste em uma chave definida pelo cliente e um valor opcional que pode facilitar o gerenciamento, a pesquisa e a filtragem de recursos.

A atribuição de tags (tagging) permite a implementação de controles agrupados. Embora não haja tipos de tags inerentes, elas permitem categorizar recursos por finalidade, proprietário, ambiente ou outros critérios. Veja os seguintes exemplos:

- Segurança: usada para determinar requisitos como criptografia.
- Confidencialidade: um identificador do nível de confidencialidade de dados específico permitido por um recurso.
- Ambiente: usado para distinguir entre as infraestruturas de desenvolvimento, teste e produção.

Para obter mais informações, consulte Estratégias de marcação da AWS e Marcação no DynamoDB.

Monitore seu uso do Amazon DynamoDB em relação às práticas recomendadas de segurança usando o AWS Security Hub.

O Security Hub usa controles de segurança para avaliar configurações de recursos e padrões de segurança que ajudam você a cumprir vários frameworks de conformidade.

Para obter mais informações sobre como usar o Security Hub para avaliar os recursos do DynamoDB, consulte Controles do Amazon DynamoDB no Guia do usuário do AWS Security Hub.

Monitorar e registrar em log no DynamoDB

O monitoramento é uma parte importante da manutenção da confiabilidade, disponibilidade e performance do DynamoDB e de suas soluções da AWS. É necessário coletar dados de monitoramento de todas as partes da solução da AWS para facilitar a depuração de uma falha em vários pontos.

Tópicos

- Plano de monitoramento
- Linha de base de performance
- Serviços integrados
- Ferramentas de monitoramento automatizadas
- Monitorar métricas no DynamoDB com o Amazon CloudWatch
- Registrar em log as operações do DynamoDB usando o AWS CloudTrail
- Analisar acesso a dados usando o CloudWatch Contributor Insights para DynamoDB

Plano de monitoramento

Antes de começar a monitorar o DynamoDB, crie um plano de monitoramento que inclua as respostas para as seguintes perguntas:

- Quais são seus objetivos de monitoramento?
- Quais recursos você vai monitorar?
- Com que frequência você vai monitorar esses recursos?
- Quais ferramentas de monitoramento você usará?
- Quem realizará o monitoramento das tarefas?
- Quem deve ser notificado quando algo der errado?

Linha de base de performance

Estabeleça uma referência de performance normal do DynamoDB no ambiente. Para isso, é necessário medir a performance em vários momentos e em diferentes condições de carga. Ao monitorar o DynamoDB, você deve pensar na possibilidade de armazenar os dados históricos de

monitoramento. Esses dados armazenados proporcionam uma linha de base com a qual comparar os dados de desempenho atuais, identificar padrões normais de desempenho e anomalias de desempenho e criar métodos para a solução de problemas. Para estabelecer uma linha de base, é preciso, no mínimo, monitorar os seguintes itens:

- O número de unidades de capacidade de leitura ou gravação consumidas ao longo do período de tempo especificado, para que você possa acompanhar quanto do throughput provisionado foi usado.
- As solicitações que excederam a capacidade de gravação ou de leitura provisionada de uma tabela durante o período especificado, para que você possa determinar as solicitações que excedem as cotas de throughput provisionado de uma tabela.
- Erros de sistema, para que você possa determinar se todas as solicitações resultaram em um erro.

Serviços integrados

O DynamoDB monitora automaticamente as tabelas para você e relata métricas por meio do Amazon CloudWatch. Além disso, o DynamoDB integra-se aos Serviços da AWS a seguir para ajudar a monitorar e solucionar problemas nos recursos do DynamoDB.

- O AWS CloudTrail captura chamadas de API e eventos relacionados feitos por sua conta da Conta da AWS ou em nome dela e entrega os arquivos de log a um bucket do Amazon S3 que você especificar. Para obter mais informações, consulte Registrar em log as operações do DynamoDB usando o AWS CloudTrail.
- O Contributor Insights é uma ferramenta de diagnóstico para identificar rapidamente as chaves com controle de utilização acessadas com maior frequência na tabela ou no índice. Para obter mais informações, consulte Analisar acesso a dados usando o CloudWatch Contributor Insights para DynamoDB.

Ferramentas de monitoramento automatizadas

A AWS fornece várias ferramentas que você pode usar para monitorar o DynamoDB. Recomendamos que as tarefas de monitoramento sejam automatizadas ao máximo possível. Você pode usar as seguintes ferramentas de monitoramento automatizadas para observar o DynamoDB e gerar relatórios quando algo estiver errado:

- Alarmes do Amazon CloudWatch: observe uma única métrica ao longo de um período que você especificar e execute uma ou mais ações com base no valor da métrica em relação a um determinado limite ao longo de vários períodos.

  A ação é uma notificação enviada para um tópico do Amazon Simple Notification Service (Amazon SNS) ou uma política do Amazon EC2 Auto Scaling. Os alarmes do Amazon CloudWatch não invocam ações simplesmente por estarem em um estado específico. O estado deve ter sido alterado e mantido por um número específico de períodos. Para obter mais informações, consulte Monitorar métricas no DynamoDB com o Amazon CloudWatch.
- Monitoramento de logs do AWS CloudTrail: compartilhe arquivos de log entre contas, monitore os arquivos de log do AWS CloudTrail em tempo real enviando-os ao AWS CloudTrail Logs, grave aplicações de processamento de logs em Java e confirme que os arquivos de log não foram alterados após a entrega pelo AWS CloudTrail. Para ter mais informações, consulte What is Amazon CloudWatch Logs no Guia do usuário do AWS CloudTrail.

Monitorar métricas no DynamoDB com o Amazon CloudWatch

É possível monitorar o Amazon DynamoDB usando o CloudWatch, o qual coleta e processa dados brutos do DynamoDB e os transforma em métricas legíveis quase em tempo real. Essas estatísticas são mantidas por um período para que você possa acessar informações históricas e ter uma perspectiva melhor sobre a performance da aplicação web ou do serviço. Por padrão, os dados de métrica do DynamoDB são enviados para o CloudWatch automaticamente. Para obter mais informações, consulte O que é o Amazon CloudWatch? e Retenção de métricas no Guia do usuário do Amazon CloudWatch.

Tópicos

- Como usar as métricas do DynamoDB?
- Visualizar métricas no console do CloudWatch
- Visualizar métricas na AWS CLI
- Métricas e dimensões do DynamoDB
- Criar alarmes do CloudWatch no DynamoDB

Como usar as métricas do DynamoDB?

As métricas relatadas pelo DynamoDB fornecem informações que você pode analisar de diferentes maneiras. A lista a seguir mostra alguns usos comuns para as métricas. Essas são sugestões para você começar, e não uma lista abrangente.

Como usar as métricas do DynamoDB?

Como?	Métricas relevantes
Como monitorar a taxa de exclusões por TTL em minha tabela?	Você pode monitorar `TimeToLiveDeletedItemCount` ao longo do período de tempo especificado, para rastrear a taxa de exclusões de TTL em sua tabela. Para ver um exemplo de uma aplicação sem servidor que usa a métrica `TimeToLiveDeletedItemCount`, consulte Automatically archive items to S3 using DynamoDB time to live (TTL) with AWS Lambda and Amazon Data Firehose.
Como posso determinar quanto do throughput provisionado está sendo usado?	Você pode monitorar `ConsumedReadCapacityUnits` ou `ConsumedWriteCapacityUnits` ao longo do período de tempo especificado, para rastrear a quantidade de throughput provisionado que está sendo usada.
Como posso determinar quais solicitações excedem as cotas de throughput provisionado de uma tabela?	`ThrottledRequests` será incrementado em um se um evento em uma solicitação exceder a cota de um throughput provisionado. Para ter uma ideia sobre qual evento está controlando a utilização de uma solicitação, compare `ThrottledRequests` com as métricas `ReadThrottleEvents` e `WriteThrottleEvents` da tabela e seus índices.
Como posso determinar se ocorreu algum erro do sistema?	Você pode monitorar `SystemErrors` para determinar se todas as solicitações resultaram em um código HTTP 500 (erro de servidor). Normalmente, essa métrica deve ser igual a zero. Se não for o caso, talvez você deva investigar.
Como posso monitorar o valor de latência das minhas operações de tabela?	É possível monitorar `SuccessfulRequestLatency` rastreando a latência média e a latência mediana por meio de

Como?	Métricas relevantes
	métricas de percentil (p50). Picos ocasionais na latência não são motivo de preocupação. No entanto, se a latência média ou p50 (mediana) for alta, poderá haver um problema subjacente a ser resolvido. Consulte Solução de problemas de latência no Amazon DynamoDB para obter mais informações.

Visualizar métricas no console do CloudWatch

As métricas são agrupadas primeiro pelo namespace do serviço e, depois, de acordo com várias combinações de dimensão dentro de cada namespace.

Para exibir métricas no console do CloudWatch

1. Abra o console do CloudWatch, em https://console.aws.amazon.com/cloudwatch/.
2. No painel de navegação, escolha Métricas, Todas as métricas.
3. Selecione o namespace DynamoDB. Também é possível selecionar o namespace Usage (Uso) para visualizar as métricas de uso do DynamoDB. Para obter mais informações sobre métricas de uso, consulte Métricas de uso da AWS.
4. A guia Procurar exibe todas as métricas no namespace.
5. (Opcional) Para adicionar o grafo de métricas a um painel do CloudWatch, escolha Ações, Adicionar ao painel.

Visualizar métricas na AWS CLI

Para ter informações sobre métricas usando a AWS CLI, utilize o comando `list-metrics` do CloudWatch. No exemplo a seguir, você lista todas as métricas no namespace `AWS/DynamoDB`.

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

Para obter estatísticas métricas, use o comando `get-metric-statistics`. O comando a seguir extrai estatísticas `ConsumedReadCapacityUnits` para a tabela `ProductCatalog` durante o período específico de 24 horas, com detalhamento de 5 minutos.

```
aws cloudwatch get-metric-statistics –namespace AWS/DynamoDB \
     –metric-name ConsumedReadCapacityUnits \
     –start-time 2023-11-01T00:00:00Z \
     –end-time 2023-11-02T00:00:00Z \
     –period 360 \
     –statistics Average \
     –dimensions Name=TableName,Value=ProductCatalog
```

A saída da amostra é exibida da seguinte maneira:

```
{
    "Datapoints": [
        {
            "Timestamp": "2023-11-01T 09:18:00+00:00",
            "Average": 20,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 04:36:00+00:00",
            "Average": 22.5,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 15:12:00+00:00",
            "Average": 20,
            "Unit": "Count"
        }, ...
        {
            "Timestamp": "2023-11-01T 17:30:00+00:00",
            "Average": 25,
            "Unit": "Count"
        }
    ],
    "Label": " ConsumedReadCapacityUnits "
}
```

Métricas e dimensões do DynamoDB

Quando você interage com o DynamoDB, ele envia as métricas e as dimensões ao CloudWatch.

As saídas do DynamoDB consumiram o throughput provisionado por períodos de um minuto. O ajuste de escala automático é acionado quando a capacidade consumida ultrapassa a meta de

utilização configurada em dois minutos consecutivos. Os alarmes do CloudWatch podem ter um pequeno atraso de até alguns minutos antes de acionar o ajuste de escala automático. Esse atraso garante uma avaliação precisa da métrica do CloudWatch. Se os picos de throughput consumidos tiverem mais de um minuto de intervalo, o ajuste de escala automático poderá não ser acionado. Da mesma forma, um evento de redução de escala verticalmente pode ocorrer quando 15 pontos de dados consecutivos estão abaixo da meta de utilização. Nos dois casos, depois que o ajuste de escala automático é acionado, a API UpdateTable é invocada. Depois, leva alguns minutos para atualizar a capacidade provisionada da tabela ou do índice. Durante esse período, todas as solicitações que excederem a capacidade provisionada anterior das tabelas terão controle de utilização.

Visualizar métricas e dimensões

O CloudWatch exibe as seguintes métricas do DynamoDB:

Métricas do DynamoDB

> **Note**
>
> O Amazon CloudWatch agrega essas métricas em intervalos de um minuto:
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`
> - `ConsumedWriteCapacityUnits`
> - `ReadAccountLimitThrottleEvents`
> - `ReadKeyRangeThroughputThrottleEvents`
> - `ReadMaxOnDemandThroughputThrottleEvents`
> - `ReadProvisionedThroughputThrottleEvents`
> - `ReadThrottleEvents`
> - `ReturnedBytes`
> - `ReturnedItemCount`
> - `ReturnedRecordsCount`
> - `SuccessfulRequestLatency`
> - `SystemErrors`
> - `TimeToLiveDeletedItemCount`
> - `ThrottledRequests`

- `TransactionConflict`
- `UserErrors`
- `WriteAccountLimitThrottleEvents`
- `WriteKeyRangeThroughputThrottleEvents`
- `WriteMaxOnDemandThroughputThrottleEvents`
- `WriteProvisionedThroughputThrottleEvents`
- `WriteThrottleEvents`

Para todas as métricas do DynamoDB, a granularidade de agregação é de cinco minutos.

Nem todas as estatísticas, como Média ou Soma, são aplicáveis a todas as métricas. No entanto, todos esses valores estão disponíveis por meio do console do Amazon DynamoDB ou usando o console do CloudWatch, AWS CLI ou AWS SDKs para todas as métricas.

Na tabela a seguir, cada métrica tem um conjunto de estatísticas válidas aplicáveis a essa métrica.

Listar métricas disponíveis

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- ConditionalCheckFailedRequests
- ConsumedChangeDataCaptureUnits
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount
- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization

- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress
- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadAccountLimitThrottleEvents
- ReadKeyRangeThroughputThrottleEvents
- ReadMaxOnDemandThroughputThrottleEvents
- ReadProvisionedThroughputThrottleEvents
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteAccountLimitThrottleEvents
- WriteKeyRangeThroughputThrottleEvents
- WriteMaxOnDemandThroughputThrottleEvents
- WriteProvisionedThroughputThrottleEvents

- WriteThrottleEvents

AccountMaxReads

O número máximo de unidades de capacidade de leitura que podem ser usadas por uma conta. Esse limite não se aplica a tabelas sob demanda nem a índices secundários globais.

Unidades: `Count`

Estatística válida:

- `Maximum`: o número máximo de unidades de capacidade de leitura que podem ser usadas por uma conta.

AccountMaxTableLevelReads

O número máximo de unidades de capacidade de leitura que podem ser usadas por uma tabela ou um índice secundário global de uma conta. Em relação a tabelas sob demanda, esse valor limita o máximo de unidades de solicitação de leitura que uma tabela ou um índice secundário global pode usar.

Unidades: `Count`

Estatística válida:

- `Maximum`: o número máximo de unidades de capacidade de leitura que podem ser usadas por uma tabela ou um índice secundário global da conta.

AccountMaxTableLevelWrites

O número máximo de unidades de capacidade de gravação que podem ser usadas por uma tabela ou um índice secundário global de uma conta. Em relação a tabelas sob demanda, esse valor limita o máximo de unidades de solicitação de gravação que uma tabela ou um índice secundário global pode usar.

Unidades: `Count`

Estatística válida:

- `Maximum`: o número máximo de unidades de capacidade de gravação que podem ser usadas por uma tabela ou um índice secundário global da conta.

AccountMaxWrites

O número máximo de unidades de capacidade de gravação que podem ser usadas por uma conta. Esse limite não se aplica a tabelas sob demanda nem a índices secundários globais.

Unidades: `Count`

Estatística válida:

- `Maximum`: o número máximo de unidades de capacidade de gravação que podem ser usadas por uma conta.

AccountProvisionedReadCapacityUtilization

O percentual de unidades de capacidade de leitura provisionada utilizadas por uma conta.

Unidades: `Percent`

Estatística válida:

- `Maximum`: o percentual máximo de unidades de capacidade de leitura provisionada utilizadas pela conta.
- `Minimum`: o percentual mínimo de unidades de capacidade de leitura provisionada utilizadas pela conta.
- `Average`: o percentual médio de unidades de capacidade de leitura provisionada utilizadas pela conta. A métrica é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de leitura provisionada, essa estatística poderá não refletir a média real.

AccountProvisionedWriteCapacityUtilization

A porcentagem de unidades de capacidade de gravação provisionada utilizadas por uma conta.

Unidades: `Percent`

Estatística válida:

- `Maximum`: a porcentagem máxima de unidades de capacidade de gravação provisionada utilizadas pela conta.

- `Minimum`: a porcentagem mínima de unidades de capacidade de gravação provisionada utilizadas pela conta.
- `Average`: a porcentagem média de unidades de capacidade de gravação provisionada utilizadas pela conta. A métrica é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de gravação provisionada, essa estatística poderá não refletir a média real.

AgeOfOldestUnreplicatedRecord

O tempo decorrido desde que um registro que ainda deve ser replicado para o fluxo de dados do Kinesis apareceu pela primeira vez na tabela do DynamoDB.

Unidades: `Milliseconds`

Dimensões: `TableName, DelegatedOperation`

Estatística válida:

- `Maximum`.
- `Minimum`.
- `Average`.

ConditionalCheckFailedRequests

O número de tentativas de executar gravações condicionais que tiveram falha. As operações `PutItem`, `UpdateItem` e `DeleteItem` permitem que você forneça uma condição lógica que deve ser avaliada como true antes que a operação possa prosseguir. Se essa condição for avaliada como false, `ConditionalCheckFailedRequests` será incrementado em uma unidade. `ConditionalCheckFailedRequests` também será incrementado em uma unidade para as instruções PartiQL Update e Delete em que uma condição lógica é fornecida e essa condição é avaliada como false.

> **Note**
>
> Uma gravação condicional com falha resultará em um erro HTTP 400 (Solicitação inválida). Esses eventos são refletidos na métrica`ConditionalCheckFailedRequests`, mas não na métrica `UserErrors`.

Unidades: `Count`

Dimensões: `TableName`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ConsumedChangeDataCaptureUnits

O número de unidades de captura de dados de alteração consumidas.

Unidades: `Count`

Dimensões: `TableName, DelegatedOperation`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`

ConsumedReadCapacityUnits

O número de unidades de capacidade de leitura consumidas ao longo do período especificado para a capacidade provisionada e sob demanda para que você possa acompanhar quanto do throughput foi usado. Você pode recuperar a capacidade de leitura total consumida para uma tabela e todos os seus índices secundários globais ou para um índice secundário global específico. Para obter mais informações, consulte Modo de capacidade de Leitura/Gravação.

A dimensão `TableName` retorna o `ConsumedReadCapacityUnits` para a tabela, mas não para nenhum índice secundário global. Para visualizar `ConsumedReadCapacityUnits` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

> **Note**
>
> Isso significa que picos curtos e intensos no consumo de capacidade que duram apenas um segundo podem não ser refletidos com precisão no grafo do CloudWatch, o que pode indicar uma menor taxa de consumo aparente naquele minuto.
> Use a estatística `Sum` para calcular o throughput consumido. Por exemplo, obtenha o valor `Sum` ao longo de um intervalo de um minuto e divida-o pelo número de segundos em um minuto (60) para calcular a média de `ConsumedReadCapacityUnits` por segundo. Você pode comparar o valor calculado com o valor do throughput provisionado fornecido ao DynamoDB.

Unidades: `Count`

Dimensões: `TableName, GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`: o número mínimo de unidades de capacidade de leitura consumidas por qualquer solicitação individual para a tabela ou o índice.
- `Maximum`: o número máximo de unidades de capacidade de leitura consumidas por qualquer solicitação individual para a tabela ou o índice.
- `Average`: a capacidade de leitura por solicitação média consumida.

  > **Note**
  >
  > O valor `Average` é influenciado por períodos de inatividade em que o valor da amostra será zero.

- `Sum`: o total de unidades de capacidade de leitura consumidas. Essa é a estatística mais útil para a métrica `ConsumedReadCapacityUnits`.
- `SampleCount`: representa a frequência com que a métrica é emitida. Mesmo tabelas com tráfego zero terão a `SampleCount` emitida regularmente, mas os valores da amostra sempre serão zero.

> **Note**
>
> O valor `SampleCount` é influenciado por períodos de inatividade em que o valor da amostra será zero.

ConsumedWriteCapacityUnits

O número de unidades de capacidade de gravação consumidas ao longo do período especificado para a capacidade provisionada e sob demanda para que você possa acompanhar quanto do throughput foi usado. Você pode recuperar a capacidade de gravação total consumida para uma tabela e todos os seus índices secundários globais ou para um índice secundário global específico. Para obter mais informações, consulte Modo de capacidade de Leitura/Gravação.

A dimensão `TableName` retorna o `ConsumedWriteCapacityUnits` para a tabela, mas não para nenhum índice secundário global. Para visualizar `ConsumedWriteCapacityUnits` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

> **Note**
>
> Use a estatística `Sum` para calcular o throughput consumido. Por exemplo, apure o valor `Sum` ao longo de um intervalo de um minuto e divida-o pelo número de segundos em um minuto (60) para calcular a `ConsumedWriteCapacityUnits` média por segundo (reconhecendo que essa média não destaca picos grandes, mas picos breves na atividade de gravação que ocorreram durante esse minuto). Você pode comparar o valor calculado com o valor do throughput provisionado fornecido ao DynamoDB.

Unidades: `Count`

Dimensões: `TableName, GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`: o número mínimo de unidades de capacidade de gravação consumidas por qualquer solicitação individual para a tabela ou o índice.
- `Maximum`: o número máximo de unidades de capacidade de gravação consumidas por qualquer solicitação individual para a tabela ou o índice.

- `Average`: a capacidade de gravação por solicitação média consumida.

> **Note**
>
> O valor `Average` é influenciado por períodos de inatividade em que o valor da amostra será zero.

- `Sum`: o total de unidades de capacidade de gravação consumidas. Essa é a estatística mais útil para a métrica `ConsumedWriteCapacityUnits`.
- `SampleCount`: representa a frequência com que a métrica é emitida. Mesmo tabelas com tráfego zero terão a `SampleCount` emitida regularmente, mas os valores da amostra sempre serão zero.

> **Note**
>
> O valor `SampleCount` é influenciado por períodos de inatividade em que o valor da amostra será zero.

FailedToReplicateRecordCount

Número de registros que o DynamoDB não conseguiu replicar no fluxo de dados do Kinesis.

Unidades: `Count`

Dimensões: `TableName`, `DelegatedOperation`

Estatística válida:

- `Sum`

MaxProvisionedTableReadCapacityUtilization

O percentual da capacidade de leitura provisionada utilizada pela tabela de leitura provisionada mais alta ou pelo índice secundário global de uma conta.

Unidades: `Percent`

Estatística válida:

- `Maximum`: a porcentagem máxima de unidades de capacidade de leitura provisionada utilizada pela tabela de leitura provisionada mais alta ou pelo índice secundário global de uma conta.

- `Minimum`: a porcentagem mínima de unidades de capacidade de leitura provisionada utilizada pela tabela de leitura provisionada mais alta ou pelo índice secundário global de uma conta.
- `Average`: a porcentagem média de unidades capacidade de leitura provisionada utilizadas pela tabela de gravação provisionada mais alta ou pelo índice secundário global da conta. A métrica é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de leitura provisionada, essa estatística poderá não refletir a média real.

MaxProvisionedTableWriteCapacityUtilization

A porcentagem da capacidade de gravação provisionada utilizada pela tabela de gravação provisionada mais alta ou pelo índice secundário global de uma conta.

Unidades: `Percent`

Estatística válida:

- `Maximum`: a porcentagem máxima de unidades capacidade de gravação provisionada utilizadas pela tabela de gravação provisionada mais alta ou pelo índice secundário global de uma conta.
- `Minimum`: a porcentagem mínima de unidades capacidade de gravação provisionada utilizadas pela tabela de gravação provisionada mais alta ou pelo índice secundário global de uma conta.
- `Average`: a porcentagem média de unidades capacidade de gravação provisionada utilizadas pela tabela de gravação provisionada mais alta ou pelo índice secundário global da conta. A métrica é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de gravação provisionada, essa estatística poderá não refletir a média real.

OnDemandMaxReadRequestUnits

O número de unidades de solicitação de leitura sob demanda especificado para uma tabela ou um índice secundário global.

Para visualizar `OnDemandMaxReadRequestUnits` referente a uma tabela, é necessário especificar `TableName`. Para visualizar `OnDemandMaxReadRequestUnits` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

Unidades: contagem

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`: a configuração mais baixa para unidades de solicitação de leitura sob demanda. Se você usar `UpdateTable` para aumentar as unidades de solicitação de leitura, essa métrica mostrará o valor mais baixo de `ReadRequestUnits` sob demanda durante esse período.
- `Maximum`: a configuração mais alta para unidades de solicitação de leitura sob demanda. Se você usar `UpdateTable` para reduzir as unidades de solicitação de leitura, essa métrica mostrará o valor mais alto de `ReadRequestUnits` sob demanda durante esse período.
- `Average`: a média de unidades de solicitação de leitura sob demanda. A métrica `OnDemandMaxReadRequestUnits` é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de solicitação de leitura sob demanda, essa estatística poderá não refletir a média real.

OnDemandMaxWriteRequestUnits

O número de unidades de solicitação de gravação sob demanda especificado para uma tabela ou um índice secundário global.

Para visualizar `OnDemandMaxWriteRequestUnits` referente a uma tabela, é necessário especificar `TableName`. Para visualizar `OnDemandMaxWriteRequestUnits` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`: a configuração mais baixa para unidades de solicitação de gravação sob demanda. Se você usar `UpdateTable` para aumentar as unidades de solicitação de gravação, essa métrica mostrará o valor mais baixo de `WriteRequestUnits` sob demanda durante esse período.
- `Maximum`: a configuração mais alta para unidades de solicitação de gravação sob demanda. Se você usar `UpdateTable` para reduzir as unidades de solicitação de gravação, essa métrica mostrará o valor mais alto de `WriteRequestUnits` sob demanda durante esse período.
- `Average`: a média de unidades de solicitação de gravação sob demanda. A métrica `OnDemandMaxWriteRequestUnits` é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de solicitação de gravação sob demanda, essa estatística poderá não refletir a média real.

OnlineIndexConsumedWriteCapacity

O número de unidades de capacidade de gravação consumidas ao adicionar um novo índice secundário global a uma tabela. Se a capacidade de gravação do índice for muito baixa, a atividade de gravação de entrada durante a fase de alocação poderá ser limitada. Isso pode aumentar o tempo necessário para criar o índice. Você deve monitorar essa estatística enquanto o índice está sendo criado para determinar se a capacidade de gravação do índice está subprovisionada.

É possível ajustar a capacidade de gravação do índice usando a operação `UpdateTable`, mesmo enquanto o índice ainda está sendo construído.

A métrica `ConsumedWriteCapacityUnits` para o índice não inclui o throughput de gravação consumido durante a criação do índice.

> **Note**
>
> Essa métrica poderá não ser emitida se a fase de preenchimento do novo índice secundário global for concluída rapidamente (menos de alguns minutos), o que poderá ocorrer se a tabela base tiver poucos ou nenhum item para preencher no índice.

Unidades: `Count`

Dimensões: `TableName, GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexPercentageProgress

A porcentagem de conclusão quando um novo índice secundário global está sendo adicionado a uma tabela. O DynamoDB deve primeiro alocar recursos para o novo índice e, em seguida, alocar atributos da tabela para o índice. Para tabelas grandes, esse processo pode ser demorado. Você

deve monitorar essa estatística para visualizar o progresso relativo à medida que o DynamoDB constrói o índice.

Unidades: `Count`

Dimensões: `TableName, GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexThrottleEvents

O número de eventos de limitação de gravação que ocorrem ao adicionar um novo índice secundário global a uma tabela. Esses eventos indicam que a criação do índice levará mais tempo para ser concluída, pois a atividade de gravação de entrada está excedendo throughput de gravação provisionado do índice.

É possível ajustar a capacidade de gravação do índice usando a operação `UpdateTable`, mesmo enquanto o índice ainda está sendo construído.

A métrica `WriteThrottleEvents` para o índice não inclui eventos de controle de utilização ocorridos durante a criação do índice.

Unidades: `Count`

Dimensões: `TableName, GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

PendingReplicationCount

Métrica para Global Tables versão 2017.11.29 (herdada) (somente tabelas globais). O número de atualizações de itens que foram gravadas em uma tabela de réplica, mas ainda não foram gravadas em outra réplica na tabela global.

Unidades: `Count`

Dimensões: `TableName, ReceivingRegion`

Estatística válida:

- `Average`
- `Sample Count`
- `Sum`

ProvisionedReadCapacityUnits

O número de unidades de capacidade de leitura provisionada para uma tabela ou um índice secundário global. A dimensão `TableName` retorna o `ProvisionedReadCapacityUnits` para a tabela, mas não para nenhum índice secundário global. Para visualizar `ProvisionedReadCapacityUnits` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

Unidades:`Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`: a configuração mais baixa para a capacidade de leitura provisionada. Se você usar `UpdateTable` para aumentar a capacidade de leitura, esta métrica mostrará o valor mais baixo de `ReadCapacityUnits` provisionado durante esse período.
- `Maximum`: a configuração mais alta para a capacidade de leitura provisionada. Se você usar `UpdateTable` para diminuir a capacidade de leitura, esta métrica mostrará o valor mais alto de `ReadCapacityUnits` provisionado durante esse período.
- `Average`: a capacidade média de leitura provisionada. A métrica `ProvisionedReadCapacityUnits` é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de leitura provisionada, essa estatística poderá não refletir a média real.

ProvisionedWriteCapacityUnits

O número de unidades de capacidade de gravação provisionada para uma tabela ou um índice secundário global.

A dimensão `TableName` retorna o `ProvisionedWriteCapacityUnits` para a tabela, mas não para nenhum índice secundário global. Para visualizar `ProvisionedWriteCapacityUnits` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Minimum`: a configuração mais baixa para capacidade de gravação provisionada. Se você usar `UpdateTable` para aumentar a capacidade de gravação, esta métrica mostrará o valor mais baixo de `WriteCapacityUnits` provisionado durante esse período.
- `Maximum`: a configuração mais alta para a capacidade de gravação provisionada. Se você usar `UpdateTable` para diminuir a capacidade de gravação, esta métrica mostrará o valor mais alto de `WriteCapacityUnits` provisionado durante esse período.
- `Average`: a capacidade média de gravação provisionada. A métrica `ProvisionedWriteCapacityUnits` é publicada para intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de gravação provisionada, essa estatística poderá não refletir a média real.

ReadAccountLimitThrottleEvents

O número de solicitações de leitura com controle de utilização devido aos limites da conta.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.

- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

ReadKeyRangeThroughputThrottleEvents

Número de solicitações de leitura com controle de utilização devido aos limites de partição.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.
- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

ReadMaxOnDemandThroughputThrottleEvents

O número de solicitações de leitura com controle de utilização devido ao throughput máximo sob demanda.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.
- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

ReadProvisionedThroughputThrottleEvents

O número de solicitações de leitura com controle de utilização devido aos limites de throughput provisionado.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.
- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

ReadThrottleEvents

Solicitações ao DynamoDB que excedem as unidades de capacidade de leitura provisionada para uma tabela ou um índice secundário global.

Uma única solicitação pode resultar em vários eventos. Por exemplo, um `BatchGetItem` que lê 10 itens é processado como 10 eventos `GetItem`. Para cada evento, `ReadThrottleEvents` será incrementado em uma unidade se esse evento for limitado. A métrica `ThrottledRequests` para todo o `BatchGetItem` não será incrementada, a menos que todos os 10 dos eventos `GetItem`sejam limitados.

A dimensão `TableName` retorna o `ReadThrottleEvents` para a tabela, mas não para nenhum índice secundário global. Para visualizar `ReadThrottleEvents` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `SampleCount`
- `Sum`

ReplicationLatency

(Esta métrica é para tabelas globais do DynamoDB.) O tempo decorrido entre um item atualizado aparecer no fluxo do DynamoDB para uma tabela-réplica e aparecer em outra réplica na tabela global.

Unidades: `Milliseconds`

Dimensões: `TableName, ReceivingRegion`

Estatística válida:

- `Average`
- `Minimum`
- `Maximum`

ReturnedBytes

O número de bytes retornados por operações `GetRecords` (Amazon DynamoDB Streams) durante o período especificado.

Unidades: `Bytes`

Dimensões: `Operation, StreamLabel, TableName`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedItemCount

A quantidade de itens retornados por operações `Query`, `Scan` ou `ExecuteStatement` (selecionar) durante o período especificado.

O número de itens retornados não necessariamente é o mesmo que o número de itens avaliados. Por exemplo, suponha que você tenha solicitado um `Scan` em uma tabela ou um índice que tinha 100 itens, mas especificou um `FilterExpression`onque reduziu os resultados para que apenas 15 itens fossem retornados. Nesse caso, a resposta de `Scan` conteria um `ScanCount` de 100 e um `Count` de 15 itens retornados.

Unidades: `Count`

Dimensões: `TableName, Operation`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedRecordsCount

O número de registros de fluxo retornados por operações `GetRecords` (Amazon DynamoDB Streams) durante o período especificado.

Unidades: `Count`

Dimensões: `Operation, StreamLabel, TableName`

Estatística válida:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

SuccessfulRequestLatency

As solicitações bem-sucedidas para o DynamoDB ou Amazon DynamoDB Streams durante o período especificado. O `SuccessfulRequestLatency` pode fornecer dois tipos diferentes de informações:

- O tempo decorrido para solicitações bem-sucedidas (`Minimum`, `Maximum`, `Sum`, `Average` ou `Percentile`).

- O número de solicitações bem-sucedidas (`SampleCount`).

`SuccessfulRequestLatency` reflete a atividade somente no DynamoDB ou no Amazon DynamoDB Streams e não leva em conta a latência da rede nem a atividade no lado do cliente.

> **Note**
>
> Para analisar valores de percentis personalizados (como p99,9), você pode inserir manualmente o percentil desejado (p. ex., p99,9) no campo de estatística de métrica do CloudWatch. Isso permite avaliar as distribuições de latência que transcendem os percentis padrão listados no menu suspenso.

Unidades: `Milliseconds`

Dimensões: `TableName, Operation, StreamLabel`

Estatística válida:

- `Minimum`
- `Maximum`
- `Sum`
- `Average`
- `Percentile`
- `SampleCount`

SystemErrors

As solicitações ao DynamoDB ou Amazon DynamoDB Streams que geram um código de status HTTP 500 durante o período especificado. Um HTTP 500 geralmente indica um erro de serviço interno.

> **Note**
>
> Quando o DynamoDB exibe um erro do sistema (HTTP 500), a maioria dos SDKs da AWS executa automaticamente um número configurável de novas tentativas. Se o problema for resolvido durante uma nova tentativa, a aplicação continuará sem ver o erro e você pode

notar um aumento na latência percebida do lado do cliente. Se o erro persistir após todas as tentativas, ele se propagará para o código da aplicação.

Unidades: `Count`

Dimensões: `TableName, Operation`

Estatística válida:

- `Sum`
- `SampleCount`

TimeToLiveDeletedItemCount

A quantidade de itens excluídos por vida útil (TTL) durante o período especificado. Essa métrica ajuda a monitorar a taxa de exclusões de TTL em sua tabela.

Unidades: `Count`

Dimensões: TableName

Estatística válida:

- `Sum`

ThrottledPutRecordCount

Número de registros que foram suprimidos pelo seu fluxo de dados do Kinesis devido a capacidade insuficiente do Kinesis Data Streams.

Unidades: `Count`

Dimensões: TableName, DelegateDoperation

Estatística válida:

- `Minimum`
- `Maximum`

- `Average`
- `SampleCount`

ThrottledRequests

Solicitações ao DynamoDB que excedem os limites de throughput provisionado em um recurso (como uma tabela ou um índice).

`ThrottledRequests` será incrementado em uma unidade se um evento em uma solicitação exceder o limite de throughput provisionado. Por exemplo, se você atualizar um item em uma tabela com índices secundários globais, haverá vários eventos — uma gravação na tabela e uma gravação em cada índice. Se um ou mais desses eventos for limitado, `ThrottledRequests` será incrementado em uma unidade.

> **Note**
>
> Em uma solicitação de lote (`BatchGetItem` ou `BatchWriteItem`), `ThrottledRequests` será incrementado somente se houver limitação para cada uma das solicitações no lote. Se qualquer solicitação individual dentro do lote for limitada, uma das seguintes métricas será incrementada:
>
> - `ReadThrottleEvents`: para um evento `GetItem` limitado em `BatchGetItem`.
> - `WriteThrottleEvents`: para um evento `PutItem` ou `DeleteItem` limitado em `BatchWriteItem`.

Para ter uma ideia sobre qual evento está controlando a utilização de uma solicitação, compare `ThrottledRequests` com `ReadThrottleEvents` e `WriteThrottleEvents` para a tabela e seus índices.

> **Note**
>
> Uma solicitação com limitação resultará em um código de status HTTP 400. Esses eventos são refletidos na métrica `ThrottledRequests`, mas não na métrica `UserErrors`.

Unidades: `Count`

Dimensões: `TableName, Operation`

Estatística válida:

- `Sum`
- `SampleCount`

TransactionConflict

Solicitações em nível de item rejeitadas devido a conflitos transacionais entre solicitações concorrentes nos mesmos itens. Para obter mais informações, consulte Tratamento de conflitos de transações no DynamoDB.

Unidades: `Count`

Dimensões: `TableName`

Estatística válida:

- `Sum`: o número de solicitações rejeitadas em nível do item devido a conflitos de transação.

> **Note**
>
> Se várias solicitações de nível de item dentro de uma chamada para `TransactWriteItems` ou `TransactGetItems` foram rejeitadas, `Sum` será incrementado em uma unidade para cada solicitação `Put`, `Update`, `Delete` ou `Get` em nível de item.

- `SampleCount`: o número de solicitações rejeitadas devido a conflitos de transação.

> **Note**
>
> Se várias solicitações de nível de item dentro de uma chamada para `TransactWriteItems` ou `TransactGetItems` forem rejeitadas, `SampleCount`só será incrementado em uma unidade.

- `Min`: o número mínimo de solicitações em nível de item rejeitadas em uma chamada para `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` ou `DeleteItem`.
- `Max`: o número máximo de solicitações em nível de item rejeitadas em uma chamada para `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` ou `DeleteItem`.

- `Average`: o número médio de solicitações em nível de item rejeitadas em uma chamada para `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` ou `DeleteItem`.

UserErrors

Solicitações ao DynamoDB ou Amazon DynamoDB Streams que geram um código de status HTTP 400 durante o período especificado. Um código HTTP 400 geralmente indica um erro no lado do cliente, como uma combinação inválida de parâmetros, uma tentativa de atualizar uma tabela inexistente ou uma assinatura de solicitação incorreta.

Alguns exemplos de exceções que registrarão em log métricas relacionadas a `UserErrors`:

- `ResourceNotFoundException`
- `ValidationException`
- `TransactionConflict`

Todos esses eventos são refletidos na métrica `UserErrors`, com exceção de:

- ProvisionedThroughputExceededException: consulte a métrica `ThrottledRequests` nesta seção.
- ConditionalCheckFailedException: consulte a métrica `ConditionalCheckFailedRequests` nesta seção.

`UserErrors` representa o agregado de erros HTTP 400 para solicitações do DynamoDB ou do Amazon DynamoDB Streams para a região da AWS atual e a conta da AWS atual.

Unidades: `Count`

Estatística válida:

- `Sum`
- `SampleCount`

WriteAccountLimitThrottleEvents

O número de solicitações de gravação com controle de utilização devido aos limites da conta.

Unidades: `Count`

Dimensões: `TableName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.
- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

WriteKeyRangeThroughputThrottleEvents

O número de solicitações de gravação com controle de utilização devido aos limites de partição.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.
- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

WriteMaxOnDemandThroughputThrottleEvents

O número de solicitações de gravação com controle de utilização devido ao throughput máximo sob demanda.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.
- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

WriteProvisionedThroughputThrottleEvents

O número de solicitações de gravação com controle de utilização devido aos limites de throughput provisionado.

Unidades: `Count`

Dimensões: `TableName`, `GlobalSecondaryIndexName`

Estatística válida:

- `Sum`: número total de eventos com controle de utilização.
- `SampleCount`: número de ocorrências de controle de utilização.
- `Minimum`: número mínimo de eventos com controle de utilização em qualquer amostra.
- `Maximum`: número máximo de eventos com controle de utilização em qualquer amostra.

WriteThrottleEvents

Solicitações ao DynamoDB que excedem as unidades de capacidade de gravação provisionada para uma tabela ou um índice secundário global.

Uma única solicitação pode resultar em vários eventos. Por exemplo, uma solicitação `PutItem` em uma tabela com três índices secundários globais resultaria em quatro eventos: a gravação da tabela e cada uma das três gravações do índice. Para cada evento, a métrica `WriteThrottleEvents` será incrementada em uma unidade se esse evento for limitado. Para solicitações `PutItem` individuais, se houver limitação para qualquer um dos eventos, `ThrottledRequests` também será incrementado em uma unidade. Para `BatchWriteItem`, a métrica `ThrottledRequests` para todo o `BatchWriteItem` não será incrementada, a menos que todos os eventos `PutItem` ou `DeleteItem` individuais tenham limitação.

A dimensão `TableName` retorna o `WriteThrottleEvents` para a tabela, mas não para nenhum índice secundário global. Para visualizar `WriteThrottleEvents` para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensões: `TableName, GlobalSecondaryIndexName`

Estatística válida:

- `Sum`
- `SampleCount`

Métricas de uso

As métricas de uso no CloudWatch permitem gerenciar o uso visualizando proativamente métricas no console do CloudWatch, criando painéis personalizados, detectando alterações na atividade com a detecção de anomalias do CloudWatch e configurando alarmes que avisam quando o uso se aproxima de um limite.

O DynamoDB também integra essas métricas de uso ao Service Quotas. É possível usar o CloudWatch para gerenciar o uso das service quotas por sua conta. Para obter mais informações, consulte Visualizar service quotas e definir alarmes.

Lista de métricas de uso disponíveis

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

A soma de unidades de capacidade de gravação provisionadas para todos os índices secundários globais e tabelas de uma conta.

Unidades: `Count`

Estatística válida:

- `Minimum`: o menor número de unidades de capacidade de gravação provisionadas durante um período.
- `Maximum`: o maior número de unidades de capacidade de gravação provisionadas durante um período.
- `Average`: o número médio de unidades de capacidade de gravação provisionadas durante um período.

A métrica é publicada em intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de gravação provisionada, essa estatística poderá não refletir a média real.

AccountProvisionedReadCapacityUnits

A soma de unidades de capacidade de leitura provisionadas para todos os índices secundários globais e tabelas de uma conta.

Unidades: `Count`

Estatística válida:

- `Minimum`: o menor número de unidades de capacidade de leitura provisionadas durante um período.
- `Maximum`: o maior número de unidades de capacidade de leitura provisionadas durante um período.
- `Average`: o número médio de unidades de capacidade de leitura provisionadas durante um período.

A métrica é publicada em intervalos de cinco minutos. Portanto, se você ajustar rapidamente as unidades de capacidade de leitura provisionada, essa estatística poderá não refletir a média real.

TableCount

O número de tabelas ativas de uma conta.

Unidades: `Count`

Estatística válida:

- `Minimum`: o menor número de tabelas durante um período.
- `Maximum`: o maior número de tabelas durante um período.
- `Average`: o número médio de tabelas durante um período.

Informações sobre métricas e dimensões para o DynamoDB

As métricas para o DynamoDB são qualificadas de acordo com os valores para a conta, nome da tabela, nome do índice secundário global ou operação. Você pode usar o console do CloudWatch para recuperar dados do DynamoDB em qualquer uma das dimensões da tabela abaixo.

Listar dimensões disponíveis

- DelegatedOperation

- GlobalSecondaryIndexName
- Operação
- OperationType
- Verb
- ReceivingRegion
- StreamLabel
- TableName

DelegatedOperation

Essa dimensão limita os dados às operações que o DynamoDB executa em seu nome. As seguintes operações são capturadas:

- Alterar captura de dados para o Kinesis Data Streams.

GlobalSecondaryIndexName

Esta dimensão limita os dados a um índice secundário global em uma tabela. Se você especificar `GlobalSecondaryIndexName`, também deverá especificar `TableName`.

Operação

Esta dimensão limita os dados a uma das seguintes operações do DynamoDB:

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`

- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

Além disso, você pode limitar os dados à seguinte operação do Amazon DynamoDB Streams:

- `GetRecords`

OperationType

Esta dimensão limita os dados a um dos seguintes tipos de operação:

- `Read`
- `Write`

Esta dimensão é emitida para solicitações `ExecuteTransaction` e `BatchExecuteStatement`.

Verb

Esta dimensão limita os dados a um dos seguintes verbos PartiQL do DynamoDB:

- Inserir: `PartiQLInsert`
- Selecionar: `PartiQLSelect`
- Atualizar: `PartiQLUpdate`
- Excluir: `PartiQLDelete`

Essa dimensão é emitida para a operação `ExecuteStatement`.

ReceivingRegion

Esta dimensão limita os dados a uma região da AWS específica. Ela é usada com métricas provenientes de tabelas de réplica em uma tabela global do DynamoDB.

StreamLabel

Esta dimensão limita os dados a um rótulo de fluxo específico. Ela é usada com métricas provenientes de operações `GetRecords` do Amazon DynamoDB Streams

TableName

Esta dimensão limita os dados a uma tabela específica. Este valor pode ser qualquer nome de tabela na região atual e na conta da AWS atual.

Criar alarmes do CloudWatch no DynamoDB

Um alarme do CloudWatch observa uma única métrica ao longo de um período especificado e realiza uma ou mais ações especificadas com base no valor da métrica em relação a um limite ao longo do tempo. A ação é uma notificação enviada para um tópico do Amazon SNS ou uma política de Auto Scaling. Também é possível adicionar alarmes aos painéis para monitorar e receber alertas sobre seus recursos da AWS e aplicações em várias regiões. Não há limite para o número de alarmes que você pode criar. Os alarmes do CloudWatch não invocam ações simplesmente por estarem em um estado específico. O estado deve ter sido alterado e mantido por um número específico de períodos. Para ver uma lista dos alarmes recomendados do DynamoDB, consulte Alarmes recomendados.

> **Note**
>
> Você deve especificar todas as dimensões necessárias ao criar o alarme do CloudWatch, pois o CloudWatch não agregará métricas para uma dimensão ausente. Criar um alarme do CloudWatch com uma dimensão ausente não resultará em um erro na criação do alarme.

Vamos supor que você tenha uma tabela provisionada com cinco unidades de capacidade de leitura. Você quer receber notificação antes de consumir toda a capacidade de leitura provisionada, então decide criar um alarme do CloudWatch para receber notificação quando a capacidade consumida atingir 80% do que você provisionou para a tabela. É possível criar alarmes usando o console do CloudWatch ou usando a AWS CLI.

Criar um alarme no console do CloudWatch

Como criar um alarme no console do CloudWatch

1. Faça login no AWS Management Console e abra o console do CloudWatch em https://console.aws.amazon.com/cloudwatch/.
2. No painel de navegação, escolha Alarms (Alarmes), All alarms (Todos os alarmes).
3. Selecione Criar alarme.

4. Encontre a linha com a tabela que você deseja monitorar e **`ConsumeReadCapacityUnits`** na coluna Nome da métrica. Marque a caixa de seleção ao lado dessa linha e escolha Selecionar métrica.
5. Em Especificar métrica e condições, em Estatística, escolha Soma. Selecione Período de 1 minuto.
6. Em Conditions (Condições), especifique o seguinte:
   a. Em Tipo de limite, escolha Estático.
   b. Em Sempre que **`ConsumedReadCapacityUnits`** for, escolha Maior que/igual a e especifique o limite como 240.
7. Escolha Próximo.
8. Em Notificação, selecione **`In alarm`** e um tópico do SNS para notificar quando o alarme estiver no estado `ALARM`.
9. Quando terminar, escolha Next (Próximo).
10. Insira um nome e uma descrição para o alarme e selecione Next (Avançar).
11. Em Preview and create (Visualizar e criar), confirme se as informações e condições são o que você deseja e escolha Create alarm (Criar alarme).

Criar um alarme na AWS CLI

```
aws cloudwatch put-metric-alarm \
    -\-alarm-name ReadCapacityUnitsLimitAlarm \
    -\-alarm-description "Alarm when read capacity reaches 80% of my provisioned read
 capacity" \
    -\-namespace AWS/DynamoDB \
    -\-metric-name ConsumedReadCapacityUnits \
    -\-dimensions Name=TableName,Value=myTable \
    -\-statistic Sum \
    -\-threshold 240 \
    -\-comparison-operator GreaterThanOrEqualToThreshold \
    -\-period 60 \
    -\-evaluation-periods 1 \
    -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

Teste o alarme.

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value OK
```

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value ALARM
```

Mais exemplos de AWS CLI

O procedimento a seguir descreve como você receberá notificações se tiver solicitações que excedem as cotas de throughput provisionado de uma tabela.

1. Crie um tópico do Amazon SNS `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput`. Para obter mais informações, consulte Configurar o Amazon Simple Notification Service.
2. Crie o alarme.

   ```
   aws cloudwatch put-metric-alarm \
           -\-alarm-name ReadCapacityUnitsLimitAlarm \
           -\-alarm-description "Alarm when read capacity reaches 80% of my
    provisioned read capacity" \
           -\-namespace AWS/DynamoDB \
           -\-metric-name ConsumedReadCapacityUnits \
           -\-dimensions Name=TableName,Value=myTable \
           -\-statistic Sum \
           -\-threshold 240 \
           -\-comparison-operator GreaterThanOrEqualToThreshold \
           -\-period 60 \
           -\-evaluation-periods 1 \
           -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
   ```
3. Teste o alarme.

   ```
   aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
   state-reason "initializing" --state-value OK
   ```

   ```
   aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
   state-reason "initializing" --state-value ALARM
   ```

O procedimento a seguir descreve como receber notificação se houver erros no sistema.

1. Crie um tópico do Amazon SNS `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors`. Para obter mais informações, consulte Configurar o Amazon Simple Notification Service.
2. Crie o alarme.

```
aws cloudwatch put-metric-alarm \
    --alarm-name SystemErrorsAlarm \
    --alarm-description "Alarm when system errors occur" \
    --namespace AWS/DynamoDB \
    --metric-name SystemErrors \
    --dimensions Name=TableName,Value=myTable
 Name=Operation,Value=aDynamoDBOperation \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --treat-missing-data breaching \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Teste o alarme.

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value ALARM
```

Registrar em log as operações do DynamoDB usando o AWS CloudTrail

O DynamoDB é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações tomadas por um usuário, uma função ou um serviço da AWS no DynamoDB. O CloudTrail captura todas as chamadas de API para o DynamoDB na forma de eventos. As chamadas capturadas incluem chamadas do console do DynamoDB e chamadas de código para as operações da API do DynamoDB usando tanto PartiQL quanto a API clássica. Se você criar uma trilha, poderá habilitar a entrega contínua de eventos do CloudTrail para um bucket do Amazon S3, incluindo eventos para

o DynamoDB. Se você não configurar uma trilha, ainda poderá visualizar os eventos mais recentes no console do CloudTrail em Event history (Histórico de eventos). Usando as informações coletadas pelo CloudTrail, é possível determinar a solicitação feita para o DynamoDB, o endereço IP no qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita, além de detalhes adicionais.

Para monitoramento e alertas robustos, você também pode integrar eventos do CloudTrail ao Amazon CloudWatch Logs. Para aprimorar sua análise da atividade de serviço do DynamoDB e identificar alterações nas atividades de uma conta da AWS, você pode consultar os logs do AWS CloudTrail usando o Amazon Athena. Por exemplo, é possível usar consultas para identificar tendências e isolar ainda mais a atividade por atributos, como endereço IP de origem ou usuário.

Para saber mais sobre o CloudTrail, incluindo como configurá-lo e ativá-lo, consulte o Guia do usuário do AWS CloudTrail.

Tópicos

- Informações do DynamoDB no CloudTrail
- Noções básicas das entradas dos arquivos de log do DynamoDB

Informações do DynamoDB no CloudTrail

O CloudTrail é habilitado em sua conta AWS ao criá-la. Quando a atividade do evento compatível ocorrer no DynamoDB, ela será registrada em um evento do CloudTrail juntamente com outros eventos de serviços da AWS no Event history (Histórico de eventos). É possível visualizar, pesquisar e baixar eventos recentes em sua AWS conta. Para ter mais informações, consulte Working with CloudTrail Event history.

Para obter um registro contínuo de eventos na sua conta da AWS, incluindo eventos do DynamoDB, crie uma trilha. Uma trilha permite que o CloudTrail entregue arquivos de log a um bucket do Amazon S3. Por padrão, quando uma trilha é criada no console, a mesma é aplicada a todas as regiões da AWS. A trilha registra logs de eventos de todas as Regiões na AWS divisória e entrega os arquivos do log para o bucket Amazon S3 especificado. Além disso, é possível configurar outros AWS serviços para melhor analisar e agir de acordo com dados coletados do evento nos logs CloudTrail. Para obter mais informações, consulte:

- Visão geral da criação de uma trilha
- Serviços e integrações compatíveis com o CloudTrail
- Configurar notificações do Amazon SNS para o CloudTrail

- Como receber arquivos de log do CloudTrail de várias regiões e Como receber arquivos de log do CloudTrail de várias contas

Eventos do ambiente de gerenciamento no CloudTrail

As seguintes ações da API são registradas por padrão como eventos nos arquivos do CloudTrail:

Amazon DynamoDB

- CreateBackup
- CreateGlobalTable
- CreateTable
- DeleteBackup
- DeleteTable
- DescribeBackup
- DescribeContinuousBackups
- DescribeGlobalTable
- DescribeLimits
- DescribeTable
- DescribeTimeToLive
- ListBackups
- ListTables
- ListTagsOfResource
- ListGlobalTables
- RestoreTableFromBackup
- RestoreTableToPointInTime
- TagResource
- UntagResource
- UpdateGlobalTable
- UpdateTable
- UpdateTimeToLive
- DescribeReservedCapacity

- DescribeReservedCapacityOfferings
- PurchaseReservedCapacityOfferings
- DescribeScalableTargets
- RegisterScalableTarget

DynamoDB Streams

- DescribeStream
- ListStreams

DynamoDB Accelerator (DAX)

- CreateCluster
- CreateParameterGroup
- CreateSubnetGroup
- DecreaseReplicationFactor
- DeleteCluster
- DeleteParameterGroup
- DeleteSubnetGroup
- DescribeClusters
- DescribeDefaultParameters
- DescribeEvents
- DescribeParameterGroups
- DescribeParameters
- DescribeSubnetGroups
- IncreaseReplicationFactor
- ListTags
- RebootNode
- TagResource
- UntagResource
- UpdateCluster

- UpdateParameterGroup
- UpdateSubnetGroup

Eventos do plano de dados do DynamoDB no CloudTrail

Para habilitar o registro das ações de API a seguir nos arquivos do CloudTrail, você precisará ativar o registro de atividades da API do plano de dados no CloudTrail. Para obter mais informações, consulte Log de eventos de dados para trilhas.

Os eventos do plano de dados podem ser filtrados por tipo de recurso para permitir um controle detalhado sobre quais chamadas de API do DynamoDB você deseja registrar seletivamente em log e pagar no CloudTrail. Por exemplo, ao especificar `AWS::DynamoDB::Stream` como um tipo de recurso, você pode registrar em log somente chamadas para as APIs do DynamoDB. Para tabelas com fluxos habilitados, o campo de recurso no evento do plano de dados contém `AWS::DynamoDB::Stream` e `AWS::DynamoDB::Table`. Se você especificar `AWS::DynamoDB::Table` como tipo de recurso, ele registrará os eventos da tabela do DynamoDB e dos fluxos do DynamoDB por padrão. Você pode adicionar mais um filtro para excluir os eventos de fluxo, se não quiser que os eventos de fluxo sejam registrados. Para ter mais informações, consulte DataResource na AWS CloudTrail API Reference.

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Query
- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> As ações de plano de dados com vida útil do DynamoDB não são registradas no CloudTrail

DynamoDB Streams

- GetRecords
- GetShardIterator

Noções básicas das entradas dos arquivos de log do DynamoDB

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log a um bucket do Amazon S3 especificado. Os arquivos de log CloudTrail contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte, e inclui informações sobre a ação solicitada, data e hora da ação, parâmetros da solicitação e assim por diante.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar o seguinte:

- Se a solicitação foi feita com credenciais de usuário da raiz ou do .
- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado.
- Se a solicitação foi feita por outro serviço da AWS.

> **Note**
>
> Valores de atributo não chaves serão registrados nos logs de ações do CloudTrail usando a API do PartiQL e não aparecerão em logs de ações usando a API clássica.

Para obter mais informações, consulte Elemento userIdentity do CloudTrail.

Os exemplos a seguir demonstram logs do CloudTrail desses tipos de eventos:

Amazon DynamoDB

- `UpdateTable`

- DeleteTable
- CreateCluster
- PutItem (com êxito)
- UpdateItem (com falha)
- TransactWriteItems (com êxito)
- TransactWriteItems (com TransactionCanceledException)
- ExecuteStatement
- BatchExecuteStatement

DynamoDB Streams

- GetRecords

UpdateTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
```

```
            "eventTime": "2015-05-04T02:14:52Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "provisionedThroughput": {
                    "writeCapacityUnits": 25,
                    "readCapacityUnits": 25
                }
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "attributeDefinitions": [
                        {
                            "attributeType": "S",
                            "attributeName": "Artist"
                        },
                        {
                            "attributeType": "S",
                            "attributeName": "SongTitle"
                        }
                    ],
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 10,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 10,
                        "lastIncreaseDateTime": "May 3, 2015 11:34:14 PM"
                    },
                    "creationDateTime": "May 3, 2015 11:34:14 PM",
                    "keySchema": [
                        {
                            "attributeName": "Artist",
                            "keyType": "HASH"
                        },
                        {
                            "attributeName": "SongTitle",
                            "keyType": "RANGE"
                        }
                    ],
                    "tableStatus": "UPDATING",
```

```
                    "tableSizeBytes": 0
                }
            },
            "requestID": "AALNP0J2L244N5O15PKISJ1KUFVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "eb834e01-f168-435f-92c0-c36278378b6e",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

DeleteTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2015-05-04T13:38:20Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "DeleteTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
```

```
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "tableName": "Music"
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 25,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 25
                    },
                    "tableStatus": "DELETING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

CreateCluster

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "bob"
            },
            "eventTime": "2019-12-17T23:17:34Z",
            "eventSource": "dax.amazonaws.com",
            "eventName": "CreateCluster",
```

```
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.304 Python/3.6.9
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.13.40",
            "requestParameters": {
                "sSESpecification": {
                    "enabled": true
                },
                "clusterName": "daxcluster",
                "nodeType": "dax.r4.large",
                "replicationFactor": 3,
                "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess"
            },
            "responseElements": {
                "cluster": {
                    "securityGroups": [
                        {
                            "securityGroupIdentifier": "sg-1af6e36e",
                            "status": "active"
                        }
                    ],
                    "parameterGroup": {
                        "nodeIdsToReboot": [],
                        "parameterGroupName": "default.dax1.0",
                        "parameterApplyStatus": "in-sync"
                    },
                    "clusterDiscoveryEndpoint": {
                        "port": 8111
                    },
                    "clusterArn": "arn:aws:dax:us-west-2:111122223333:cache/
daxcluster",
                    "status": "creating",
                    "subnetGroup": "default",
                    "sSEDescription": {
                        "status": "ENABLED",
                        "kMSMasterKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/764898e4-adb1-46d6-a762-e2f4225b4fc4"
                    },
                    "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess",
                    "clusterName": "daxcluster",
                    "activeNodes": 0,
                    "totalNodes": 3,
```

```
                    "preferredMaintenanceWindow": "thu:13:00-thu:14:00",
                    "nodeType": "dax.r4.large"
                }
            },
            "requestID": "585adc5f-ad05-4e27-8804-70ba1315f8fd",
            "eventID": "29158945-28da-4e32-88e1-56d1b90c1a0c",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

PutItem (com êxito)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2019-01-19T15:41:54Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "PutItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
```

```
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Scared of My Shadow"
                },
                "item": [
                    "Artist",
                    "SongTitle",
                    "AlbumTitle"
                ],
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```


`UpdateItem` (com falha)


```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
```

```
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T22:27:15Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "errorCode": "ConditionalCheckFailedException",
            "errorMessage": "The conditional request failed",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Today"
                },
                "updateExpression": "SET #Y = :y, #AT = :t",
                "expressionAttributeNames": {
                    "#Y": "Year",
                    "#AT": "AlbumTitle"
                },
                "conditionExpression": "attribute_not_exists(#Y)",
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
```

```
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

`TransactWriteItems` (com êxito)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
```

```
            "eventTime": "2020-09-03T21:48:12Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
                        "ConditionExpression": "attribute_not_exists(Rating)",
                        "ExpressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Delete",
                        "TableName": "Music",
```

```
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Yesterday"
                    },
                    "conditionExpression": "#P between :lo and :hi",
                    "expressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "ConditionCheck",
                    "TableName": "Music",
                    "Key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Now"
                    },
                    "ConditionExpression": "#P between :lo and :hi",
                    "ExpressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                }
            ],
            "returnConsumedCapacity": "TOTAL",
            "returnItemCollectionMetrics": "SIZE"
        },
        "responseElements": null,
        "requestID": "45EN32OM6TQSMV2MI65O4L5TNFVV4KQNSO5AEMVJF66Q9ASUAAJG",
        "eventID": "4f1cc78b-5c94-4174-a6ad-3ee78605381c",
        "readOnly": false,
        "resources": [
            {
                "accountId": "111122223333",
                "type": "AWS::DynamoDB::Table",
                "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
            }
        ],
        "eventType": "AwsApiCall",
        "apiVersion": "2012-08-10",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data"
    }
```

```
    ]
}
```

TransactWriteItems (com **TransactionCanceledException**)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2019-02-01T00:42:34Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.93 Python/3.4.7
 Linux/4.9.119-0.1.ac.277.71.329.metal1.x86_64 botocore/1.12.83",
            "errorCode": "TransactionCanceledException",
            "errorMessage": "Transaction cancelled, please refer cancellation reasons
 for specific reasons [ConditionalCheckFailed, None]",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
```

```
                    "tableName": "Music",
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Today"
                    },
                    "items": [
                        "Artist",
                        "SongTitle",
                        "AlbumTitle"
                    ],
                    "conditionExpression": "#AT = :A",
                    "expressionAttributeNames": {
                        "#AT": "AlbumTitle"
                    },
                    "returnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "Update",
                    "tableName": "Music",
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Tomorrow"
                    },
                    "updateExpression": "SET #AT = :newval",
                    "ConditionExpression": "attribute_not_exists(Rating)",
                    "ExpressionAttributeNames": {
                        "#AT": "AlbumTitle"
                    },
                    "returnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "Delete",
                    "TableName": "Music",
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Yesterday"
                    },
                    "conditionExpression": "#P between :lo and :hi",
                    "expressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
```

```
                        "operation": "ConditionCheck",
                        "TableName": "Music",
                        "Key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Now"
                        },
                        "ConditionExpression": "#P between :lo and :hi",
                        "ExpressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    }
                ],
                "returnConsumedCapacity": "TOTAL",
                "returnItemCollectionMetrics": "SIZE"
            },
            "responseElements": null,
            "requestID": "A0GTQEKLBB9VD8E05REA5A3E1VVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "43e437b5-908a-46af-84e6-e27fffb9c5cd",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

ExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
```

```
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:06:45Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "ExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "statement": "SELECT * FROM Music WHERE Artist = 'No One You Know' AND
SongTitle = 'Call Me Today' AND nonKeyAttr = ***(Redacted)"
            },
            "responseElements": null,
            "requestID": "V7G2KCSFLP83ORB7MMFG6RIAD3VV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "0b5c4779-e169-4227-a1de-6ed01dd18ac7",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
```

```
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

BatchExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:24:48Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "BatchExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "requestItems": [
                    {
```

```
                        "statement": "UPDATE Music SET Album = ***(Redacted) WHERE
 Artist = 'No One You Know' AND SongTitle = 'Call Me Today'"
                    },
                    {
                        "statement": "INSERT INTO Music VALUE {'Artist' :
 ***(Redacted), 'SongTitle' : ***(Redacted), 'Album' : ***(Redacted)}"
                    }
                ]
            },
            "responseElements": null,
            "requestID": "23PE7ED291UD65P9SMS6TISNVBVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "f863f966-b741-4c36-b15e-f867e829035a",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

GetRecords

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
```

```
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-04-15T04:15:02Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "GetRecords",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.50 Python/3.6.13
 Linux/4.9.230-0.1.ac.224.84.332.metal1.x86_64 botocore/1.20.50",
            "requestParameters": {
                "shardIterator": "arn:aws:dynamodb:us-west-2:123456789012:table/
Music/stream/2021-04-15T04:02:47.428|1|AAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGV1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpluWsHyCW3PPbKsMlJSruVRWoitRhSd3S6slEWEPB0bDC7+
+ISH5mXrCHOnvyezQKlqNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bpOUI2cuRTchgUpPSe3ur2sQrRj3KlbmIyCz7P
+H3CYlugafi8fQ5kipDSkESkIWS6O5ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAk3ab68aUXGSFEZLONntgNIhQkcMo00/
mJlaIgkEdBUyqvZO1vtKUBH5YonIrZqSUhv8Coc+mh24vOg1YI+SPIXlr
+Lnl54BG6AjrmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cskcX2ISaLHwYzJPaMBSftxOgeRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXmOgay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMhlBVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrzOEzeB4uconEXAMPLE=="
            },
            "responseElements": null,
            "requestID": "1M0U1Q80P4LDPT7A7N1A758N2VVV4KQNSO5AEMVJF66Q9EXAMPLE",
            "eventID": "09a634f2-da7d-4c9e-a259-54aceexample",
            "readOnly": true,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
```

```
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

Analisar acesso a dados usando o CloudWatch Contributor Insights para DynamoDB

O Amazon CloudWatch Contributor Insights para Amazon DynamoDB é uma ferramenta de diagnóstico para identificar rapidamente as chaves acessadas com maior frequência e mais limitadas em sua tabela ou em um índice. Essa ferramenta usa o CloudWatch Contributor Insights.

Ao habilitar o CloudWatch Contributor Insights para DynamoDB em uma tabela ou em um índice secundário global, é possível visualizar os itens mais acessados e mais limitados nesses recursos.

> **Note**
>
> As cobranças do CloudWatch aplicam-se ao Contributor Insights para DynamoDB. Para obter mais informações sobre a definição de preço, consulte Definição de preço do Amazon CloudWatch.

Tópicos

- CloudWatch Contributor Insights para DynamoDB: como funciona
- Conceitos básicos do CloudWatch Contributor Insights para DynamoDB
- Usar o IAM com o CloudWatch Contributor Insights para DynamoDB

CloudWatch Contributor Insights para DynamoDB: como funciona

O Amazon DynamoDB integra-se ao CloudWatch Contributor Insights para fornecer informações sobre os itens mais acessados e com maior controle de utilização em uma tabela ou em um índice

secundário global. O DynamoDB fornece essas informações a você por meio de regras, relatórios e grafos de dados de relatório do CloudWatch Contributor Insights.

O CloudWatch Contributor Insights para DynamoDB foi projetado para não ter impacto na performance da sua tabela do DynamoDB.

Para ter mais informações sobre o CloudWatch Contributor Insights, consulte Usar o Contributor Insights para analisar dados de alta cardinalidade no Guia do usuário do Amazon CloudWatch.

As seções a seguir descrevem os principais conceitos e o comportamento do CloudWatch Contributor Insights para DynamoDB.

Tópicos

- Modos do CloudWatch Contributor Insights para DynamoDB
- Regras do CloudWatch Contributor Insights para DynamoDB
- Entender o CloudWatch Contributor Insights para gráficos do DynamoDB
- Interações com outros recursos do DynamoDB
- Faturamento do CloudWatch Contributor Insights para DynamoDB

Modos do CloudWatch Contributor Insights para DynamoDB

O CloudWatch Contributor Insights para DynamoDB oferece dois modos distintos para atender às diferentes necessidades de monitoramento.

modo de chaves limitadas

Esse modo se concentra exclusivamente em solicitações com controle de utilização, processando eventos somente quando ocorre controle. Ele oferece informações sobre problemas de desempenho sem os custos indiretos de rastrear todos os padrões de acesso. Nesse modo, o DynamoDB rastreia somente:

- Itens mais limitados: aqueles que experimentam mais eventos de controle de utilização.

Esse modo é ideal quando:

- Sua principal preocupação é identificar e resolver problemas de controle de utilização.
- Você deseja manter o Contributor Insights habilitado continuamente para detecção de controle de utilização em tempo real.

- Você quer uma abordagem otimizada em termos de custo para monitorar problemas de controle de utilização.

> **Note**
>
> O modo de chaves limitadas processa eventos somente quando ocorre controle de utilização, o que o torna econômico para o monitoramento contínuo. Essa abordagem direcionada permite que você deixe o recurso habilitado permanentemente com um impacto mínimo nos custos, ao mesmo tempo em que oferece visibilidade imediata sobre os problemas de controle de utilização à medida que eles acontecem.

Se sua tabela não sofrer controle de utilização, você não verá nenhum dado nos grafos do Contributor Insights, o que indica um desempenho íntegro. Quando há detecção de controle de utilização, os grafos gerados ajudam a identificar padrões de acesso específicos que causam problemas de desempenho. Essas informações podem ajudar você a implementar estratégias para lidar com padrões de acesso não uniformes.

Para ter estratégias de monitoramento abrangentes, você pode integrar esses insights de controle de utilização a outras métricas do CloudWatch para criar painéis unificados que correlacionam os eventos de controle de utilização com o desempenho geral da tabela.

Modo de chaves acessadas e limitadas

Esse modo oferece monitoramento abrangente de itens acessados e limitados. Nesse modo, o DynamoDB rastreia:

- Itens mais acessados: aqueles que consomem a maior capacidade de leitura e gravação.
- Itens mais limitados: aqueles que experimentam mais eventos de controle de utilização.

Esse modo é ideal quando você precisa de visibilidade completa dos padrões de acesso da sua tabela e quer entender tanto os itens de alto tráfego quanto os problemas de controle de utilização.

Alternar entre os modos

Você pode alternar entre os modos a qualquer momento usando o console, a AWS CLI ou as APIs do DynamoDB. Quando você alterna os modos:

- As regras existentes do CloudWatch são atualizadas para corresponder ao novo modo.

- As regras de chaves limitadas do CloudWatch permanecem intactas, mantendo seus dados históricos contínuos para métricas de controle de utilização:
  - Quando você alterna do modo de chaves limitadas para o modo de chaves acessadas e limitadas, as regras de chaves limitadas existentes são preservadas e novas regras de chaves acessadas são criadas.
  - Quando você alterna do modo de chaves acessadas e limitadas para o modo de chaves limitadas, as regras de chaves limitadas existentes são preservadas e as regras de chaves acessadas são removidas.
- A cobrança é ajustada imediatamente para exibir o processamento de eventos do novo modo

Regras do CloudWatch Contributor Insights para DynamoDB

Ao habilitar o CloudWatch Contributor Insights para o DynamoDB em uma tabela ou em um índice secundário global, o DynamoDB cria regras em seu nome com base no modo selecionado.

> **Note**
>
> Ao habilitar o Contributor Insights na sua tabela do DynamoDB, observe que os limites das regras do Contributor Insights continuam aplicáveis. Para obter mais informações, consulte Cotas de serviço do CloudWatch.

Regras para o modo de chaves acessadas e limitadas

No modo de chaves acessadas e limitadas, o DynamoDB cria as seguintes regras:

- Itens mais acessados (chave de partição): identifica as chaves de partição dos itens mais acessados na tabela ou no índice secundário global.

  Formato do nome da regra do CloudWatch: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`
- Chaves mais limitadas (chave de partição): identifica as chaves de partição dos itens mais limitados na tabela ou no índice secundário global.

  Formato do nome da regra do CloudWatch: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Se a tabela ou o índice secundário global tiver uma chave de classificação, o DynamoDB também criará as seguintes regras específicas para chaves de classificação:

- Chaves mais acessadas (chaves de partição e de classificação): identifica as chaves de partição e de classificação dos itens mais acessados na tabela ou no índice secundário global.

  Formato do nome da regra do CloudWatch: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`
- Chaves mais limitadas (chaves de partição e de classificação): identifica as chaves de partição e de classificação dos itens mais limitadas na tabela ou no índice secundário global.

  Formato do nome da regra do CloudWatch: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Regras para o modo de chaves limitadas

No modo de chaves limitadas, o DynamoDB cria somente regras relacionadas ao controle de utilização:

- Chaves mais limitadas (chave de partição): identifica as chaves de partição dos itens mais limitados na tabela ou no índice secundário global.

  Formato do nome da regra do CloudWatch: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Se a tabela ou o índice secundário global tiver uma chave de classificação, o DynamoDB também criará:

- Chaves mais limitadas (chaves de partição e de classificação): identifica as chaves de partição e de classificação dos itens mais limitadas na tabela ou no índice secundário global.

  Formato do nome da regra do CloudWatch: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Essa abordagem direcionada reduz o número de regras ativas e diminui o volume de eventos processados para diagnosticar melhor os eventos de controle de utilização.

> **Note**
>
> - Ao usar o console ou as APIs do CloudWatch para visualizar o CloudWatch Contributor Insights para DynamoDB, só é possível ver as regras correspondentes ao modo selecionado.
> - Não é possível usar as APIs ou o console do CloudWatch para excluir ou modificar diretamente as regras criadas pelo CloudWatch Contributor Insights para DynamoDB. Desabilitar o CloudWatch Contributor Insights para DynamoDB em uma tabela ou em um índice secundário global exclui automaticamente as regras criadas para essa tabela ou para esse índice secundário global.
> - Quando a operação GetInsightRuleReport é usada com regras do CloudWatch Contributor Insights criadas pelo DynamoDB, somente `MaxContributorValue` e `Maximum` retornam estatísticas úteis. As outras estatísticas dessa lista não retornam valores significativos.
> - O CloudWatch Contributor Insights para DynamoDB tem um limite de 25 colaboradores. Solicitar mais de 25 colaboradores retornará um erro.

Você pode criar alarmes do CloudWatch usando as regras do CloudWatch Contributor Insights para DynamoDB. Isso permite que você seja notificado quando um item exceder ou atingir um limite específico para `ConsumedThroughputUnits` ou `ThrottleCount`. Para ter mais informações, consulte Setting an alarm on Contributor Insights metric data.

Entender o CloudWatch Contributor Insights para gráficos do DynamoDB

Dependendo do modo selecionado, o CloudWatch Contributor Insights para DynamoDB exibe diferentes tipos de grafos no console do DynamoDB e no console do CloudWatch.

Disponibilidade de grafos por modo

Os grafos exibidos dependem do modo selecionado do Contributor Insights.

- O modo de chaves acessadas e limitadas exibe os grafos de itens mais acessados e de itens mais limitados.
- O modo de chaves aceleradas exibe somente os grafos de itens mais limitados.

Itens mais acessados

Esse grafo está disponível somente no modo de chaves acessadas e limitadas. Use esse gráfico para identificar os itens mais acessados na tabela ou no índice secundário global. O gráfico exibe `ConsumedThroughputUnits` no eixo y e o tempo no eixo x. Cada uma das principais chaves N são exibidas em sua própria cor com uma legenda exibida abaixo do eixo x.

O DynamoDB mede a frequência de acesso a chaves usando `ConsumedThroughputUnits`, que mede o tráfego de leitura e de gravação combinados. `ConsumedThroughputUnits` é definido como o seguinte:

- Provisionado: (3 x unidades de capacidade de gravação consumidas) + unidades de capacidade de leitura consumidas
- Sob demanda: (3 x unidades de solicitação de gravação) + unidades de solicitação de leitura

No console do DynamoDB, cada ponto de dados no gráfico representa o máximo de `ConsumedThroughputUnits` em um período de 1 minuto. Por exemplo, um valor do gráfico de 180.000 `ConsumedThroughputUnits` indica que o item foi acessado continuamente ao throughput máximo por item de 1.000 unidades de solicitação de gravação ou 3.000 unidades de solicitação de leitura durante um intervalo de 60 segundos dentro desse período de 1 minuto (3.000 x 60 segundos). Em outras palavras, os valores no gráfico representam o minuto com tráfego mais alto dentro de cada período de um minuto. Você pode alterar a granularidade de tempo da métrica `ConsumedThroughputUnits` (por exemplo, para exibir métricas de 5 minutos em vez de um minuto) no console do CloudWatch.

Se forem exibidas várias linhas estreitamente agrupadas sem pontos fora da curva óbvios, isso indicará que a workload está relativamente equilibrada entre os itens na janela de tempo especificada. Se forem exibidos pontos isolados no gráfico em vez de linhas conectadas, isso indicará um item que foi frequentemente acessado somente por um breve período.

Se a tabela ou o índice secundário global tiver uma chave de classificação, o DynamoDB criará dois gráficos: um para as chaves de partição mais acessadas e outro para os pares de chave de classificação + partição mais acessados. É possível ver o tráfego no nível de chave de partição no gráfico apenas de chave de partição. Você pode ver o tráfego no nível do item nos gráficos de chave de partição + classificação.

Itens com maior controle de utilização

Esse grafo está disponível nos dois modos. Use esse gráfico para identificar os itens mais limitados na tabela ou no índice secundário global. O gráfico exibe `ThrottleCount` no eixo y e o tempo no eixo x. Cada uma das principais chaves N são exibidas em sua própria cor com uma legenda exibida abaixo do eixo x.

O DynamoDB mede a frequência de limitação usando `ThrottleCount`, que é a contagem dos erros `ProvisionedThroughputExceededException`, `ThrottlingException` e `RequestLimitExceeded`.

A controle de utilização de gravação causado por capacidade de gravação insuficiente para um índice secundário global não é medida. Você pode usar o gráfico Itens mais acessados do índice secundário global para identificar padrões de acesso desequilibrados que podem causar controle de utilização de gravação. Para ter mais informações, consulte Considerações sobre throughput provisionado para índices secundários globais.

No console do DynamoDB, cada ponto de dados no gráfico representa a contagem de eventos de limitação em um período de 1 minuto.

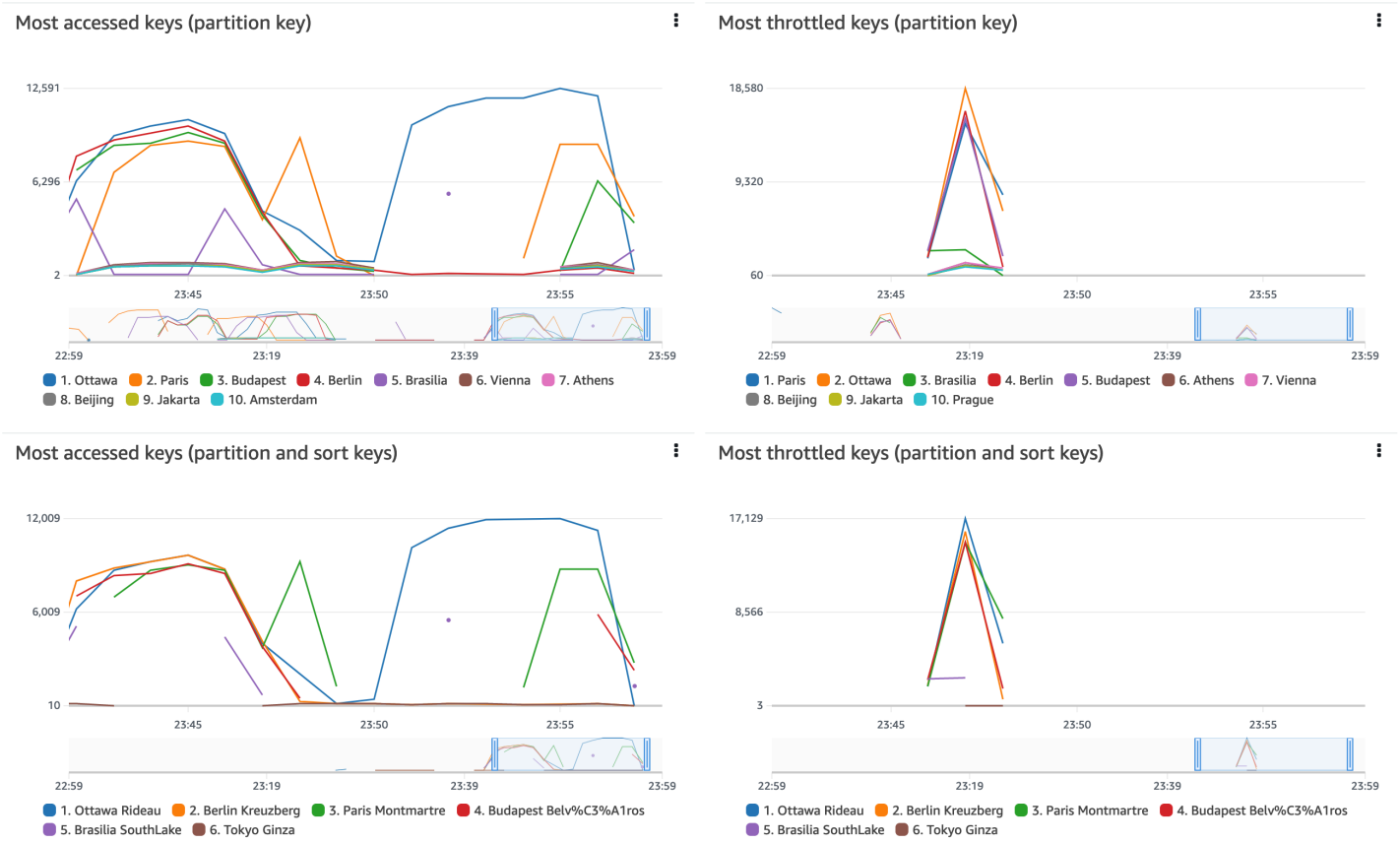
Se não forem exibidos dados neste gráfico, isso indicará que as solicitações não estão sendo limitadas. Se você vir pontos isolados no gráfico em vez de linhas conectadas, isso indica que um item foi frequentemente limitado por um breve período.

Se a tabela ou o índice secundário global tiver uma chave de classificação, o DynamoDB criará dois gráficos: um para as chaves de partição mais limitadas e outro para os pares de chave de classificação + partição mais limitados. É possível ver a contagem de limitação no nível da chave de partição no gráfico somente da chave de partição, e a contagem de limitação no nível do item nos gráficos de chave de classificação + partição.

> **Note**
>
> No modo de chaves limitadas, esse é o único tipo de grafo que você verá. A ausência de dados nesses grafos indica um desempenho íntegro da tabela, sem a ocorrência de controle de utilização.

Exemplos de relatório

O exemplo a seguir mostra os relatórios gerados para uma tabela com uma chave de partição e uma chave de classificação nos modos de chaves acessadas e limitadas. No modo de chaves limitadas, é possível ver somente a parte relacionada ao controle de utilização desse relatório.



Interações com outros recursos do DynamoDB

As seções a seguir descrevem como o CloudWatch Contributor Insights para DynamoDB se comporta e interage com vários outros recursos no DynamoDB. Esses comportamentos se aplicam a ambos os modos, a menos que seja especificado de outra forma.

Tabelas globais

O CloudWatch Contributor Insights para DynamoDB monitora réplicas de tabelas globais como tabelas distintas. Os gráficos do Contributor Insights para uma réplica em uma região da AWS podem não mostrar os mesmos padrões que outra região. Isso ocorre porque os dados de gravação são replicados em todas as réplicas em uma tabela global, mas cada réplica pode atender ao tráfego de leitura limitado à região.

Cada réplica pode ser configurada com um modo diferente do Contributor Insights de forma independente. Por exemplo, você pode usar o modo de chaves acessadas e limitadas em sua região principal para ter um monitoramento abrangente e usa o modo de chaves limitadas em regiões secundárias para manter a visibilidade sobre problemas de desempenho.

DynamoDB Accelerator (DAX)

O CloudWatch Contributor Insights para DynamoDB não mostra respostas em cache do DAX. Ele mostra apenas respostas para acessar uma tabela ou um índice secundário global.

> **Note**
>
> O CloudWatch Contributor Insights para DynamoDB não comporta solicitações PartiQL.

Criptografia em repouso

O CloudWatch Contributor Insights para DynamoDB não afeta a maneira como a criptografia funciona no DynamoDB. Os dados de chave primária publicados no CloudWatch são criptografados com a Chave pertencente à AWS. Porém, o DynamoDB também é compatível com Chave gerenciada pela AWS e uma chave gerenciada pelo cliente.

O CloudWatch Contributor Insights para DynamoDB exibe as chaves de partição e as chaves de classificação (se aplicável) de itens acessados com frequência e itens com controle de utilização. Embora o CloudWatch Contributor Insights funcione com tabelas criptografadas do DynamoDB, é importante observar que ele usa seu próprio contexto de criptografia de propriedade da Amazon, que é separado da criptografia configurada da tabela.

Se a chave primária da sua tabela do DynamoDB contiver informações confidenciais e as políticas de segurança da sua organização exigirem controle total sobre os processos de criptografia, habilitar o CloudWatch Contributor Insights pode não ser adequado.

Controle de acesso refinado

O CloudWatch Contributor Insights para DynamoDB não funciona de forma diferente para tabelas com controle de acesso refinado (FGAC). Em outras palavras, qualquer usuário com as permissões apropriadas do CloudWatch poderá visualizar chaves primárias protegidas por FGAC nos gráficos do CloudWatch Contributor Insights.

Se a chave primária da tabela tiver dados protegidos por um controle de acesso refinado (FGAC) que você não deseja que sejam publicados no CloudWatch, não habilite o CloudWatch Contributor Insights para DynamoDB para essa tabela.

Controle de acesso

O controle de acesso ao CloudWatch Contributor Insights para DynamoDB é feito com o AWS Identity and Access Management (IAM) por meio da limitação das permissões do ambiente de gerenciamento do DynamoDB e das permissões do plano de dados do CloudWatch. Para obter mais informações, consulte Usar o IAM com o CloudWatch Contributor Insights para DynamoDB.

Faturamento do CloudWatch Contributor Insights para DynamoDB

As cobranças do CloudWatch Contributor Insights para DynamoDB aparecem na seção CloudWatch da sua fatura mensal. Essas cobranças são calculadas com base no número de eventos do DynamoDB que são processados e no modo selecionado.

Cobrança de acordo com o modo

Os dois modos do Contributor Insights têm características de cobrança diferentes.

- Cobrança no modo de chaves acessadas e limitadas: nesse modo, cada item gravado ou lido por meio de uma operação de plano de dados representa um evento, independentemente de a solicitação ter êxito ou sofrer controle de utilização. Se uma tabela ou um índice secundário global incluir uma chave de classificação, cada item que for lido ou gravado representará dois eventos. Isso ocorre porque o DynamoDB está identificando os principais colaboradores de séries temporais separados: um para chaves de partições somente e outro para pares de chaves de partição e de classificação.
- Cobrança no modo de chaves limitadas: nesse modo, somente solicitações com controle de utilização geram eventos faturáveis. Os eventos são gerados somente quando as solicitações resultam em um erro `ProvisionedThroughputExceededException`, `ThrottlingException` ou `RequestLimitExceeded`. Se uma tabela ou índice secundário global incluir uma chave de classificação, cada item com controle de utilização representará dois eventos (rastreamento da chave de partição e rastreamento da chave de partição + classificação).

Exemplos de faturamento

Por exemplo, vamos supor que sua aplicação execute as seguintes operações do DynamoDB: `GetItem`, `PutItem` e uma `BatchWriteItem` que insere cinco itens. Suponha também que a operação `PutItem` sofra controle de utilização e todas as outras operações sejam bem-sucedidas.

- Modo de chaves acessadas e limitadas
  - Se a tabela ou o índice secundário global tiver apenas uma chave de partição, isso resultará em 7 eventos (1 para `GetItem`, 1 para `PutItem` e 5 para `BatchWriteItem`).
  - Se a tabela ou o índice secundário global tiver uma chave de partição e uma chave de classificação, isso resultará em 14 eventos (2 para `GetItem`, 2 para `PutItem` e 10 para `BatchWriteItem`).
- Modo de chaves limitadas
  - Se a tabela ou o índice secundário global tiver apenas uma chave de partição, isso resultará em um único evento (somente para a operação `PutItem` com controle de utilização).
  - Se a tabela ou o índice secundário global tiver apenas uma chave de classificação, isso resultará em dois eventos (dois para a operação `PutItem` com controle de utilização).

  As operações `GetItem` e `BatchWriteItem` bem-sucedidas não geram eventos no modo de chaves limitadas.

Fatores comuns de cobrança

Uma operação `Query` sempre resulta em um evento, independentemente do número de itens exibidos.

Ao contrário de outros recursos do DynamoDB, o CloudWatch Contributor Insights para faturamento do DynamoDB não varia de acordo com o seguinte:

- O modo de capacidade (provisionado versus sob demanda)
- Se você executa solicitações de leitura ou gravação
- O tamanho (KB) dos itens lidos ou gravados

Conceitos básicos do CloudWatch Contributor Insights para DynamoDB

Esta seção descreve como habilitar e usar o Amazon CloudWatch Contributor Insights em diferentes modos por meio do console do Amazon DynamoDB ou da AWS Command Line Interface (AWS CLI) para atender às suas necessidades de monitoramento.

Nos exemplos a seguir, você usará a tabela do DynamoDB definida no tutorial Conceitos básicos do DynamoDB.

Tópicos

- Escolher um modo do Contributor Insights
- Usar o Contributor Insights (console)
- Uso do Contributor Insights (AWS CLI)

Escolher um modo do Contributor Insights

Antes de habilitar o Contributor Insights, você deve entender como funcionam os dois modos disponíveis. Analise a comparação de modos para selecionar a opção mais adequada aos seus requisitos específicos.

Aspecto	Modo de chaves acessadas e limitadas	modo de chaves limitadas
Monitores	Todas as solicitações (bem-sucedidas e limitadas)	Somente solicitações limitadas
Gráficos	Itens mais acessados + itens com utilização mais limitada	Somente itens com utilização mais limitada
Melhor para	Análise e otimização direcionadas	Monitoramento do controle de utilização
Use quando	Precisar de visibilidade completa dos padrões de acesso. Estiver realizando análises ou depurações de curto prazo.	Sua principal preocupação for identificar e resolver problemas de controle de utilização. Desejar manter o Contributor Insights habilitado continuamente para

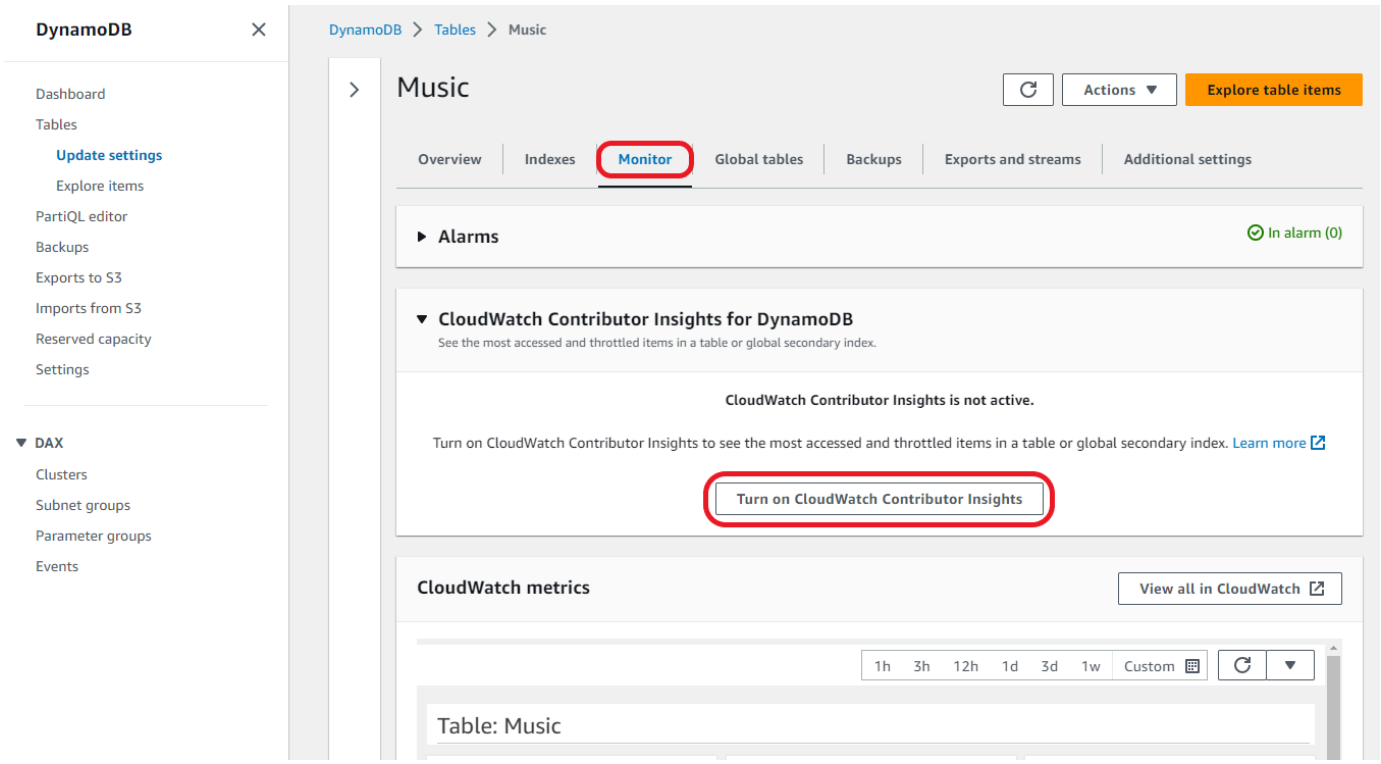
Aspecto	Modo de chaves acessadas e limitadas	modo de chaves limitadas
		detecção de controle de utilização em tempo real.

Usar o Contributor Insights (console)

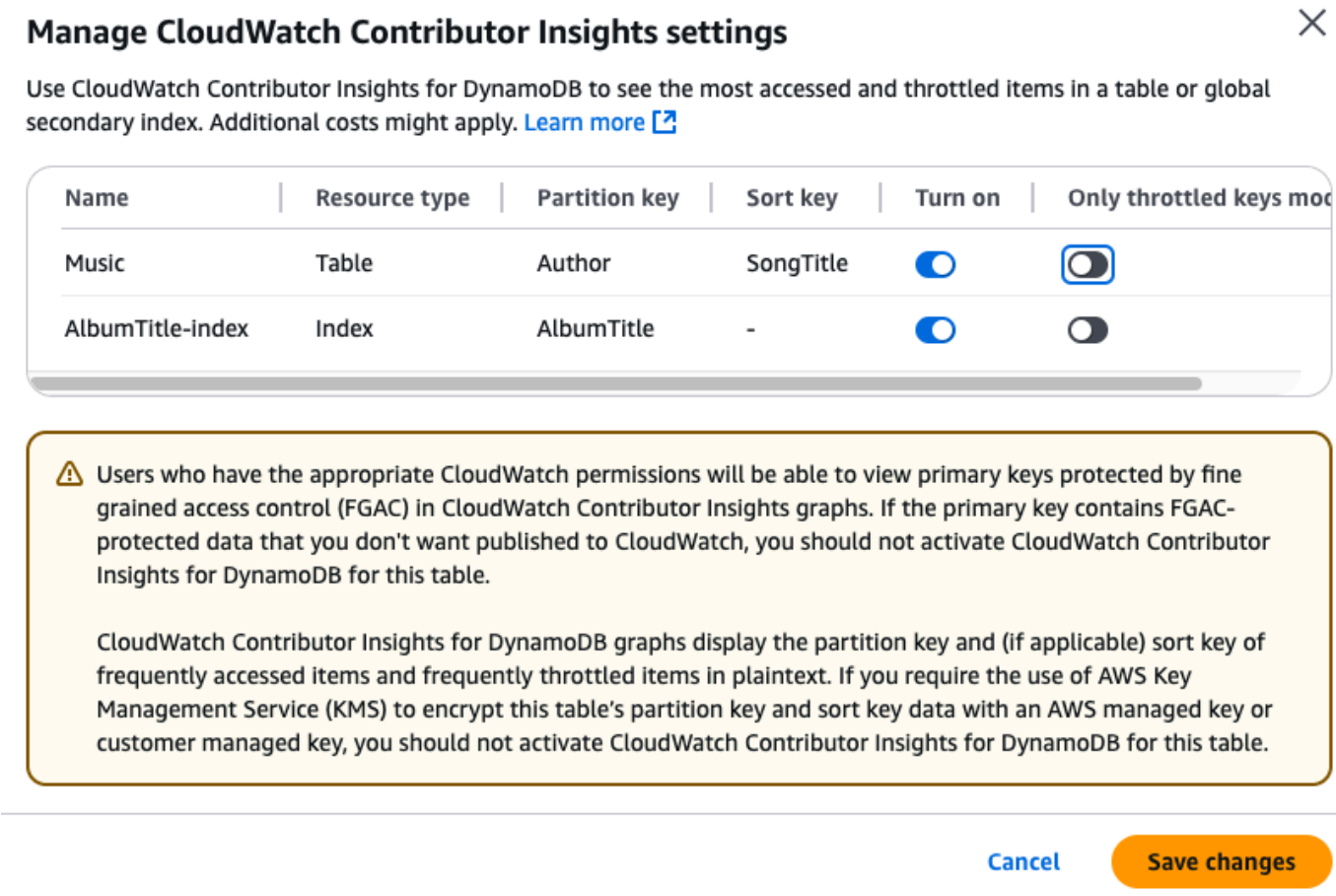
O console oferece uma maneira intuitiva de habilitar o Contributor Insights e selecionar o modo apropriado para suas necessidades de monitoramento.

Como usar o Contributor Insights no console

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Escolha a tabela `Music`.
4. Escolha a guia Monitor (Monitorar).
5. Escolha Gerenciar o CloudWatch Contributor Insights.

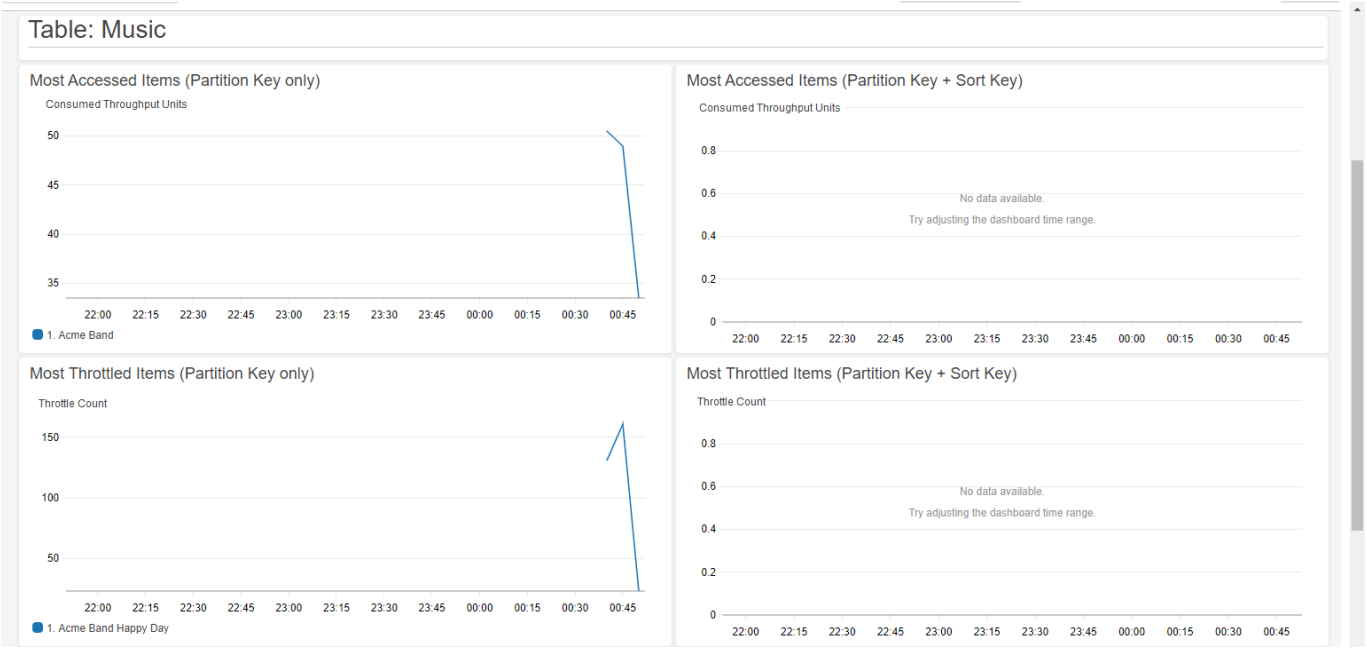


6. Na caixa de diálogo Gerenciar configurações do CloudWatch Contributor Insights, marque Ativar para a tabela base `Music` e o índice secundário global `AlbumTitle-index`.
7. Deixe a opção Apenas modo de chaves limitadas na posição desativada para ambos e escolha Salvar alterações.



   Isso habilita o modo padrão de chaves acessadas e limitadas para a tabela e o GSI, que oferece o monitoramento de itens acessados e limitados. Alternar Apenas modo de chaves limitadas para a posição ativada habilita o modo de chaves limitadas.

   Se a operação falhar, consulte DescribeContributorInsights FailureException na Referência da API do Amazon DynamoDB em busca de possíveis razões.
8. Os grafos do CloudWatch Contributor Insights agora estão visíveis na guia Monitor para a tabela `Music`. Como você habilitou o modo de chaves acessadas e limitadas, é possível ver grafos de itens acessados e limitados.



Alternar entre os modos

É possível alternar entre os modos a qualquer momento sem desabilitar o Contributor Insights.

Como alternar entre os modos do Contributor Insights

1. Na guia Monitor da tabela, escolha Gerenciar o CloudWatch Contributor Insights.
2. Na caixa de diálogo Gerenciar configurações do CloudWatch Contributor Insights, para cada tabela base ou GSI:
   - Ative ou desative Apenas modo de chaves limitadas para habilitar o modo de chaves limitadas ou volte para o modo padrão de chaves acessadas e limitadas.
   - Marque ou desmarque Ativar para desabilitar o CloudWatch Contributor Insights para uma tabela ou GSI.
3. Escolha Salvar alterações.

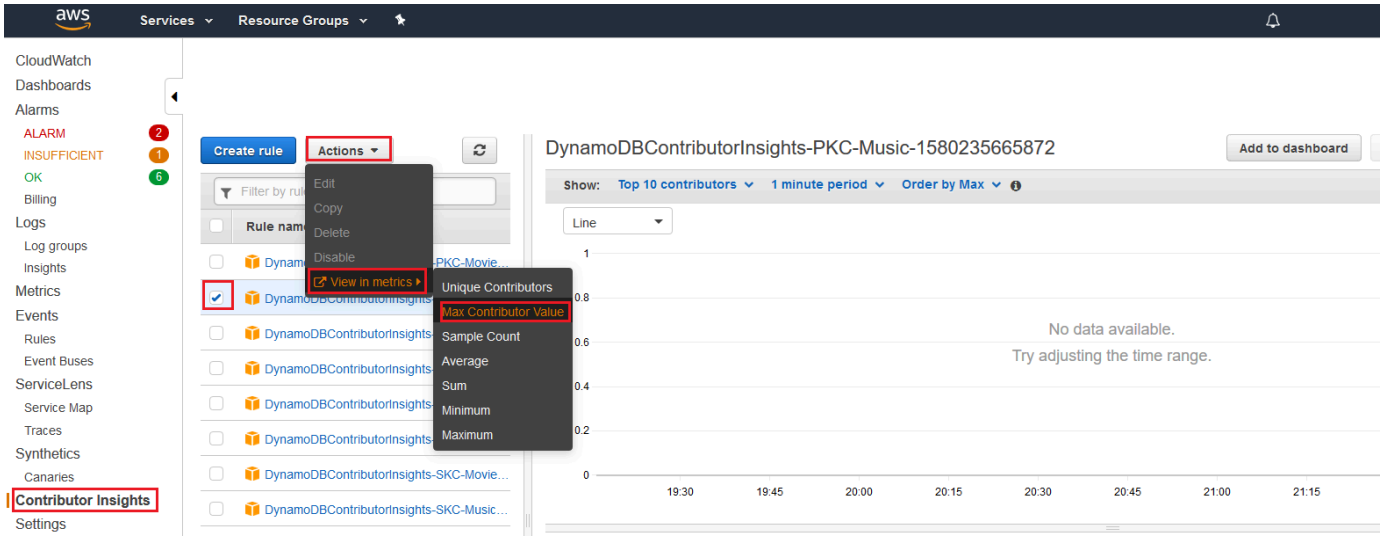
   Depois de concluído, os grafos mostrarão o novo modo.

Criar alarmes do CloudWatch

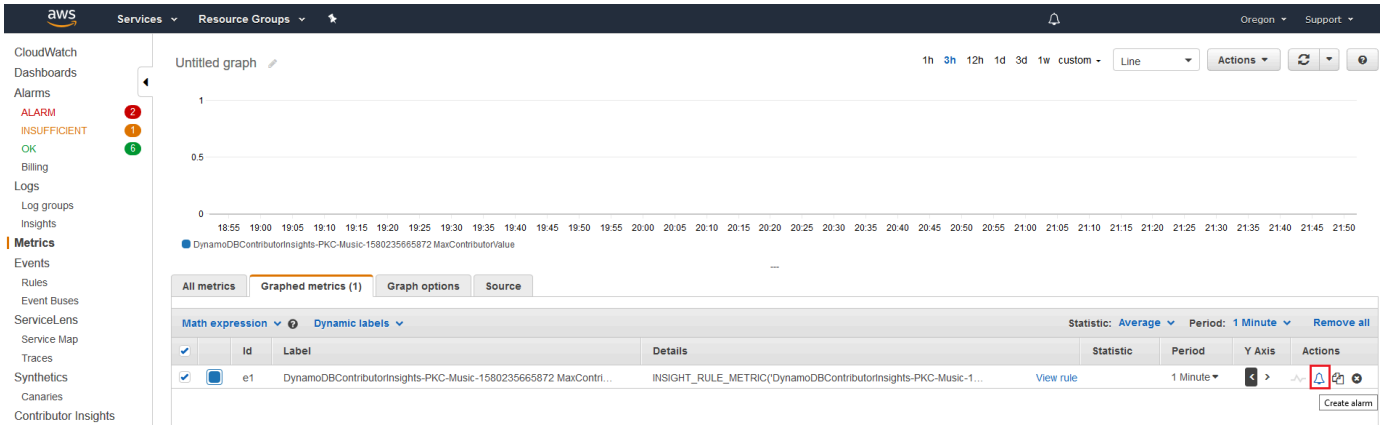
Siga as etapas abaixo para criar um alarme do CloudWatch e receber notificação quando uma chave de partição consumir mais de 50 mil ConsumedThroughputUnits ou sofrer controle de utilização.

1. Faça login no AWS Management Console e abra o console do CloudWatch em https://console.aws.amazon.com/cloudwatch/.
2. No painel de navegação à esquerda do console, escolha Contributor Insights.
3. Escolha a regra apropriada com base no modo escolhido e no que você deseja monitorar:
   - Para monitoramento de itens acessados (apenas modo de chaves acessadas e limitadas): escolha DynamoDBContributorInsights-PKC-Music.
   - Para monitoramento de itens limitados (ambos os modos): escolha DynamoDBContributorInsights-PKT-Music.
4. Selecione a lista suspensa Actions (Ações).
5. Escolha View in metrics (Exibir nas métricas).
6. Escolha Max Contributor Value (Valor máximo do colaborador).

> **Note**
>
> Somente `Max Contributor Value` e `Maximum` retornam estatísticas úteis. As outras estatísticas dessa lista não retornam valores significativos.

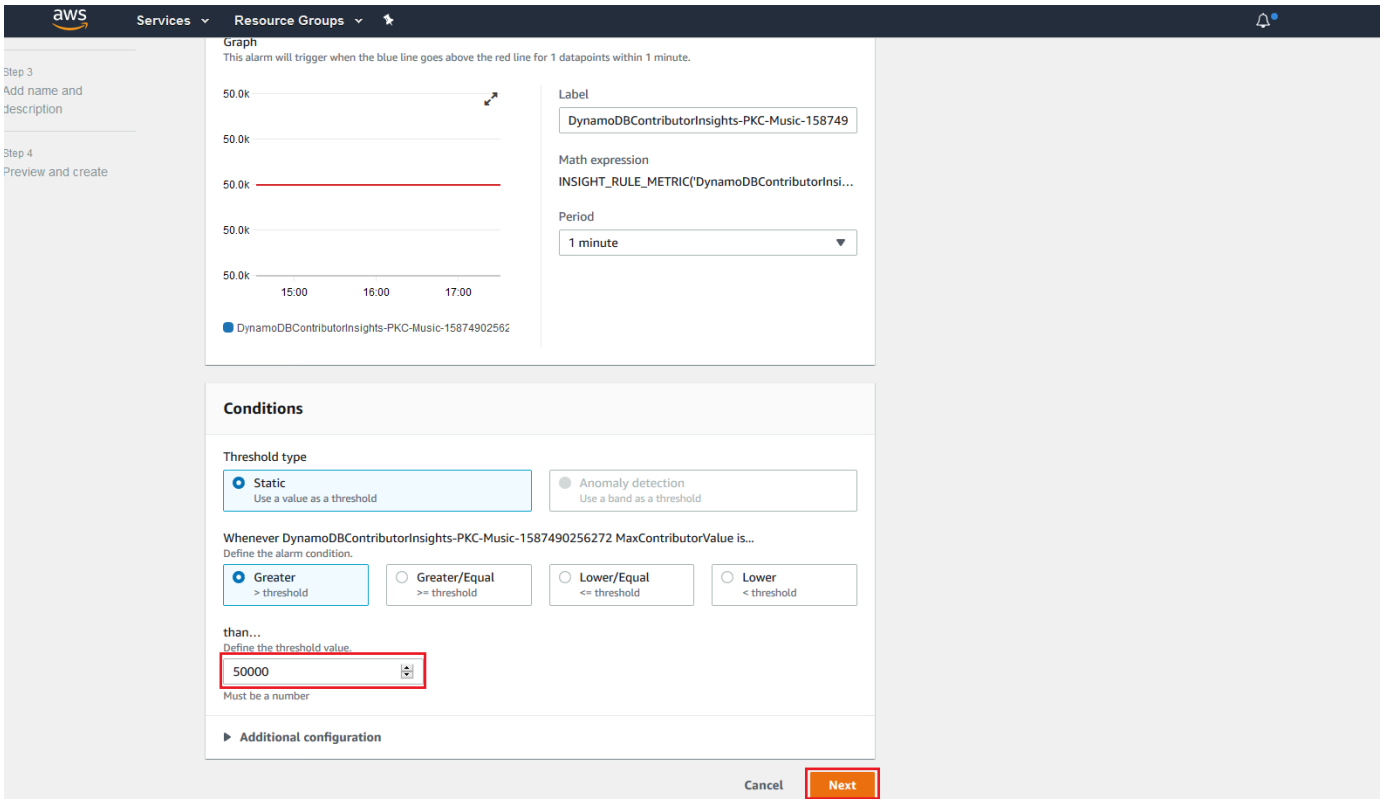


7. Na coluna Actions (Ações), escolha Create Alarm (Criar alarme).



8. Insira um valor limite apropriado e escolha Próximo:

   - Para itens acessados (regras PKC): insira 50 mil para `ConsumedThroughputUnits`.
   - Para itens limitados (regras PKT): insira 1 para `ThrottleCount` para receber alertas sobre qualquer controle de utilização.



9. Consulte Usar os alarmes do Amazon CloudWatch para obter detalhes sobre como configurar a notificação para o alarme.

Uso do Contributor Insights (AWS CLI)

A AWS CLI oferece acesso programático ao Contributor Insights com compatibilidade total para os dois modos. Você pode especificar o modo ao habilitar o Contributor Insights ou alternar de modo posteriormente.

Operações básicas com o modo padrão

Como usar o Contributor Insights com as configurações padrão

1. Habilite o CloudWatch Contributor Insights para DynamoDB na tabela base `Music` com o modo de chaves acessadas e limitadas. Como `ACCESSED_AND_THROTTLED_KEYS` é o modo padrão, é possível omitir o parâmetro `--contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --contributor-insights-action=ENABLE
   ```

2. Habilite o Contributor Insights para DynamoDB no índice secundário global `AlbumTitle-index`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --index-name AlbumTitle-index \
                       --contributor-insights-action=ENABLE
   ```

3. Obtenha o status e as regras da tabela `Music` e todos os seus índices.

   ```
   aws dynamodb describe-contributor-insights
                       --table-name Music
   ```

   A resposta incluirá o campo `ContributorInsightsMode` mostrando `ACCESSED_AND_THROTTLED_KEYS`.

4. Obtenha o status da tabela `Music` e de todos os respectivos índices.

   ```
   aws dynamodb list-contributor-insights --table-name Music
   ```

Habilitar o modo de chaves limitadas

Como usar o Contributor Insights no modo de chaves limitadas

1. Habilite o CloudWatch Contributor Insights para DynamoDB na tabela base `Music` com o modo de chaves limitadas.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

2. Habilite o Contributor Insights no modo de chaves limitadas para o índice secundário global `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --index-name AlbumTitle-index \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Descreva a configuração do Contributor Insights para verificar o modo.

```
aws dynamodb describe-contributor-insights --table-name Music
```

   A resposta exibirá `ContributorInsightsMode` como `THROTTLED_KEYS` e menos regras em comparação com o modo padrão.

Alternar entre os modos

Como alternar entre os modos do Contributor Insights

1. Alterne do modo de chaves limitadas para o modo de chaves acessadas e limitadas.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS
```

2. Alterne do modo de chaves acessadas e limitadas para o modo de chaves limitadas.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Verifique o status durante a transição.

```
aws dynamodb describe-contributor-insights --table-name Music
```

Durante a mudança de modo, `ContributorInsightsStatus` será exibido como `ENABLING`. Assim que concluída, o status será exibido como `ENABLED` no novo modo.

Gerenciar relatórios do Contributor Insights

Como gerenciar as configurações do Contributor Insights

1. Desabilite o CloudWatch Contributor Insights para DynamoDB no índice secundário global `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
                    --table-name Music --index-name AlbumTitle-index \
                    --contributor-insights-action=DISABLE
```

2. Liste todas as configurações do Contributor Insights em sua conta.

```
aws dynamodb list-contributor-insights
```

Isso mostra todos os índices e tabelas com o Contributor Insights habilitado, bem como os respectivos modos.

3. Obtenha informações detalhadas sobre uma configuração específica.

```
aws dynamodb describe-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index
```

Exemplos de resposta

Veja abaixo exemplos de resposta que mostram as diferenças entre os modos:

Resposta no modo de chaves acessadas e limitadas

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKC-Music-1234567890123",
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKC-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "ACCESSED_AND_THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:30:00.000Z"
}
```

Resposta no modo de chaves limitadas

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:35:00.000Z"
}
```

Observe que o modo de chaves limitadas tem menos regras (apenas PKT e SKT), o que corresponde a um monitoramento mais direcionado.

Usar o IAM com o CloudWatch Contributor Insights para DynamoDB

Na primeira vez que você habilitar o Amazon CloudWatch Contributor Insights para DynamoDB, o Amazon DynamoDB criará automaticamente uma função vinculada ao serviço do AWS Identity and Access Management (IAM) para você. Essa função `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` permite que o DynamoDB gerencie as regras do CloudWatch Contributor Insights em seu nome. Não exclua essa função vinculada ao serviço. Se você excluí-la, todas as regras gerenciadas não serão mais removidas quando excluir a tabela ou o índice secundário global.

Para obter mais informações sobre funções vinculadas a serviços, consulte Usando funções vinculadas a serviços no Guia do Usuário do IAM.

As seguintes permissões são necessárias:

- Para habilitar ou desabilitar o CloudWatch Contributor Insights para DynamoDB, é necessário ter a permissão `dynamodb:UpdateContributorInsights` na tabela ou no índice.
- Para visualizar os gráficos do CloudWatch Contributor Insights para DynamoDB, é necessário ter a permissão `cloudwatch:GetInsightRuleReport`.
- Para descrever o CloudWatch Contributor Insights para DynamoDB para determinada tabela ou índice do DynamoDB, é necessário ter a permissão `dynamodb:DescribeContributorInsights`.
- Para listar os status do CloudWatch Contributor Insights para DynamoDB para cada tabela e cada índice secundário global, é necessário ter a permissão `dynamodb:ListContributorInsights`.

Exemplo: ativar ou desativar o CloudWatch Contributor Insights para DynamoDB

A política do IAM a seguir concede permissões para ativar ou desativar o CloudWatch Contributor Insights para DynamoDB.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
```

```
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        }
    ]
}
```

Para tabelas criptografadas pela chave do KMS, o usuário precisa ter permissões kms:Decrypt para atualizar o Contributor Insights.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:kms:*:*:key/*",
            "Action": [
                "kms:Decrypt"
            ]
        }
    ]
```

```
}
```

Exemplo: recuperar o relatório de regras do CloudWatch Contributor Insights

A política do IAM a seguir concede permissões para recuperar o relatório de regras do CloudWatch Contributor Insights.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

Exemplo: aplicar seletivamente o CloudWatch Contributor Insights para permissões do DynamoDB com base em recurso

A política do IAM a seguir permite as ações `ListContributorInsights` e `DescribeContributorInsights` e nega a ação `UpdateContributorInsights` para um índice secundário global específico.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "dynamodb:ListContributorInsights",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books/
index/Author-index"
        }
    ]
}
```

Usar funções vinculadas ao serviço para o CloudWatch Contributor Insights para DynamoDB

O CloudWatch Contributor Insights para DynamoDB usa funções vinculadas ao serviço do AWS Identity and Access Management (IAM). Uma função vinculada ao serviço é um tipo exclusivo de perfil do IAM que é vinculado diretamente ao CloudWatch Contributor Insights para DynamoDB. As funções vinculadas ao serviço são predefinidas pelo CloudWatch Contributor Insights para DynamoDB e incluem todas as permissões que o serviço requer para chamar outros serviços da AWS em seu nome.

Uma função vinculada ao serviço facilita a configuração do CloudWatch Contributor Insights para DynamoDB porque você não precisa adicionar as permissões necessárias manualmente. O CloudWatch Contributor Insights para DynamoDB define as permissões de suas funções vinculadas ao serviço e, exceto se definido de outra forma, somente o CloudWatch Contributor Insights para DynamoDB poderá assumir suas funções. As permissões definidas incluem a política de confiança e a política de permissões, que não pode ser anexada a nenhuma outra entidade do IAM.

Para obter informações sobre outros serviços compatíveis com funções vinculadas a serviços, consulte Serviços da AWS compatíveis com o IAM e procure os serviços que contenham Sim na coluna Função vinculada ao serviço. Escolha um Sim com um link para visualizar a documentação da função vinculada a esse serviço.

Permissões de funções vinculadas ao serviço para o CloudWatch Contributor Insights para DynamoDB

O CloudWatch Contributor Insights para DynamoDB usa a função vinculada ao serviço chamada AWSServiceRoleForDynamoDBCloudWatchContribuorInsights. O objetivo da função vinculada ao serviço é permitir que o Amazon DynamoDB gerencie as regras do Amazon CloudWatch Contributor Insights criadas para tabelas do e índices secundários globais do DynamoDB em seu nome.

O perfil vinculado ao serviço `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` confia nos seguintes serviços para aceitar o perfil:

- `contributorinsights.dynamodb.amazonaws.com`

A política de permissões da função permite que o CloudWatch Contributor Insights para DynamoDB conclua as seguintes ações nos recursos especificados:

- Ação: `Create and manage Insight Rules` em `DynamoDBContributorInsights`

Você deve configurar permissões para que uma entidade do IAM (por exemplo, um usuário, grupo ou função) crie, edite ou exclua um perfil vinculado a serviço. Para mais informações, consulte Permissões de perfil vinculado ao serviço no Guia do usuário do IAM.

Criar uma função vinculada ao serviço para o CloudWatch Contributor Insights para DynamoDB

Não é necessário criar manualmente uma função vinculada ao serviço. Quando você habilita o Colaborador Insights via AWS Management Console, AWS CLI ou AWS API, o CloudWatch Contributor Insights para DynamoDB cria uma função vinculada ao serviço para você.

Se excluir essa função vinculada ao serviço e precisar criá-la novamente, você poderá usar esse mesmo processo para recriar a função em sua conta. Quando você habilita o Contributor Insights, o CloudWatch Contributor Insights para DynamoDB cria uma função vinculada ao serviço para você mais uma vez.

Editar uma função vinculada ao serviço para o CloudWatch Contributor Insights para DynamoDB

O CloudWatch Contributor Insights para DynamoDB não permite editar a função vinculada ao serviço `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Depois que criar um perfil vinculado ao serviço, você não poderá alterar o nome do perfil, pois várias entidades podem fazer

referência a ele. No entanto, será possível editar a descrição do perfil usando o IAM. Para obter mais informações, consulte Editar um perfil vinculado ao serviço no Guia do usuário do IAM.

Excluir uma função vinculada ao serviço para o CloudWatch Contributor Insights para DynamoDB

Você não precisa excluir manualmente a função `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Quando você desabilita o Contributor Insights via AWS Management Console, AWS CLI ou AWS API, o CloudWatch Contributor Insights para DynamoDB limpa os recursos.

Também é possível usar o console do IAM, a AWS CLI ou a API da AWS para excluir manualmente a função vinculada ao serviço. Para isso, primeiro você deve limpar manualmente os recursos de sua função vinculada ao serviço e depois excluí-la manualmente.

> **Note**
>
> Se o serviço CloudWatch Contributor Insights para DynamoDB estiver usando a função quando você tentar excluir os recursos, poderá haver falha na exclusão. Se isso acontecer, espere alguns minutos e tente a operação novamente.

Como excluir manualmente o perfil vinculado ao serviço usando o IAM

Use o console do IAM, a AWS CLI ou a API da AWS para excluir a função vinculada ao serviço `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Para obter mais informações, consulte Excluir um perfil vinculado ao serviço no Guia do usuário do IAM.

Práticas recomendadas para desenhar e arquitetar com o DynamoDB

Use esta seção para encontrar rapidamente recomendações e maximizar a performance e minimizar os custos de throughput ao trabalhar com o DynamoDB.

Tópicos

- Design em NoSQL para DynamoDB
- Usar Lentes do Well-Architected para DynamoDB para otimizar uma workload do DynamoDB
- Práticas recomendadas para projetar e usar chaves de partição de maneira eficaz no DynamoDB
- Práticas recomendadas para usar chaves de classificação para organizar dados no DynamoDB
- Práticas recomendadas para uso de índices secundários no DynamoDB
- Práticas recomendadas para armazenar itens e atributos grandes no DynamoDB
- Práticas recomendadas para lidar com dados de séries temporais no DynamoDB
- Práticas recomendadas para gerenciar relações de muitos para muitos no DynamoDB
- Práticas recomendadas para consulta e varredura de dados no DynamoDB
- Práticas recomendadas para o design de tabelas do DynamoDB
- Práticas recomendadas para o design de tabelas globais do DynamoDB
- Práticas recomendadas para gerenciar o ambiente de gerenciamento no DynamoDB
- Práticas recomendadas para usar operações de dados em massa no DynamoDB
- Práticas recomendadas para implementar o controle de versão no DynamoDB
- Práticas recomendadas para entender os relatórios de uso e faturamento da AWS no DynamoDB
- Migrar uma tabela do DynamoDB de uma conta para outra
- Recomendações para integrar o DAX às aplicações do DynamoDB

- Considerações ao usar o AWS PrivateLink para Amazon DynamoDB

Design em NoSQL para DynamoDB

Os sistemas de bancos de dados NoSQL, como o Amazon DynamoDB, usam os modelos alternativos para o gerenciamento de dados, como pares de chave-valor ou armazenamento de

documentos. Ao mudar de um sistema de gerenciamento de bancos de dados relacionais para um sistema de banco de dados NoSQL como o DynamoDB, é importante compreender as principais diferenças e as abordagens específicas de design.

Tópicos

- Diferenças entre design relacional de dados e NoSQL
- Dois conceitos-chave para design em NoSQL
- Abordar o design em NoSQL
- NoSQL Workbench para DynamoDB

Diferenças entre design relacional de dados e NoSQL

Os sistemas de bancos de dados relacionais (RDBMS) e os bancos de dados NoSQL têm diferentes pontos fortes e fracos:

- No RDBMS, as consultas de dados são flexíveis, mas têm um custo relativamente alto e não escalam com facilidade em situações de grande volume de tráfego (consulte Primeiras passos para a modelagem de dados relacionais no DynamoDB).
- Em um banco de dados NoSQL como o DynamoDB, há formas limitadas de consultar dados com eficiência. As demais formas de consulta podem apresentar alto custo e baixa performance.

Essas diferenças tornam o design de banco de dados diferente entre os dois sistemas:

- Em RDBMS, você cria o design para obter flexibilidade sem se preocupar com detalhes de implementação ou desempenho. A otimização de consultas geralmente não afeta o design do esquema, mas a normalização é importante.
- No DynamoDB, você deve projetar o esquema especificamente para fazer as consultas mais comuns e importantes do modo mais rápido e barato possível. Suas estruturas de dados são adaptadas aos requisitos específicos de seus casos de uso de negócios.

Dois conceitos-chave para design em NoSQL

O design do NoSQL exige uma visão diferente daquela no design do RDBMS. Para um RDBMS, você pode criar um modelo de dados normalizado sem pensar nos padrões de acesso. Você poderá estendê-lo posteriormente quando surgirem novas perguntas e requisitos de consulta. Você pode organizar cada tipo de dados em sua própria tabela.

Como o design de NoSQL é diferente

- De forma contrastante, você não deve começar a projetar o esquema do DynamoDB até saber a quais perguntas ele precisará responder. É essencial compreender os problemas de negócios e os casos de uso de aplicativo antecipadamente.
- Você deve manter o mínimo de tabelas possível em uma aplicação do DynamoDB. Com menos tabelas, há mais escalabilidade, menos gerenciamento de permissões e menor sobrecarga para o DynamoDB. Isso também pode ajudar a manter os custos de backup mais baixos em geral.

Abordar o design em NoSQL

A primeira etapa ao projetar a aplicação do DynamoDB é identificar os padrões específicos de consulta aos quais o sistema deve atender.

Especificamente, é importante compreender três propriedades fundamentais de padrões de acesso de seu aplicativo antes de começar:

- Tamanho de dados: saber o volume de dados que serão armazenados e solicitados ao mesmo tempo ajudará a determinar a maneira mais eficiente de particionar os dados.
- Forma dos dados: em vez de remodelar dados quando uma consulta é processada (como um sistema RDBMS faz), um banco de dados NoSQL organiza os dados para que sua forma no banco de dados corresponda ao que será consultado. Esse é um fator importante no aumento da velocidade e da escalabilidade.
- Velocidade dos dados: o DynamoDB é escalado aumentando-se o número de partições físicas que estão disponíveis para processar consultas e distribuindo-se os dados com eficiência entre essas partições. Saber antecipadamente qual é o pico das cargas de consulta pode ajudar a determinar como particionar os dados para melhor utilizar a capacidade de E/S.

Após identificar os requisitos específicos da consulta, você pode organizar dados de acordo com os princípios gerais que regem o desempenho:

- Mantenha os dados relacionados juntos. Pesquisas demonstram que o princípio de “localidade de referência” (manter os dados relacionados em um só lugar), há muitos anos considerado importante para otimizar tabelas de roteamento, é um fator fundamental para melhorar a performance e os tempos de resposta em sistemas NoSQL.

Como regra geral, você deve manter o mínimo de tabelas possível em uma aplicação do DynamoDB.

As exceções são os casos que envolvem dados de séries temporais de alto volume ou conjuntos de dados que têm padrões muito diferentes de acesso. Uma única tabela com índices invertidos pode normalmente habilitar consultas simples para criar e recuperar estruturas de dados hierárquicas e complexas, exigidas pelo aplicativo.

- Use a ordem de classificação. Os itens relacionados podem ser agrupados juntos e consultados de modo eficiente se o design da chave fizer com que sejam classificados juntos. Essa é uma estratégia importante de design do NoSQL.
- Distribua as consultas. Também é importante que um alto volume de consultas não se concentre em uma parte do banco de dados, onde podem exceder a capacidade de E/S. Em vez disso, você deve projetar chaves de dados para distribuir o tráfego entre as partições do modo mais uniforme possível, evitando hotspots.
- Use índices secundários globais. Ao criar índices secundários globais específicos, você pode permitir consultas diferentes daquelas aceitas na tabela principal, que sejam rápidas e relativamente econômicas.

Esses princípios gerais traduzem-se em alguns padrões comuns de design que você pode usar para modelar dados no DynamoDB com eficiência.

NoSQL Workbench para DynamoDB

O NoSQL Workbench para DynamoDB é uma aplicação GUI multiplataforma do lado do cliente para operações e desenvolvimento de bancos de dados modernos. Ele está disponível para Windows, macOS e Linux. O NoSQL Workbench é uma ferramenta de desenvolvimento visual que fornece recursos de modelagem de dados, visualização de dados, geração de dados de amostra e desenvolvimento de consultas para ajudar você a projetar, criar, consultar e gerenciar tabelas do DynamoDB. Com o NoSQL Workbench para DynamoDB, você pode criar novos modelos de dados a partir de ou projetar modelos com base em modelos de dados existentes que satisfaçam os padrões de acesso a dados das suas aplicações. Pode também importar e exportar o modelo de dados designado no final do processo. Para obter mais informações, consulte Criar modelos de dados com o NoSQL Workbench.

Usar Lentes do Well-Architected para DynamoDB para otimizar uma workload do DynamoDB

Esta seção descreve o recurso Lentes do Well-Architected para o Amazon DynamoDB, um conjunto de princípios de design e orientações para projetar workloads bem arquitetadas do DynamoDB.

Otimizar os custos nas tabelas do DynamoDB

Esta seção aborda as práticas recomendadas sobre como otimizar os custos das tabelas do DynamoDB. Você deve examinar as estratégias a seguir para ver qual estratégia de otimização de custos é a melhor para atender às suas necessidades e abordá-las de forma iterativa. Cada estratégia fornecerá uma visão geral do que pode estar afetando seus custos, quais sinais procurar e orientações prescritivas sobre como reduzi-los.

Tópicos

- Avaliar seus custos no nível da tabela
- Avaliar o modo de capacidade de uma tabela do DynamoDB
- Avaliar as configurações do ajuste de escala automático de uma tabela do DynamoDB
- Avaliar a seleção de classes de tabela do DynamoDB
- Identificar recursos não utilizados no DynamoDB
- Avaliar os padrões de uso de tabelas do DynamoDB
- Avaliar o uso de fluxos do DynamoDB
- Avaliar a capacidade provisionada para o provisionamento do tamanho certo de uma tabela do DynamoDB

Avaliar seus custos no nível da tabela

A ferramenta Explorador de Custos encontrada no AWS Management Console permite ver os custos divididos por tipo, como taxas de leitura, gravação, armazenamento e backup. Também é possível ver esses custos resumidos por período, como mês ou dia.

Um desafio que os administradores podem enfrentar é precisar revisar os custos apenas de uma tabela específica. Alguns desses dados estão disponíveis por meio do console do DynamoDB ou por meio de chamadas para a API `DescribeTable`, mas, por padrão, o Explorador de Custos não

permite filtrar ou agrupar por custos associados a uma tabela específica. Esta seção mostrará como usar a marcação para realizar análises de custos de tabelas individuais no Explorador de Custos.

Tópicos

- Como visualizar os custos de uma única tabela do DynamoDB
- Visualização padrão do Explorador de Custos
- Como usar e aplicar tags de tabela no Explorador de Custos

Como visualizar os custos de uma única tabela do DynamoDB

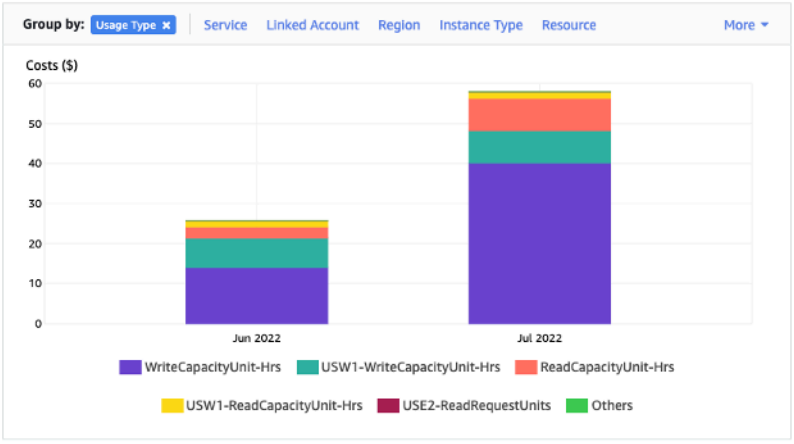
Tanto o AWS Management Console do Amazon DynamoDB quanto a API `DescribeTable` mostrarão informações sobre uma única tabela, incluindo o esquema de chave primária, quaisquer índices na tabela e o tamanho e a contagem de itens da tabela e de quaisquer índices. O tamanho da tabela e o dos índices podem ser usados para calcular o custo mensal de armazenamento da tabela. Por exemplo, USD 0,25 por GB na região us-east-1.

Se a tabela estiver no modo de capacidade provisionada, as configurações atuais de RCU e WCU também serão retornadas. Elas podem ser usadas para calcular os custos atuais de leitura e gravação da tabela, mas esses custos podem mudar, especialmente se a tabela tiver sido configurada com o Auto Scaling.

> **Note**
>
> Se a tabela estiver no modo de capacidade sob demanda, `DescribeTable` não ajudará a estimar os custos de throughput, pois eles são cobrados com base no uso real em cada período, e não no uso provisionado.

Visualização padrão do Explorador de Custos

A visualização padrão do Explorador de Custos fornece gráficos que mostram o custo dos recursos consumidos, como throughput e armazenamento. É possível optar por agrupar os custos por período, como totais por mês ou por dia. Os custos de armazenamento, leituras, gravações e outros recursos também podem ser divididos e comparados.



Como usar e aplicar tags de tabela no Explorador de Custos

Por padrão, o Explorador de Custos não fornece um resumo dos custos de nenhuma tabela específica, pois combinará os custos de várias tabelas em um total. No entanto, é possível usar a marcação de recursos da AWS para identificar cada tabela por uma tag de metadados. As tags são pares de chave-valor que podem ser usadas para diversas finalidades, como identificar todos os recursos pertencentes a um projeto ou departamento. Neste exemplo, vamos supor que você tenha uma tabela chamada MyTable.

1. Defina uma tag com a chave de table_name e o valor de MyTable.
2. Ative a tag no Explorador de Custos e filtre o valor da tag para obter mais visibilidade dos custos de cada tabela.

> **Note**
>
> Pode levar um ou dois dias para que a tag comece a aparecer no Explorador de Custos.

Você mesmo pode definir tags de metadados no console ou por meio de automação, como a CLI da AWS ou o SDK da AWS. Considere exigir que uma tag table_name seja definida como parte do processo de criação de tabelas da sua organização. Para tabelas existentes, há um utilitário Python disponível que encontrará e aplicará essas tags a todas as tabelas existentes em uma determinada região da sua conta. Consulte Eponymous Table Tagger no GitHub para obter mais detalhes.

Avaliar o modo de capacidade de uma tabela do DynamoDB

Esta seção apresenta uma visão geral de como selecionar o modo de capacidade adequado para a tabela do DynamoDB. Cada modo é ajustado para atender às necessidades de uma carga de trabalho diferente em termos de capacidade de resposta a mudanças no throughput, bem como de como esse uso é cobrado. Você deve ponderar sobre esses fatores ao tomar sua decisão.

Tópicos

- Quais modos de capacidade de tabela estão disponíveis
- Quando selecionar o modo de capacidade sob demanda
- Quando selecionar o modo de capacidade provisionada
- Fatores adicionais a serem considerados ao escolher um modo de capacidade de tabela

Quais modos de capacidade de tabela estão disponíveis

Ao criar uma tabela do DynamoDB, é necessário selecionar o modo de capacidade sob demanda ou provisionada.

É possível alternar as tabelas do modo de capacidade provisionada para o modo sob demanda até quatro vezes em um período contínuo de 24 horas. É possível alternar as tabelas do modo sob demanda para o modo de capacidade provisionada a qualquer momento.

Modo de capacidade sob demanda

O modo de capacidade sob demanda foi projetado para eliminar a necessidade de planejar ou provisionar a capacidade da tabela do DynamoDB. Nesse modo, a tabela atenderá instantaneamente às solicitações feitas a ela sem a necessidade de escalar recursos (até o dobro do throughput máximo anterior da tabela).

Para solicitações de leitura e de gravação, o DynamoDB sob demanda oferece o modelo de preço de pagamento por solicitação para que você pague apenas pelo que usar.

Modo de capacidade provisionada

O modo de capacidade provisionada é um modelo mais tradicional em que é possível definir a capacidade que a tabela tem disponível para solicitações feitas diretamente ou com a ajuda do ajuste de escala automático. Como uma capacidade específica é provisionada para a tabela a qualquer momento, o faturamento baseia-se na capacidade total provisionada e não no número de

solicitações consumidas. Ultrapassar a capacidade alocada também pode fazer com que a tabela rejeite solicitações e reduza a experiência dos usuários de suas aplicações.

O modo de capacidade provisionada exige monitoramento constante para encontrar um equilíbrio entre não superprovisionar ou subprovisionar a tabela para manter o controle de utilização baixo e os custos ajustados.

Quando selecionar o modo de capacidade sob demanda

Para otimizar o custo, o modo sob demanda é a melhor opção quando se tem uma workload semelhante aos grafos a seguir.

Os seguintes fatores contribuem para esse tipo de workload:

- Padrão de tráfego que evolui com o tempo
- Volume variável de solicitações (resultante de cargas de trabalho em lote)
- Tempo de solicitação imprevisível (resultando em picos de tráfego)
- Queda para zero ou abaixo de 30% do pico em determinada hora

Com relação a workloads com os fatores acima, usar o ajuste de escala automático para manter capacidade suficiente na tabela e responder aos picos de tráfego provavelmente fará com que a tabela seja superprovisionada e tenha um custo superior ao necessário ou seja subprovisionada e as solicitações tenham um controle de utilização desnecessário. O modo de capacidade sob demanda é a melhor escolha porque pode lidar com tráfego flutuante sem exigir que você preveja nem ajuste a capacidade.

Com o preço de pagamento por solicitação do modo sob demanda, você não precisa se preocupar com a capacidade ociosa, pois paga apenas pelo throughput que realmente usa. Você recebe a cobrança por solicitação de leitura ou de gravação consumida; portanto, seus custos refletem diretamente o uso real, o que facilita o equilíbrio entre custos e performance. Você também pode

configurar o throughput máximo de leitura ou de gravação (ou de ambas) por segundo para tabelas individuais sob demanda e índices secundários globais a fim de ajudar a limitar os custos e o uso. Para ter mais informações, consulte Throughput máximo para tabelas sob demanda.

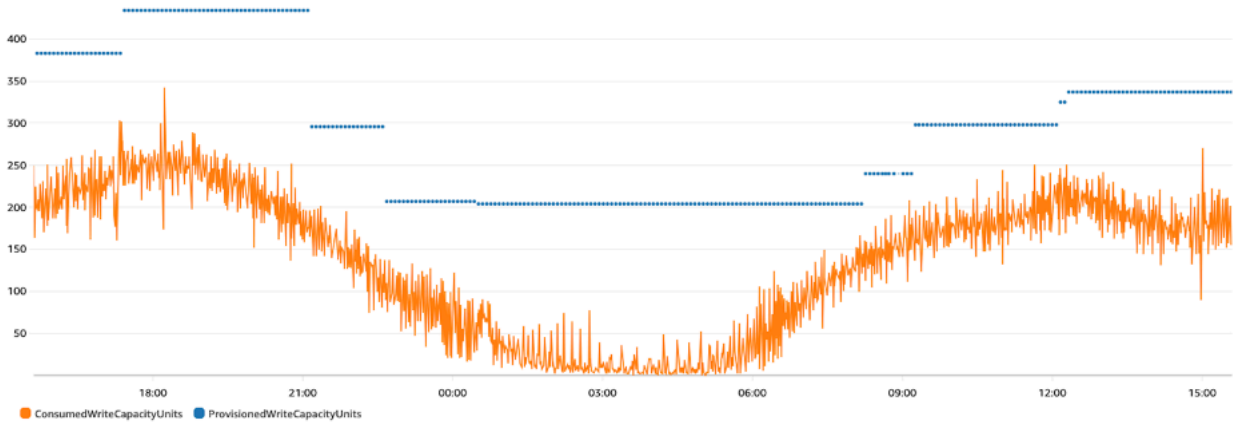
Quando selecionar o modo de capacidade provisionada

Uma workload ideal para o modo de capacidade provisionada é aquela com um padrão de uso mais estável e previsível, como o grafo abaixo.

> **Note**
>
> Recomendamos analisar as métricas em um período detalhado, como 14 dias, antes de utilizar a capacidade provisionada.

Os seguintes fatores contribuem para esse tipo de workload:

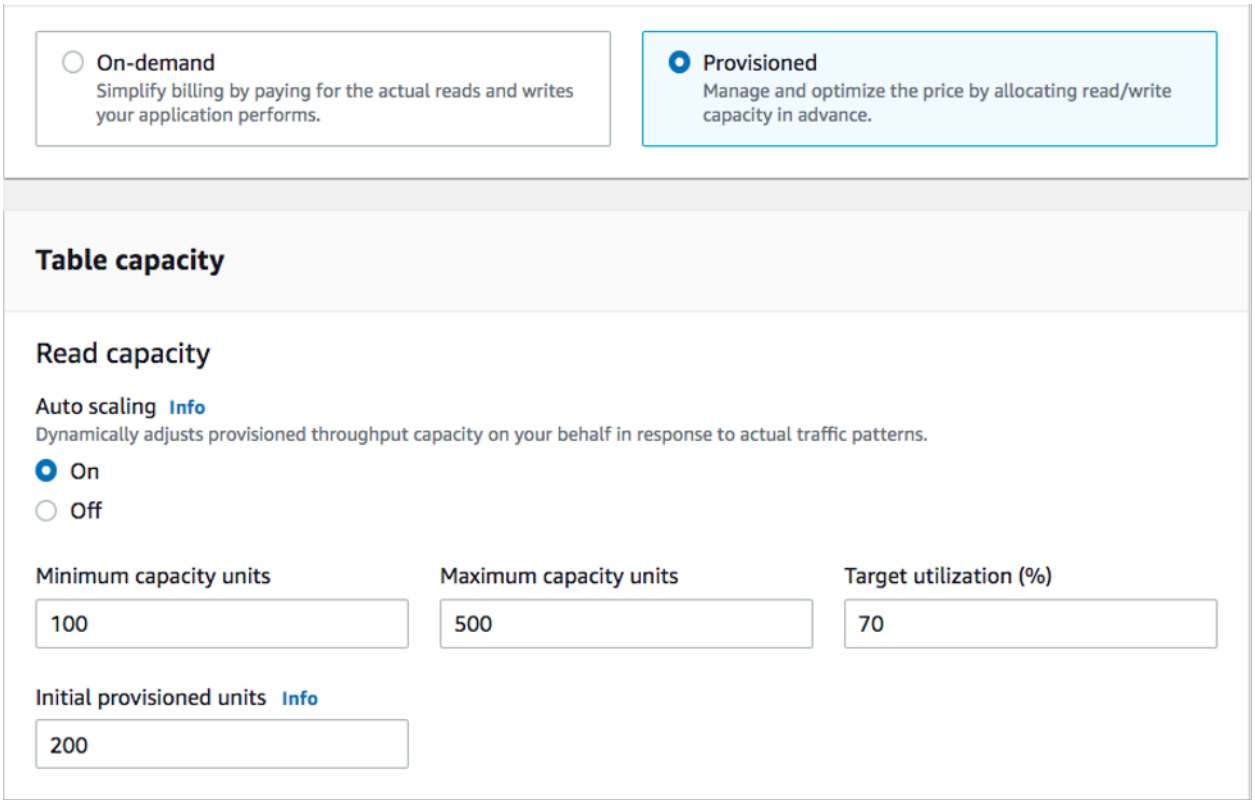
- Tráfego constante, previsível e cíclico em determinada hora ou dia
- Intermitências limitadas de tráfego de curto prazo



Como os volumes de tráfego em determinada hora ou determinado dia são mais estáveis, é possível definir a capacidade provisionada da tabela relativamente próxima à capacidade real consumida. A otimização de custos de uma tabela de capacidade provisionada é, em última análise, um exercício para obter a capacidade provisionada (linha azul) o mais próximo possível da capacidade consumida (linha laranja) sem aumentar `ThrottledRequests` na tabela. O espaço entre as duas linhas representa tanto capacidade desperdiçada quanto garantia contra uma experiência inadequada do usuário devido ao controle de utilização. Se você puder prever os requisitos de throughput da sua

aplicação e preferir a previsibilidade de custos do controle da capacidade de leitura e de gravação, recomenda-se continuar usando tabelas provisionadas.

O DynamoDB fornece autoescalabilidade para tabelas de capacidade provisionada que equilibrarão isso automaticamente em seu nome. Isso permite que você acompanhe a capacidade consumida ao longo do dia e defina a capacidade da tabela com base em algumas variáveis. Ao usar o ajuste de escala automático, sua tabela ficará superprovisionada, e você precisará ajustar a proporção entre o controle de utilização e as unidades de capacidade superprovisionadas para atender às necessidades da workload.



Unidades de capacidade mínima

É possível definir a capacidade mínima de uma tabela para limitar o controle de utilização, mas isso não reduzirá o custo da tabela. Se a tabela tiver períodos de baixo uso seguidos de uma expansão repentina de alto uso, definir o mínimo poderá impedir que a autoescalabilidade defina a capacidade da tabela com um valor muito baixo.

Unidades de capacidade máxima

É possível definir a capacidade máxima de uma tabela para impedir que a tabela escale acima do pretendido. Considere a possibilidade de aplicar um máximo para tabelas de desenvolvimento ou teste em que os testes de carga em grande escala não são desejados. É possível definir um máximo

para qualquer tabela, mas avalie regularmente essa configuração em relação à linha de base da tabela ao usá-la na produção para evitar controle de utilização acidental.

Utilização pretendida

Definir a meta de utilização da tabela é o principal meio de otimização de custos para uma tabela de capacidade provisionada. Definir um valor percentual mais baixo aqui aumentará o superprovisionamento da tabela, o que eleva os custos, mas reduz o risco de controle de utilização. Definir um valor percentual mais alto diminuirá o superprovisionamento da tabela, mas aumentará o risco de controle de utilização.

Fatores adicionais a serem considerados ao escolher um modo de capacidade de tabela

Ao decidir entre os dois modos, há alguns fatores adicionais que vale a pena considerar.

Utilização da capacidade provisionada

Para determinar quando o modo sob demanda custaria menos do que a capacidade provisionada, é útil analisar a utilização da capacidade provisionada, que se refere à eficiência com que os recursos alocados (ou "provisionados") estão sendo usados. O modo sob demanda custa menos para workloads com utilização média da capacidade provisionada abaixo de 35%. Em muitos casos, mesmo para workloads com utilização da capacidade provisionada superior a 35%, pode ser mais econômico usar o modo sob demanda, principalmente se a workload tiver períodos de baixa atividade combinados com picos ocasionais.

Capacidade reservada

Para tabelas de capacidade provisionada, o DynamoDB oferece a opção de comprar capacidade reservada para leitura e gravação [unidades de capacidade de gravação replicada (rWCU) e tabelas Standard-IA não estão qualificadas atualmente]. A capacidade reservada oferece descontos significativos em relação ao preço padrão da capacidade provisionada.

Ao decidir entre os dois modos de tabela, considere quanto esse desconto adicional afetará o custo da tabela. Em alguns casos, pode custar menos executar uma workload relativamente imprevisível do que uma tabela de capacidade superprovisionada com capacidade reservada.

Melhorar a previsibilidade da workload

Em algumas situações, uma workload pode aparentemente ter um padrão previsível e imprevisível. Embora isso possa ser facilmente atendido com uma tabela sob demanda, os custos provavelmente serão melhores se for possível melhorar os padrões imprevisíveis da workload.

Uma das causas mais comuns para esses padrões são as importações em lote. Esse tipo de tráfego geralmente pode exceder a capacidade básica da tabela a tal ponto que o controle de utilização ocorreria se ela fosse executada. Para manter uma workload como essa em execução em uma tabela de capacidade provisionada, considere as seguintes opções:

- Se o lote ocorrer em horários programados, será possível programar um aumento na capacidade de autoescalabilidade antes da execução
- Caso o lote ocorra aleatoriamente, considere a possibilidade de tentar estender o tempo de execução em vez de executá-lo o mais rápido possível
- Adicione um período de aceleração à importação no qual a velocidade da importação começa pequena, mas aumenta lentamente em alguns minutos até que o ajuste de escala automático tenha a oportunidade de começar a ajustar a capacidade da tabela.

Avaliar as configurações do ajuste de escala automático de uma tabela do DynamoDB

Esta seção apresenta uma visão geral de como avaliar as configurações de Auto Scaling nas tabelas do DynamoDB. O Auto Scaling do Amazon DynamoDB é um recurso que gerencia o throughput da tabela e do índice secundário global (GSI) com base no tráfego do sua aplicação e na métrica de utilização desejada. Isso garante que suas tabelas ou GSIs tenham a capacidade necessária para os padrões de sua aplicação.

O serviço de Auto Scaling da AWS monitorará a utilização atual da tabela e a comparará com o valor de utilização alvo: `TargetValue`. Ele notificará você se for hora de aumentar ou diminuir a capacidade alocada.

Tópicos

- Noções básicas de suas configurações de Auto Scaling
- Como identificar tabelas com baixa utilização prevista (<= 50%)
- Como lidar com cargas de trabalho com variação sazonal
- Como lidar com cargas de trabalho com pico com padrões desconhecidos
- Como lidar com workloads com aplicações vinculadas

Noções básicas de suas configurações de Auto Scaling

Definir o valor correto para a utilização prevista, a etapa inicial e os valores finais é uma atividade que exige o envolvimento de sua equipe de operações. Isso permite que você defina

adequadamente os valores com base no histórico de uso da aplicação, que será usado para acionar as políticas de Auto Scaling da AWS. A utilização prevista é a porcentagem de sua capacidade total que deve ser atingida durante um período antes que as regras de Auto Scaling sejam aplicadas.

Quando você define uma utilização prevista alta (uma utilização de cerca de 90%), isso significa que seu tráfego deve ser superior a 90% por um período de tempo antes que o Auto Scaling comece. Você não deve usar uma alta utilização prevista, a menos que sua aplicação seja muito constante e não receba picos de tráfego.

Quando você define uma utilização prevista muito baixa (uma utilização inferior a 50%), isso significa que seu aplicativo deveria atingir 50% da capacidade provisionada antes de acionar uma política de Auto Scaling. A menos que o tráfego da sua aplicação cresça a uma taxa muito agressiva, isso geralmente se traduz em capacidade não utilizada e desperdício de recursos.

Como identificar tabelas com baixa utilização prevista (<= 50%)

Você pode usar a AWS CLI ou o AWS Management Console para monitorar e identificar os `TargetValues` das políticas de Auto Scaling em seus recursos do DynamoDB:

AWS CLI

1. Retorne a lista completa de recursos executando o seguinte comando:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb
   ```

   Esse comando retornará a lista completa de políticas de Auto Scaling emitidas para qualquer recurso do DynamoDB. Se quiser apenas recuperar os recursos de uma tabela específica, você poderá adicionar o `–resource-id parameter`. Por exemplo:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>”
   ```

2. Retorne somente as políticas de Auto Scaling para um GSI específico executando o seguinte comando:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
   ```

Os valores nos quais estamos interessados para as políticas de Auto Scaling estão destacados abaixo. Queremos garantir que o valor alvo seja maior que 50% para evitar o provisionamento excessivo. Você deverá obter um resultado semelhante ao seguinte:

```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName":$<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:ReadCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBReadCapacityUtilization"
                }
            },
            "Alarms": [
```

```
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

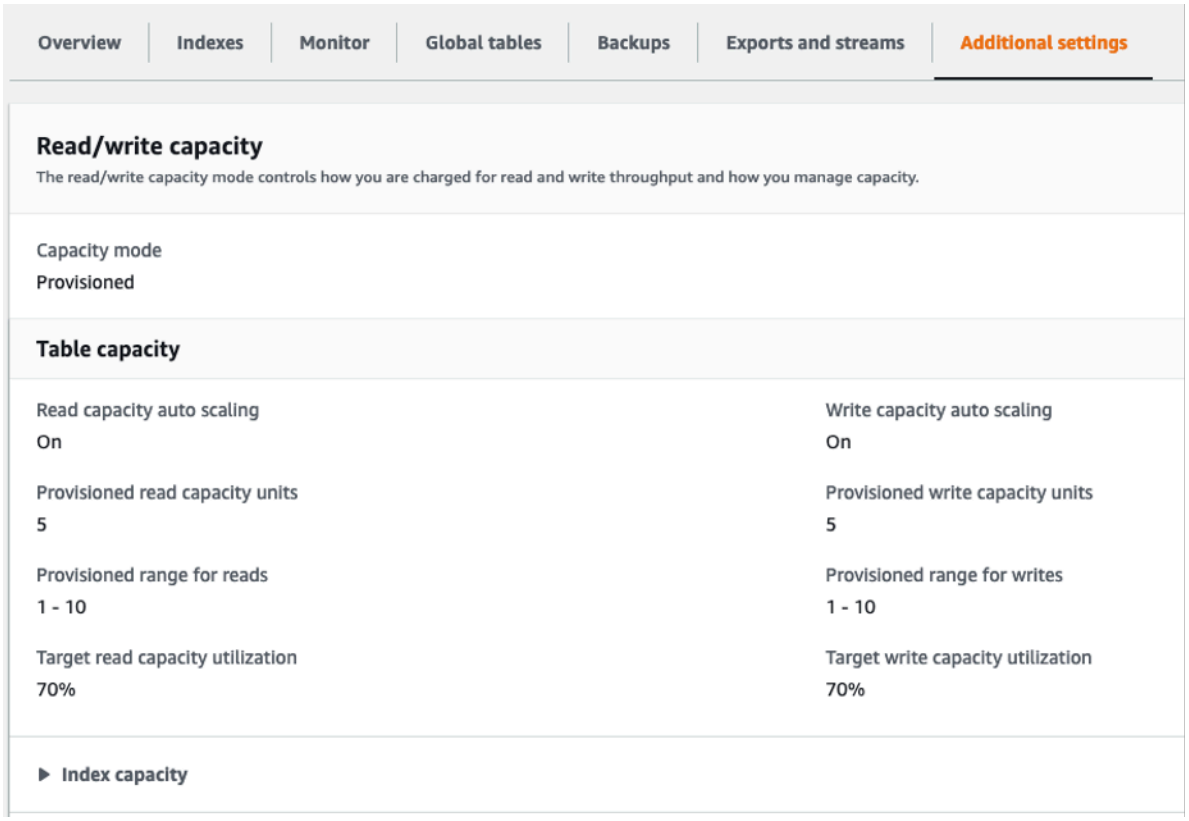
AWS Management Console

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.

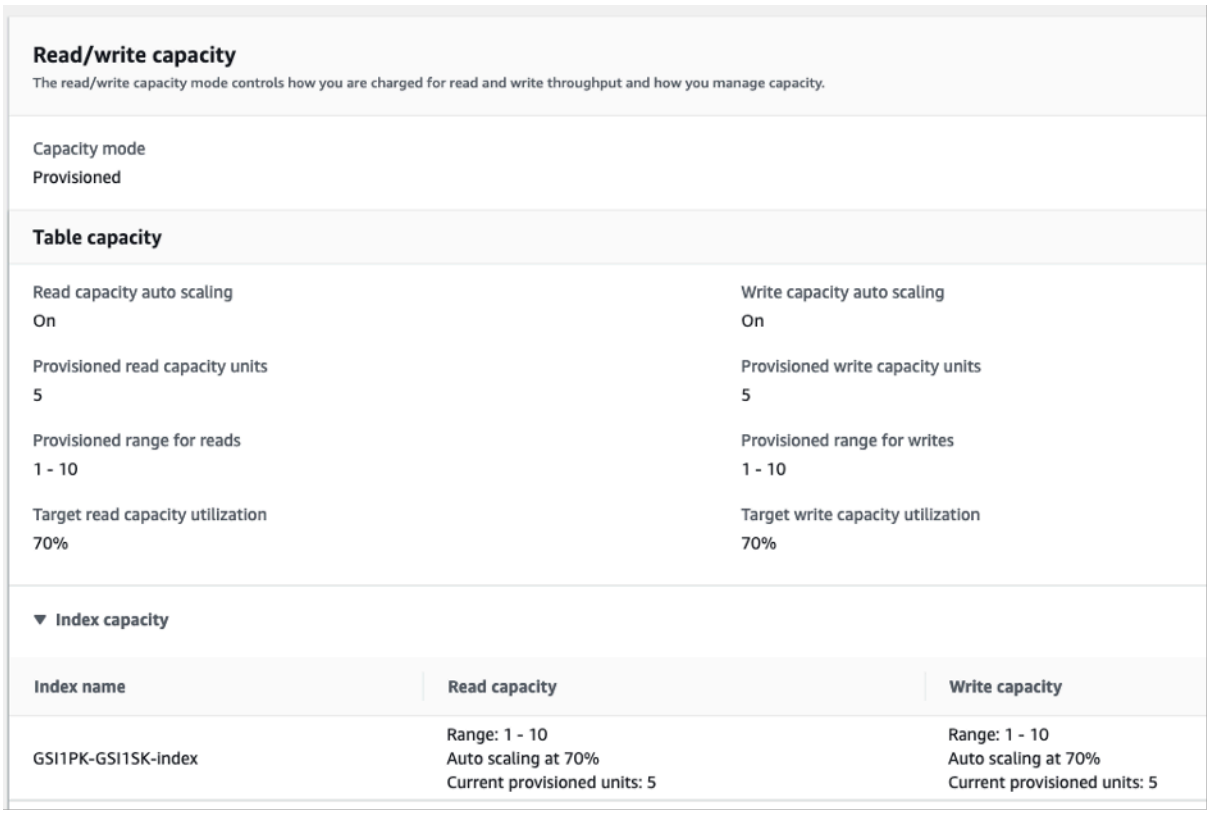
   Selecione uma Região da AWS adequada, se necessário.

2. Na barra de navegação à esquerda, selecione Tabelas. Na página Tabelas, selecione o Nome da tabela.

3. Na página Detalhes da tabela, na guia Configurações adicionais e revise as configurações de ajuste de escala automático da tabela.



Para índices, expanda a guia Capacidade de índice para revisar as configurações do ajuste de escala automático do índice.



Se os valores de suas metas de utilização forem menores ou iguais a 50%, você deverá explorar as métricas de utilização da tabela para ver se elas estão subprovisionadas ou superprovisionadas.

Como lidar com cargas de trabalho com variação sazonal

Considere o seguinte cenário: sua aplicação está operando abaixo de um valor médio mínimo na maioria das vezes, mas a meta de utilização é baixa para que sua aplicação possa reagir rapidamente a eventos que acontecem em determinadas horas do dia e você tenha capacidade suficiente e evite ter controle de utilização. Esse cenário é comum quando você tem uma aplicação muito movimentada durante o horário normal de expediente (das 9h às 17h), mas funciona em um nível básico após o expediente. Como alguns usuários começarão a se conectar antes das 9h, a aplicação usa esse limite baixo para aumentar rapidamente a capacidade necessária durante os horários de pico.

Esse cenário pode ser parecido com:

- Entre 17h e 9h, as unidades `ConsumedWriteCapacity` ficam entre 90 e 100

- Os usuários começam a se conectar à aplicação antes das 9h e as unidades de capacidade aumentam consideravelmente (o valor máximo que você viu é 1,5 mil WCU)
- Em média, o uso da sua aplicação varia entre 800 e 1.200 durante o horário comercial

Se o cenário anterior se aplicar a você, considere usar o Auto Scaling programado, em que sua tabela ainda pode ter uma regra de Auto Scaling de aplicações configurada, mas com uma utilização prevista menos agressiva que forneça somente a capacidade extra nos intervalos específicos necessários.

Você pode usar a AWS CLI para executar as etapas a seguir para criar uma regra de Auto Scaling programado que será executada com base na hora do dia e no dia da semana.

1. Registre sua tabela do DynamoDB ou GSI como destino escalável no Application Auto Scaling. Um destino escalável é um recurso cuja escala pode ser aumentada ou reduzida horizontalmente pelo Application Auto Scaling.

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace dynamodb \
       --scalable-dimension dynamodb:table:WriteCapacityUnits \
       --resource-id table/<table-name> \
       --min-capacity 90 \
       --max-capacity 1500
   ```

2. Configure ações programadas de acordo com seus requisitos.

   Precisaremos de duas regras para cobrir o cenário: uma para aumentar e outra para reduzir a escala verticalmente. A primeira regra para aumentar a escala verticalmente da ação programada:

   ```
   aws application-autoscaling put-scheduled-action \
       --service-namespace dynamodb \
       --scalable-dimension dynamodb:table:WriteCapacityUnits \
       --resource-id table/<table-name> \
       --scheduled-action-name my-8-5-scheduled-action \
       --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
       --schedule "cron(45 8 ? * MON-FRI *)" \
       --timezone "Australia/Brisbane"
   ```

   A segunda regra para reduzir a escala verticalmente da ação programada:

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

3. Execute o seguinte comando para validar que ambas as regras foram ativadas:

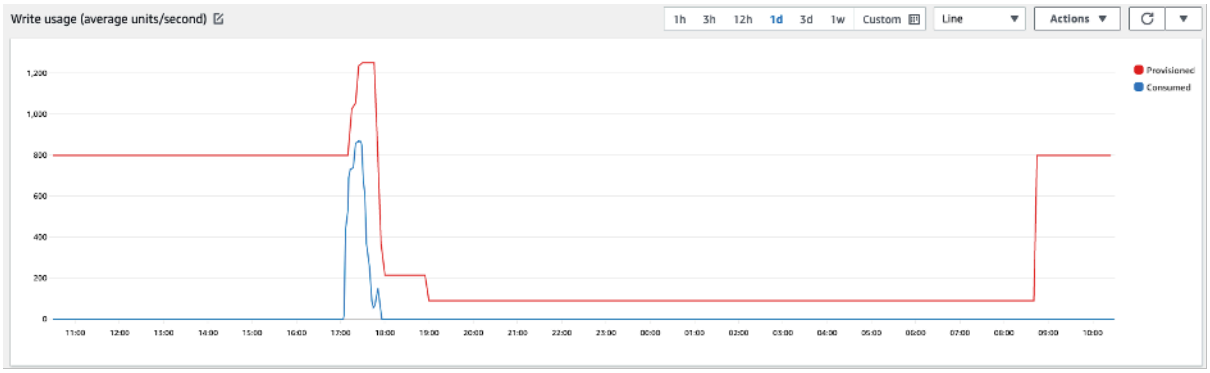
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

Você deve obter um resultado parecido com este:

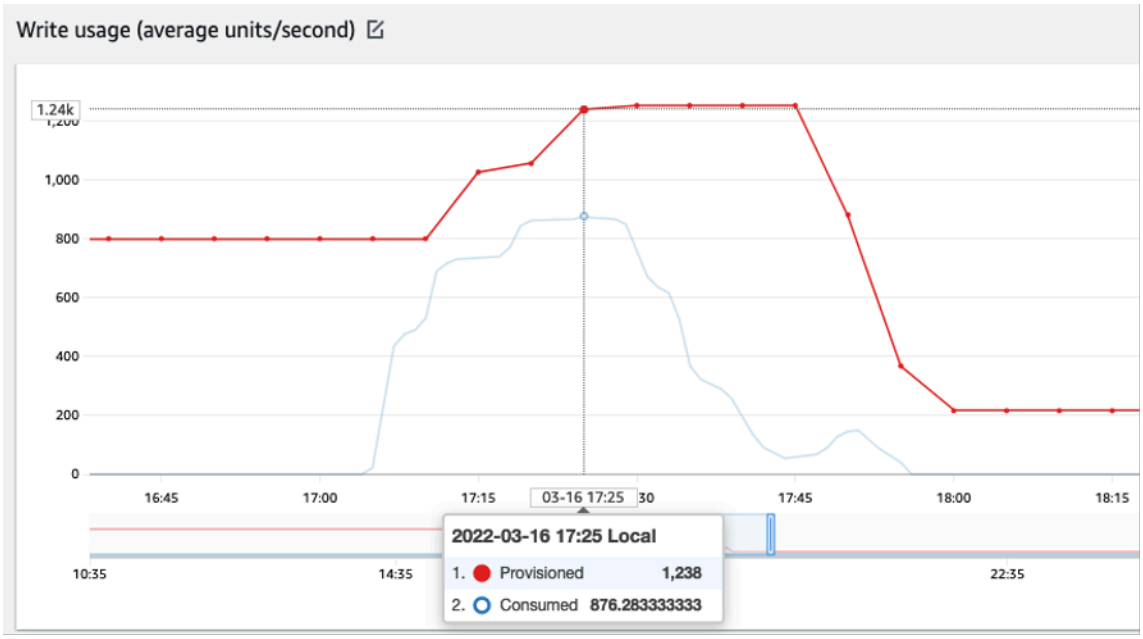
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
```

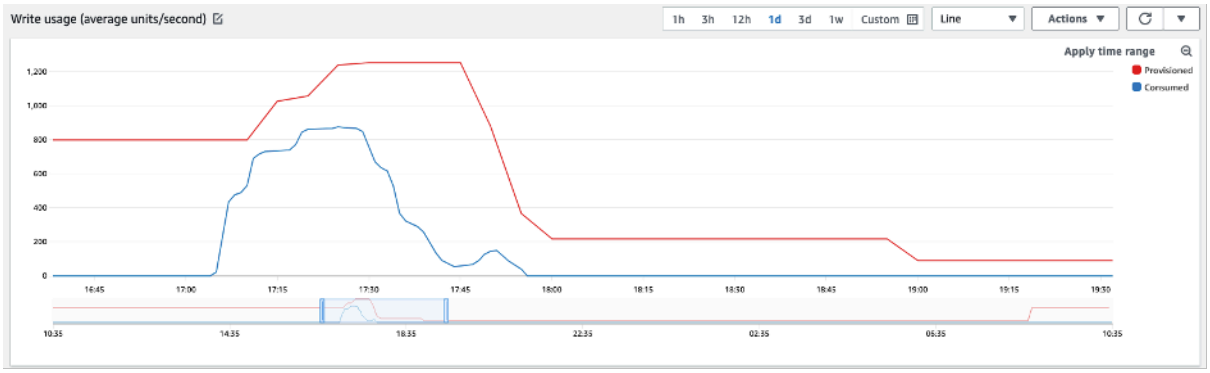
```
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

A figura a seguir mostra um exemplo de workload que sempre mantém a utilização prevista de 70%. Observe como as regras de Auto Scaling ainda se aplicam e o throughput não será reduzido.



Ao ampliar, podemos ver que houve um pico na aplicação que acionou o limite do ajuste de escala automático de 70%, forçando o ajuste de escala automático a entrar em ação e fornecer a capacidade extra necessária para a tabela. A ação programada de Auto Scaling afetará os valores máximo e mínimo, e é sua responsabilidade configurá-los.





Como lidar com cargas de trabalho com pico com padrões desconhecidos

Nesse cenário, a aplicação usa uma utilização prevista muito baixa porque você ainda não conhece os padrões da aplicação e quer garantir que sua workload não tenha controle de utilização.

Em vez disso, considere usar o modo de capacidade sob demanda. As tabelas sob demanda são perfeitas para cargas de trabalho com pico nas quais você não conhece os padrões de tráfego. Com o modo de capacidade sob demanda, você paga por solicitação pelas leituras e gravações de dados que sua aplicação executa em suas tabelas. Não é necessário especificar o throughput de leitura e gravação que você espera que sua aplicação execute, pois o DynamoDB acomoda instantaneamente o crescimento e redução das workloads.

Como lidar com workloads com aplicações vinculadas

Nesse cenário, a aplicação depende de outros sistemas, como cenários de processamento em lote, nos quais você pode ter grandes picos de tráfego de acordo com os eventos na lógica da aplicação.

Considere desenvolver uma lógica personalizada de Auto Scaling que reaja aos eventos em que você pode aumentar a capacidade da tabela e dos `TargetValues` dependendo de suas necessidades específicas. Você pode se beneficiar do Amazon EventBridge e usar uma combinação de serviços da AWS, como o Lambda e o Step Functions, para reagir às necessidades específicas da sua aplicação.

Avaliar a seleção de classes de tabela do DynamoDB

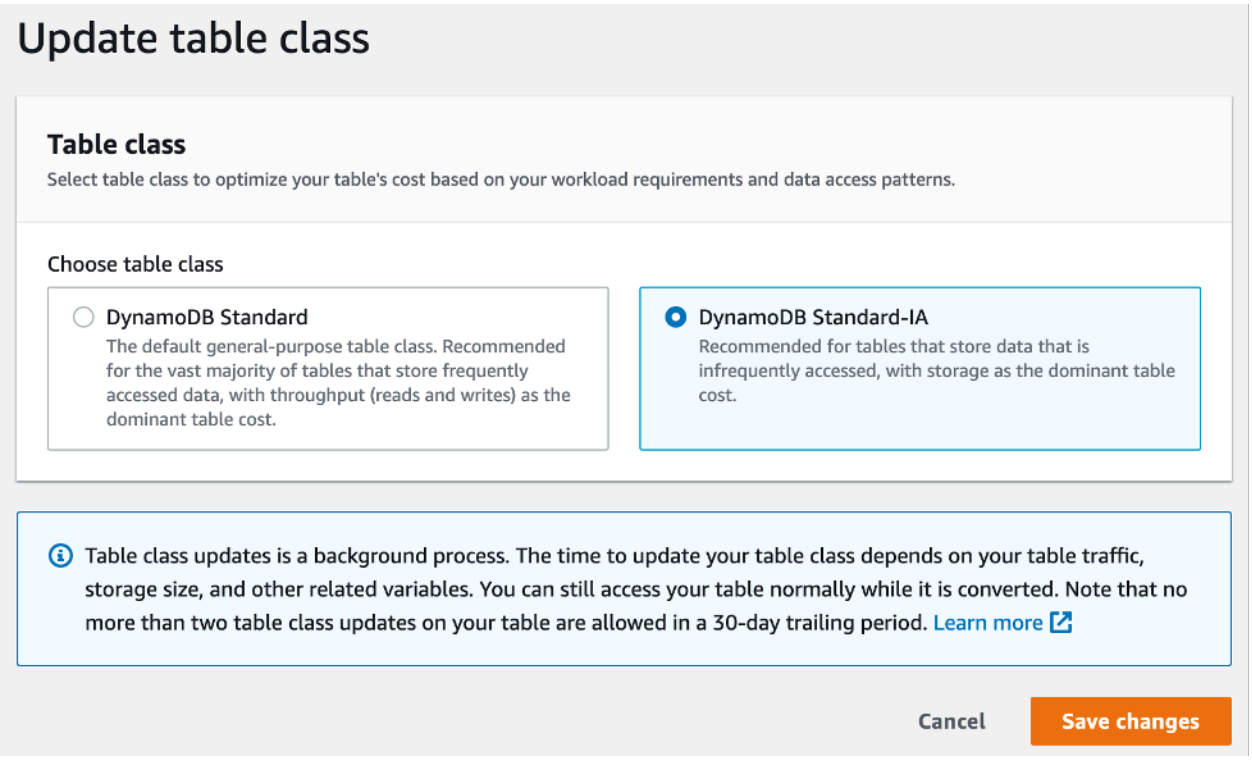
Esta seção apresenta uma visão geral de como selecionar a classe de tabela adequada do DynamoDB. Com o lançamento da classe de tabela Standard Infrequent-Access (Standard-IA), agora é possível otimizar uma tabela para reduzir o custo de armazenamento ou do throughput.

Tópicos

- Quais classes de tabela estão disponíveis
- Quando selecionar a classe de tabela padrão do DynamoDB
- Quando selecionar a classe de tabela Standard-IA do DynamoDB
- Fatores adicionais a serem considerados ao escolher uma classe de tabela

Quais classes de tabela estão disponíveis

Ao criar uma tabela do DynamoDB, é necessário selecionar o DynamoDB Standard ou o DynamoDB Standard-IA como a classe da tabela. É possível alterar a classe da tabela duas vezes em um período de 30 dias; portanto sempre é possível alterá-la no futuro. A seleção de qualquer classe de tabela não afeta a performance, a disponibilidade, a confiabilidade ou a durabilidade da tabela.



Classe de tabela Standard

A classe de tabela Standard é a opção padrão para novas tabelas. Essa opção mantém o equilíbrio do faturamento original do DynamoDB, que oferece um equilíbrio entre os custos do throughput e do armazenamento das tabelas com dados acessados com frequência.

Classe de tabela Standard-IA

A classe de tabela Standard-IA oferece redução do custo de armazenamento (~ 60% menor) para workloads que exigem armazenamento de longo prazo com atualizações ou leituras pouco frequentes. Como a classe é otimizada para acesso pouco frequente, as leituras e gravações serão cobradas a um custo um pouco mais alto (~ 25% maior) ao da classe de tabela Standard.

Quando selecionar a classe de tabela padrão do DynamoDB

A classe de tabela Standard do DynamoDB é mais adequada para tabelas cujo custo de armazenamento é aproximadamente 50% do custo mensal geral da tabela ou inferior. Esse equilíbrio de custos é indicativo de uma workload que acessa ou atualiza regularmente itens já armazenados no DynamoDB.

Quando selecionar a classe de tabela Standard-IA do DynamoDB

A classe de tabela Standard-IA do DynamoDB é mais adequada para tabelas cujo custo de armazenamento é aproximadamente 50% do custo mensal geral da tabela ou superior. Esse equilíbrio de custos é indicativo de uma workload que cria ou lê menos itens por mês do que mantém no armazenamento.

Um uso comum da classe de tabela Standard-IA é mover dados acessados com menor frequência para tabelas individuais do Standard-IA. Para obter mais informações, consulte Optimizing the storage costs of your workloads with Amazon DynamoDB Standard-IA table class (Otimizar os custos de armazenamento de suas workloads com a classe de tabela Standard-IA do Amazon DynamoDB).

Fatores adicionais a serem considerados ao escolher uma classe de tabela

Ao decidir entre as duas classes de tabela, há alguns fatores adicionais que vale a pena considerar como parte da sua decisão.

Capacidade reservada

Atualmente, não é possível comprar capacidade reservada para tabelas que usam a classe de tabela Standard-IA. Ao fazer a transição de uma tabela Standard com capacidade reservada para uma tabela Standard-IA sem capacidade reservada, você pode não ver um benefício de custo.

Identificar recursos não utilizados no DynamoDB

Esta seção apresenta uma visão geral de como avaliar recursos não utilizados regularmente. À medida que os requisitos da aplicação evoluem, você deve garantir que nenhum recurso não seja utilizado e contribua para custos desnecessários do Amazon DynamoDB. Os procedimentos descritos abaixo usarão as métricas do Amazon CloudWatch para identificar recursos não utilizados e ajudarão você a identificar e agir com base nesses recursos para reduzir custos.

É possível monitorar o Amazon DynamoDB usando o CloudWatch, o qual coleta e processa dados brutos do DynamoDB e os transforma em métricas legíveis quase em tempo real. Essas estatísticas são retidas por um período de tempo, para que você possa acessar informações do histórico e entender melhor a utilização. Por padrão, os dados de métrica do DynamoDB são enviados para o CloudWatch automaticamente. Para obter mais informações, consulte O que é o Amazon CloudWatch? e Retenção de métricas no Guia do usuário do Amazon CloudWatch.

Tópicos

- Como identificar recursos não utilizados

- Identificar recursos de tabela não utilizados
- Limpar recursos de tabela não utilizados
- Identificar recursos GSI não utilizados
- Limpar recursos GSI não utilizados
- Limpar tabelas globais não utilizadas
- Limpar backups não utilizados ou recuperação a um ponto anterior no tempo (PITR)

Como identificar recursos não utilizados

Para identificar tabelas ou índices não utilizados, analisaremos as seguintes métricas do CloudWatch durante um período de 30 dias para entender se há alguma leitura ou gravação ativa na tabela ou alguma leitura nos índices secundários globais (GSIs):

ConsumedReadCapacityUnits

O número de unidades de capacidade de leitura consumidas ao longo do período especificado para que você possa acompanhar quanto da capacidade consumida foi usada. Você pode recuperar a capacidade de leitura total consumida para uma tabela e todos os seus índices secundários globais ou para um índice secundário global específico.

ConsumedWriteCapacityUnits

O número de unidades de capacidade de gravação consumidas ao longo do período especificado para que você possa acompanhar quanto da capacidade consumida foi usada. Você pode recuperar a capacidade de gravação total consumida para uma tabela e todos os seus índices secundários globais ou para um índice secundário global específico.

Identificar recursos de tabela não utilizados

O Amazon CloudWatch é um serviço de monitoramento e capacidade de observação que fornece métricas de tabela do DynamoDB usadas para identificar recursos não utilizados. As métricas do CloudWatch podem ser visualizadas por meio do AWS Management Console e do AWS Command Line Interface.

AWS Command Line Interface

Para visualizar as métricas das tabelas por meio do AWS Command Line Interface, é possível usar os seguintes comandos.

1. Primeiro, avalie as leituras da tabela:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

Para evitar a identificação errada de uma tabela como não usada, avalie as métricas por um período mais longo. Escolha um intervalo adequado de início e término, como 30 dias, e um período apropriado, como 86400.

Nos dados retornados, qualquer soma acima de 0 indica que a tabela que você está avaliando teve tráfego de leitura durante esse período.

O resultado a seguir mostra uma tabela com tráfego de leitura no período avaliado:

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

O resultado a seguir mostra uma tabela sem tráfego de leitura no período avaliado:

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. Em seguida, avalie as gravações da tabela:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

Para evitar a identificação errada de uma tabela como não usada, avalie as métricas por um período mais longo. Escolha um intervalo adequado de início e término, como 30 dias, e um período apropriado, como 86400.

Nos dados retornados, qualquer soma acima de 0 indica que a tabela que você está avaliando teve tráfego de leitura durante esse período.

O resultado a seguir mostra uma tabela com tráfego de gravação no período avaliado:

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

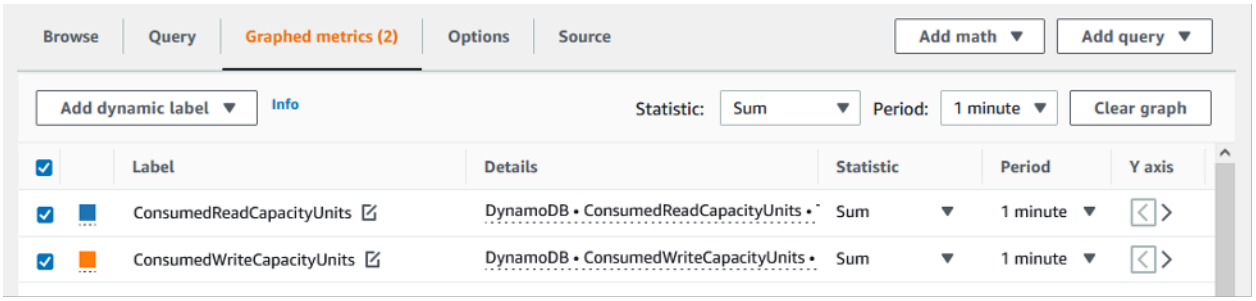
O resultado a seguir mostra uma tabela sem tráfego de gravação no período avaliado:

```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

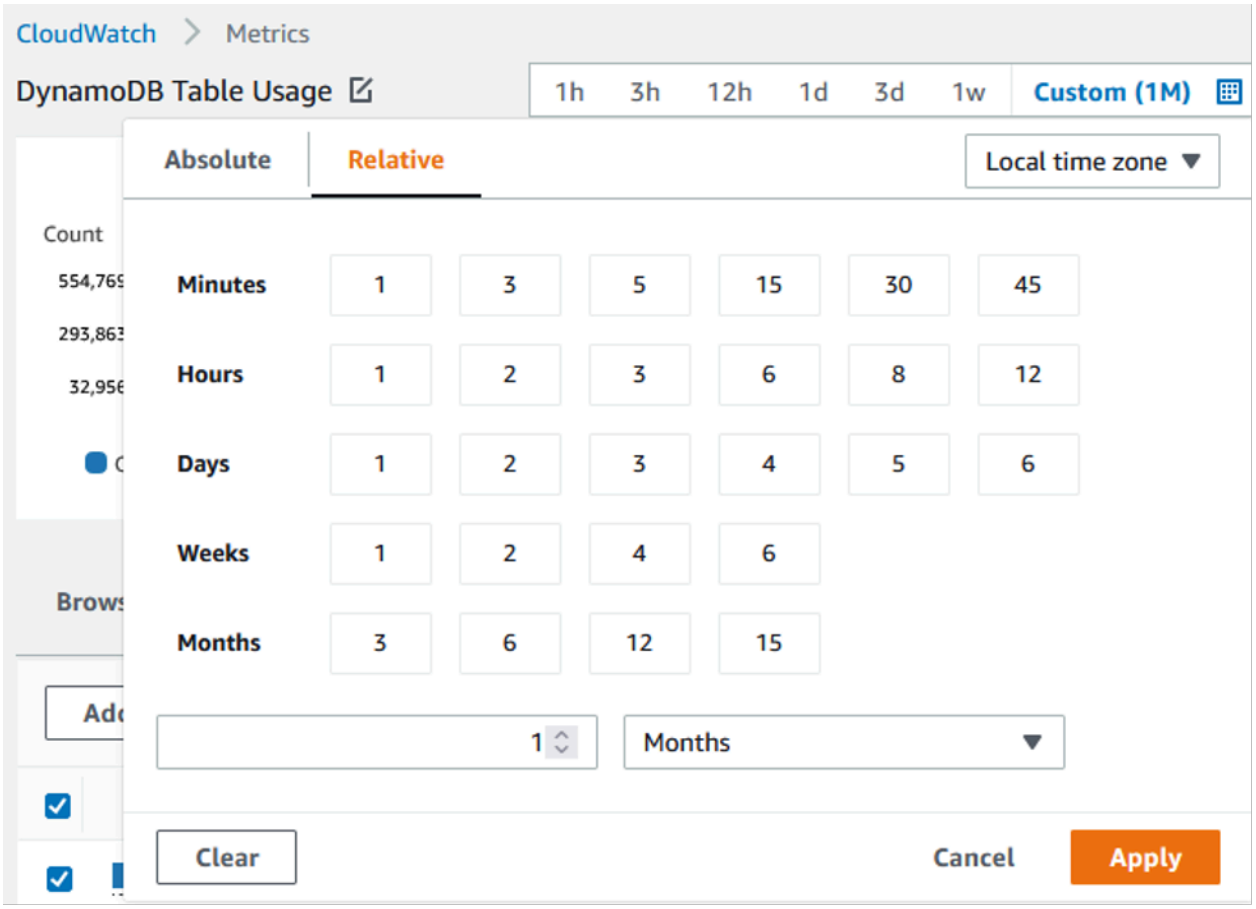
AWS Management Console

As etapas a seguir permitirão avaliar a utilização dos recursos por meio do AWS Management Console.

1. Faça login no console da AWS e vá para a página de serviço do CloudWatch: https://console.aws.amazon.com/cloudwatch/. Selecione a região da AWS apropriada no canto superior direito do console, se necessário.
2. Na barra de navegação à esquerda, localize a seção Metrics (Métricas) e selecione All metrics (Todas as métricas).
3. A ação acima abrirá um painel com dois painéis. No painel superior, você verá as métricas atualmente representadas graficamente. Na parte inferior, você selecionará as métricas disponíveis para representar graficamente. Selecione DynamoDB no painel inferior.
4. No painel de seleção de métricas do DynamoDB, selecione a categoria Table Metrics (Métricas de tabela) para mostrar as métricas das tabelas na região atual.
5. Identifique o nome da sua tabela rolando o menu para baixo e, em seguida, selecione as métricas das `ConsumedReadCapacityUnits` e das `ConsumedWriteCapacityUnits` para a tabela.
6. Selecione a guia Graphed metrics (2) [Métricas gráficas (2)] e ajuste a coluna Statistic (Estatística) como Sum (Soma).

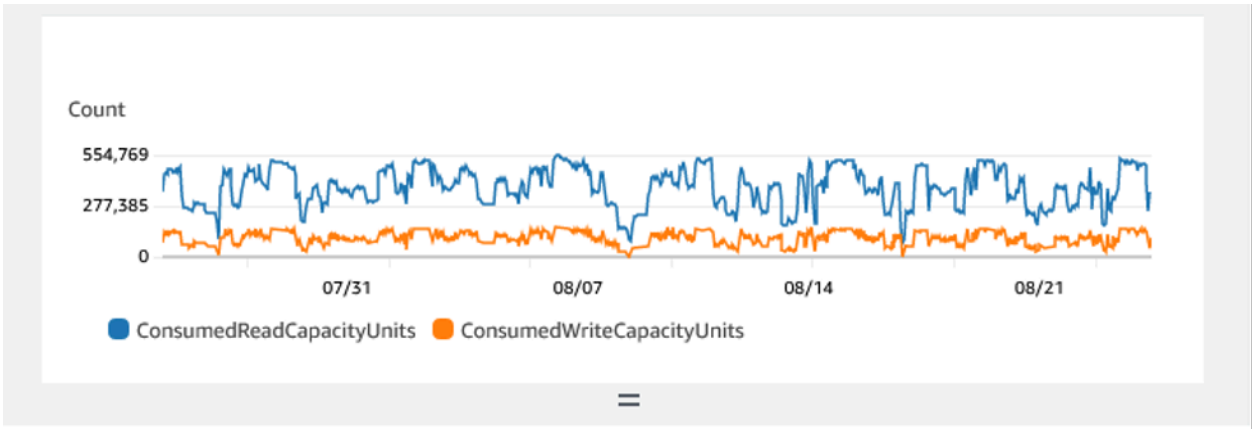


7. Para evitar a identificação errada de uma tabela como não usada, avalie as métricas por um período mais longo. Na parte superior do painel gráfico, escolha um período de tempo apropriado, como um mês, para avaliar a tabela. Selecione Custom (Personalizado), selecione 1 Months (Um mês) nos menus suspensos e escolha Apply (Aplicar).

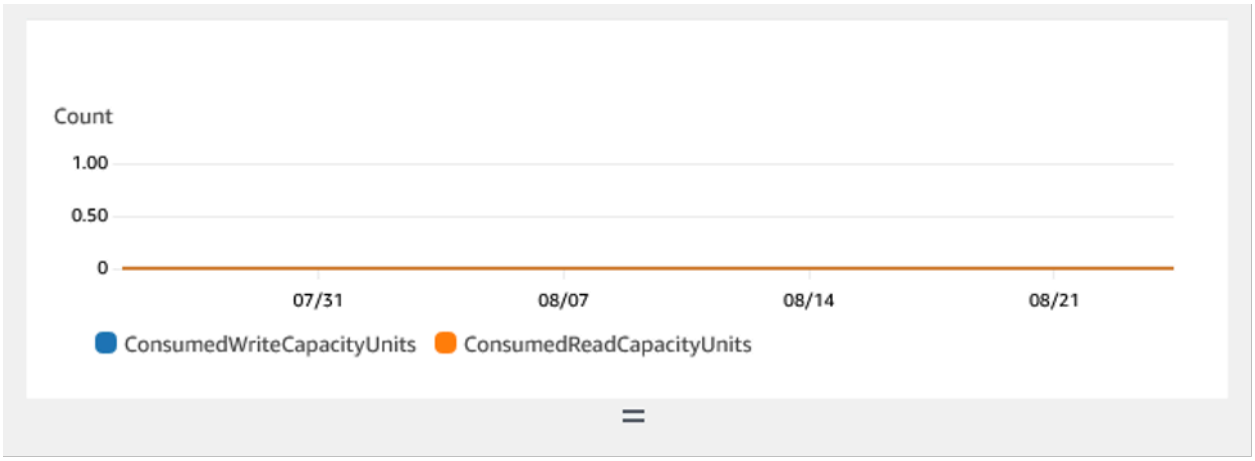


8. Avalie as métricas gráficas da tabela para determinar se ela está sendo usada. Métricas acima de 0 indicam que uma tabela foi usada durante o período avaliado. Um gráfico plano em 0 para leitura e gravação indica uma tabela que não está sendo usada.

   A imagem a seguir mostra uma tabela com tráfego de leitura:



   A imagem a seguir mostra uma tabela sem tráfego de leitura:



Limpar recursos de tabela não utilizados

Se você identificou recursos de tabela não utilizados, poderá reduzir os custos contínuos das seguintes formas.

> **Note**
>
> Se você identificou uma tabela não utilizada, mas ainda gostaria de mantê-la disponível caso ela precise ser acessada no futuro, considere mudá-la para o modo sob demanda. Caso contrário, você pode considerar fazer backup e excluir totalmente a tabela.

Modos de capacidade

O DynamoDB cobra pela leitura, gravação e armazenamento de dados nas respectivas tabelas.

O DynamoDB oferece dois modos de capacidade, que vêm com opções de faturamento específicas do processamento de leitura e gravação nas tabelas: sob demanda e provisionada. O modo de capacidade de leitura/gravação controla como você é cobrado por throughput de leitura e gravação e como você gerencia a capacidade.

Para tabelas do modo sob demanda, não é necessário especificar o throughput de leitura e gravação que você espera que sua aplicação execute. O DynamoDB cobra pelas leituras e gravações que sua aplicação realiza nas tabelas em termos de unidades de solicitação de leitura e unidades de solicitação de gravação. Se não houver atividade na tabela/no índice, você não pagará pelo throughput, mas ainda terá uma taxa de armazenamento.

Classe de tabela

O DynamoDB também oferece duas classes de tabela projetadas para ajudar a otimizar o custo. A classe de tabela Standard do DynamoDB é a padrão e recomendada para a grande maioria das workloads. A classe de tabela do DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) é otimizada para tabelas em que o armazenamento é o custo dominante.

Se não houver atividade na tabela ou no índice, é provável que o armazenamento seja o custo dominante e a mudança de classe de tabela oferecerá uma economia significativa.

Excluir tabelas

Se você descobriu uma tabela não utilizada e gostaria de excluí-la, talvez queira fazer um backup ou exportar os dados antes.

Os backups feitos por meio do Backup da AWS podem aproveitar a hierarquização do armazenamento frio, reduzindo ainda mais os custos. Consulte a documentação Como usar a AWS Backup com o DynamoDB para obter informações sobre como habilitar backups por meio do Backup da AWS, bem como a documentação Gerenciar de planos de backup para obter informações sobre como usar o ciclo de vida para mover o backup para armazenamento frio.

Como alternativa, é possível exportar os dados da tabela para o S3. Para fazer isso, consulte a documentação Exportar para o Amazon S3. Depois de exportar os dados, caso queira aproveitar o S3 Glacier Instant Retrieval, S3 Glacier Flexile Retrieval ou S3 Glacier Deep Archive para reduzir ainda mais os custos, consulte Gerenciar o ciclo de vida do armazenamento.

Depois de fazer o backup da tabela, é possível optar por excluí-la por meio do AWS Management Console ou do AWS Command Line Interface.

Identificar recursos GSI não utilizados

As etapas para identificar um secundário global não utilizado são semelhantes a identificar uma tabela não utilizada. Como o DynamoDB replica itens gravados na tabela-base do GSI, se eles contiverem o atributo usado como chave de partição do GSI, é provável que um GSI não utilizado ainda tenha `ConsumedWriteCapacityUnits` mais de 0 se sua tabela-base estiver em uso. Como resultado, você avaliará somente a métrica das `ConsumedReadCapacityUnits` para determinar se o GSI não está sendo utilizado.

Para visualizar as métricas do GSI por meio da AWS CLI da AWS, é possível usar os seguintes comandos para avaliar as leituras das tabelas:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
```

```
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
Name=GlobalSecondaryIndexName,Value=<index-name>
```

Para evitar a identificação errada de uma tabela como não usada, avalie as métricas por um período mais longo. Escolha um intervalo adequado de início e término, como 30 dias, e um período apropriado, como 86400.

Nos dados retornados, qualquer soma acima de 0 indica que a tabela que você está avaliando teve tráfego de leitura durante esse período.

O resultado a seguir mostra um GSI com tráfego de leitura no período avaliado:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

O resultado a seguir mostra um GSI com tráfego de leitura mínimo no período avaliado:

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
          "Unit": "Count"
        },
```

O resultado a seguir mostra um GSI sem tráfego de leitura no período avaliado:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 0.0,
```

```
      "Unit": "Count"
    },
    {
      "Timestamp": "2022-08-11T21:20:00Z",
      "Sum": 0.0,
      "Unit": "Count"
    },
```

Limpar recursos GSI não utilizados

Se você identificou um GSI não utilizado, poderá excluí-lo. Como todos os dados presentes em um GSI também estão presentes na tabela-base, não é necessário fazer backup adicional antes de excluir um GSI. Se no futuro o GSI for novamente necessário, será possível adicioná-lo novamente à tabela.

Caso você tenha identificado um GSI utilizado com pouca frequência, considere mudanças de design na aplicação que permitiriam excluí-lo ou reduzir o custo. Por exemplo, embora as verificações do DynamoDB devam ser utilizadas com moderação, pois podem consumir grandes quantidades de recursos do sistema, elas poderão ser mais econômicas do que um GSI se o padrão de acesso suportado for utilizado com pouca frequência.

Além disso, caso seja necessário um GSI para oferecer suporte a um padrão de acesso pouco frequente, considere projetar um conjunto mais limitado de atributos. Embora isso possa exigir consultas subsequentes na tabela-base para oferecer suporte aos padrões de acesso pouco frequentes, pode potencialmente oferecer uma redução significativa nos custos de armazenamento e gravação.

Limpar tabelas globais não utilizadas

As tabelas globais do Amazon DynamoDB fornecem uma solução totalmente gerenciada para a implantação de um banco de dados multiativo em várias regiões, sem exigir que você crie e mantenha sua própria solução de replicação.

As tabelas globais são ideais para fornecer acesso de baixa latência a dados próximos aos usuários e também a uma região secundária para recuperação de desastres.

Se a opção de tabelas globais estiver habilitada para um recurso em um esforço para fornecer acesso de baixa latência aos dados, mas não fizer parte de sua estratégia de recuperação de desastres, valide se as duas réplicas estão ativamente fornecendo tráfego de leitura ao avaliar suas métricas do CloudWatch. Se uma réplica não fornecer tráfego de leitura, talvez ela seja um recurso não utilizado.

Se as tabelas globais fizerem parte de sua estratégia de recuperação de desastres, uma réplica sem tráfego de leitura pode ser esperada em um padrão ativo/em espera.

Limpar backups não utilizados ou recuperação a um ponto anterior no tempo (PITR)

O DynamoDB oferece dois estilos de backup. A recuperação a um ponto anterior no tempo fornece backups contínuos por até 35 dias para ajudar a se proteger contra gravações ou exclusões acidentais, enquanto o backup sob demanda permite a criação de snapshots que podem ser salvos a longo prazo. É possível definir o período de recuperação para qualquer valor entre 1 e 35 dias. Os dois tipos de backup têm custos associados a eles.

Consulte a documentação do Backup e restauração para o DynamoDB e da Backups para um ponto no tempo do DynamoDB para determinar se as tabelas têm backups habilitados que podem não ser mais necessários.

Avaliar os padrões de uso de tabelas do DynamoDB

Esta seção apresenta uma visão geral de como avaliar se você está usando suas tabelas do DynamoDB de forma eficiente. Existem certos padrões de uso que não são ideais para o DynamoDB e permitem a otimização tanto do ponto de vista do desempenho quanto do custo.

Tópicos

- Realizar menos operações de leitura altamente consistente
- Executar menos transações para operações de leitura
- Realizar menos verificações
- Encurtar nomes de atributo
- Habilitar a vida útil (TTL)
- Substituir tabelas globais por backups entre regiões

Realizar menos operações de leitura altamente consistente

O DynamoDB permite que você configure a consistência de leitura por solicitação. As solicitações de leitura são finais consistentes por padrão. Leituras finais consistentes são cobradas a 0,5 RCU para até 4 KB de dados.

A maioria das partes das workloads distribuídas é flexível e pode tolerar uma eventual consistência. No entanto, pode haver padrões de acesso que exijam leituras altamente consistentes. Leituras

altamente consistentes são cobradas em 1 RCU para até 4 KB de dados, basicamente dobrando seus custos de leitura. O DynamoDB oferece a flexibilidade de usar os dois modelos de consistência na mesma tabela.

Você pode avaliar sua workload e o código da aplicação para confirmar se leituras altamente consistentes são usadas somente quando necessário.

Executar menos transações para operações de leitura

O DynamoDB permite que você agrupe determinadas ações de uma forma de tudo ou nada, o que significa que você tem a capacidade de executar transações ACID com o DynamoDB. No entanto, como é o caso dos bancos de dados relacionais, não é necessário seguir essa abordagem para cada ação.

Uma operação de leitura transacional de até 4 KB consome 2 RCUs em oposição às 0,5 RCUs padrão para ler a mesma quantidade de dados de uma forma final consistente. Os custos são duplicados nas operações de gravação, o que significa que uma gravação transacional de até 1 KB equivale a 2 WCUs.

Para determinar se todas as operações em suas tabelas são transações, as métricas do CloudWatch para a tabela podem ser filtradas até as APIs de transação. Se as APIs de transação forem os únicos gráficos disponíveis nas`SuccessfulRequestLatency` métricas da tabela, isso confirmaria que cada operação é uma transação para essa tabela. Como alternativa, se a tendência geral de utilização da capacidade corresponder à tendência da API de transação, considere revisitar o design da aplicação, pois ele parece dominado pelas APIs transacionais.

Realizar menos verificações

O uso extensivo de operações `Scan` geralmente decorre da necessidade de executar consultas analíticas nos dados do DynamoDB. Executar operações `Scan` frequentes em uma tabela grande pode ser ineficiente e caro.

Uma alternativa melhor é usar o recurso Exportar para o S3 e escolher um momento para exportar o estado da tabela para o S3. A AWS oferece serviços como o Athena, que podem então ser usados para executar consultas analíticas nos dados sem consumir nenhuma capacidade da tabela.

A frequência das operações `Scan` pode ser determinada usando a estatística `SampleCount` na métrica `SuccessfulRequestLatency` para `Scan`. Se as operações `Scan` forem realmente muito frequentes, os padrões de acesso e o modelo de dados devem ser reavaliados.

Encurtar nomes de atributo

O tamanho total de um item no DynamoDB é a soma dos tamanhos e valores de seus nomes de atributo. Ter nomes de atributos longos não só contribui para os custos de armazenamento, mas também pode levar a um maior consumo de RCU/WCU. Recomendamos que você escolha nomes mais curtos para os atributos. Ter nomes de atributos mais curtos pode ajudar a limitar o tamanho do item dentro do próximo limite de 4 KB/1 KB, após o qual você consumiria RCU/WCU adicional para acessar os dados.

Habilitar a vida útil (TTL)

O Tempo de vida (TTL) pode identificar itens mais antigos do que o tempo de expiração que você definiu em um item e removê-los da tabela. Se seus dados crescerem com o tempo e os dados mais antigos se tornarem irrelevantes, habilitar o TTL na tabela pode ajudar a reduzir seus dados e economizar nos custos de armazenamento.

Outro aspecto útil do TTL é que os itens expirados ocorrem em seus fluxos do DynamoDB, portanto, em vez de apenas remover os dados dos seus dados, é possível consumir esses itens do fluxo e arquivá-los em um nível de armazenamento de menor custo. Além disso, a exclusão de itens via TTL não tem nenhum custo adicional. Ela não consome capacidade e não há sobrecarga na criação de uma aplicação de limpeza.

Substituir tabelas globais por backups entre regiões

As tabelas globais permitem que você mantenha várias tabelas de réplicas ativas em diferentes regiões. Todas elas podem aceitar operações de gravação e replicar dados entre si. É fácil configurar réplicas e a sincronização é gerenciada para você. As réplicas convergem para um estado consistente usando uma estratégia de “último gravador ganha”.

Se estiver usando tabelas globais apenas como parte da estratégia de failover ou recuperação de desastres (DR), você poderá substituí-las por uma cópia de backup entre regiões para obter objetivos de ponto de recuperação relativamente flexíveis e requisitos de objetivo de tempo de recuperação. Se você não precisa de acesso local rápido e da alta disponibilidade de cinco noves, manter uma réplica de tabela global pode não ser a melhor abordagem para o failover.

Como alternativa, considere usar oAWS Backup para gerenciar backups do DynamoDB. Você pode programar backups regulares e copiá-los entre regiões para atender aos requisitos de DR em uma abordagem mais econômica em comparação com o uso de tabelas globais.

Avaliar o uso de fluxos do DynamoDB

Esta seção apresenta uma visão geral de como avaliar o uso do DynamoDB Streams. Existem certos padrões de uso que não são ideais para o DynamoDB e permitem a otimização tanto do ponto de vista do desempenho quanto do custo.

Você tem duas integrações nativas de streaming para casos de uso de streaming e orientados por eventos:

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

Esta página se concentrará apenas nas estratégias de otimização de custos para essas opções. Se você quiser, em vez disso, descobrir como escolher entre as duas opções, consulte Opções de streaming para captura de dados de alteração.

Tópicos

- Otimizar custos para o DynamoDB Streams
- Otimizar os custos para Kinesis Data Streams
- Estratégias de otimização de custos para os dois tipos de uso do Streams

Otimizar custos para o DynamoDB Streams

Conforme mencionado na página de definição de preço do DynamoDB Streams, independentemente do modo de capacidade de throughput da tabela, o DynamoDB cobra pelo número de solicitações de leitura feitas ao DynamoDB Stream da tabela. As solicitações de leitura feitas para um DynamoDB Stream são diferentes das solicitações de leitura feitas para uma tabela do DynamoDB.

Cada solicitação de leitura em termos de fluxo está na forma de uma chamada à API `GetRecords` que pode retornar até 1.000 registros ou 1 MB de registros na resposta, o que for atingido primeiro. Nenhuma das outras APIs do DynamoDB Stream é cobrada e os DynamoDB Streams não são cobrados por estarem ociosos. Em outras palavras, se nenhuma solicitação de leitura for feita para um DynamoDB Stream, nenhuma cobrança será cobrada por ter um DynamoDB Stream habilitado em uma tabela.

Veja algumas aplicações de consumidor do DynamoDB Streams:

- Função(ões) do AWS Lambda

- Aplicações baseadas no Amazon Kinesis Data Streams
- Aplicações para clientes e consumidores criados usando um AWS SDK

As solicitações de leitura feitas por consumidores baseados no AWS Lambda são gratuitas, enquanto as chamadas feitas por consumidores de qualquer outro tipo são cobradas. Todos os meses, os primeiros 2,5 milhões de solicitações de leitura feitas por consumidores não pertencentes ao Lambda também são gratuitas. Isso se aplica a todas as solicitações de leitura feitas a qualquer DynamoDB Streams em uma conta da AWS para cada região da AWS.

Monitorar seu uso do DynamoDB Streams

As cobranças do DynamoDB Streams no console de cobrança são agrupadas para todos os DynamoDB Streams na região da AWS em uma conta da AWS. No momento, a marcação de DynamoDB Streams não é compatível, portanto, não é possível usar as tags de alocação de custos para identificar custos granulares dos DynamoDB Streams. O volume de chamadas `GetRecords` pode ser obtido no nível do DynamoDB Stream para calcular as cobranças por fluxo. O volume é representado pela métrica `SuccessfulRequestLatency` do CloudWatch do DynamoDB Stream e sua estatística `SampleCount`. Essa métrica também incluirá chamadas `GetRecords` feitas por tabelas globais para realizar a replicação contínua, bem como chamadas feitas por consumidores do AWS Lambda, ambas sem cobrança. Para obter informações sobre outras métricas do CloudWatch publicadas pelo DynamoDB Streams, consulte Métricas e dimensões do DynamoDB.

Usar o AWS Lambda como consumidor

Avalie se é possível usar as funções do AWS Lambda como consumidores dos DynamoDB Streams, pois isso pode eliminar os custos associados à leitura do DynamoDB Stream. Por outro lado, os aplicativos de consumo baseados no DynamoDB Streams Kinesis Adapter ou SDK serão cobrados pelo número de chamadas `GetRecords` feitas ao DynamoDB Stream.

As invocações de funções do Lambda serão cobradas com base no preço padrão do Lambda, mas nenhuma cobrança será cobrada pelo DynamoDB Streams. O Lambda sonda os fragmentos em sua transmissão do DynamoDB em busca de registros a uma taxa básica de 4 vezes por segundo. Quando os registros estão disponíveis, o Lambda invoca a função e aguarda o resultado. Se o processamento for bem-sucedido, o Lambda continua a sondagem até que ela receba mais registros.

Ajustar aplicações de consumidor baseadas no DynamoDB Streams Kinesis Adapter

Como as solicitações de leitura feitas por consumidores não baseados no Lambda são cobradas por DynamoDB Stream, é importante encontrar um equilíbrio entre a exigência quase em tempo real e o número de vezes que a aplicação de consumidor deve sondar o DynamoDB Stream.

A frequência da sondagem de DynamoDB Streams usando uma aplicação baseada no DynamoDB Streams Kinesis Adapter é definida pelo valor `idleTimeBetweenReadsInMillis` configurado. Esse parâmetro determina a quantidade de tempo em milissegundos que o consumidor deve esperar antes de processar um fragmento, caso a chamada `GetRecords` anterior feita para o mesmo fragmento não tenha retornado nenhum registro. Por padrão, o valor desse parâmetro é 1.000 ms. Se o processamento quase em tempo real não for necessário, esse parâmetro poderá ser aumentado para que o aplicativo consumidor faça menos chamadas `GetRecords` e otimize as chamadas do DynamoDB Streams.

Otimizar os custos para Kinesis Data Streams

Quando um Kinesis Data Stream é definido como o destino para fornecer eventos de captura de dados de alteração para uma tabela do DynamoDB, o Kinesis Data Stream pode precisar de um gerenciamento de dimensionamento separado, o que afetará os custos gerais. O DynamoDB cobra em termos de unidades de captura de dados de alteração (CDUs), em que cada unidade é composta por um tamanho de item do DynamoDB de até 1 KB tentado pelo serviço do DynamoDB até o Kinesis Data Stream de destino.

Além das cobranças do serviço DynamoDB, serão cobradas taxas padrão do Kinesis Data Stream. Conforme mencionado na página de definição de preço, o preço do serviço difere com base no modo de capacidade, provisionado e sob demanda, que são distintos dos modos de capacidade da tabela do DynamoDB e são definidos pelo usuário. Em um nível alto, o Kinesis Data Streams cobra uma taxa por hora com base no modo de capacidade, bem como na ingestão de dados do fluxo pelo serviço DynamoDB. Pode haver cobranças adicionais, como recuperação de dados (para o modo sob demanda), retenção estendida de dados (além das 24 horas padrão) e recuperações avançadas do consumidor, dependendo da configuração do usuário para o Kinesis Data Stream.

Monitorar seu uso Kinesis Data Streams

O Kinesis Data Streams for DynamoDB publica métricas do DynamoDB, além das métricas padrão do CloudWatch do Kinesis Data Stream. Talvez uma tentativa `Put` do serviço DynamoDB seja limitada pelo serviço Kinesis devido à capacidade insuficiente de Kinesis Data Streams ou por componentes dependentes, como um serviço AWS KMS que possa ser configurado para criptografar os dados do Kinesis Data Stream em repouso.

Para saber mais sobre as métricas do CloudWatch publicadas pelo serviço DynamoDB para o Kinesis Data Stream, consulte Monitorar a captura de dados de alterações com o Kinesis Data Streams. Para evitar custos adicionais de novas tentativas de serviço devido a limitações, é importante dimensionar corretamente o Kinesis Data Stream no caso de Modo provisionado.

Como escolher o modo de capacidade certo para Kinesis Data Streams

O Kinesis Data Streams é compatível com dois modos de capacidade: modo provisionado e modo sob demanda.

- Se a workload envolvendo o Kinesis Data Stream tiver tráfego previsível de aplicações, tráfego consistente ou que aumente gradualmente ou tráfego que possa ser previsto com precisão, o modo provisionado do Kinesis Data Streams é adequado e será mais econômico
- Se a workload for nova, tiver tráfego de aplicações imprevisível ou se você preferir não gerenciar a capacidade, o modo sob demanda do Kinesis Data Streams é adequado e será mais econômico

Uma prática recomendada para otimizar os custos é avaliar se a tabela do DynamoDB associada ao Kinesis Data Stream tem um padrão de tráfego previsível que possa aproveitar o modo provisionado do Kinesis Data Streams. Se a workload for nova, você poderá usar o modo sob demanda para o Kinesis Data Streams por algumas semanas iniciais, analisar as métricas do CloudWatch para entender os padrões de tráfego e, em seguida, mudar o mesmo fluxo para o modo provisionado com base na natureza da workload. No caso do modo provisionado, a estimativa do número de fragmentos pode ser feita seguindo as considerações de gerenciamento de fragmentos para o Kinesis Data Streams.

Avaliar suas aplicações de consumidor do Kinesis Data Streams para DynamoDB

Como o Kinesis Data Streams não cobra pelo número de chamadas `GetRecords`, como o DynamoDB Streams, as aplicações de consumidor podem fazer o maior número possível de chamadas, desde que a frequência esteja abaixo dos limites de controle de utilização de `GetRecords`. Em termos de modo sob demanda para o Kinesis Data Streams, as leituras de dados são cobradas por GB. No modo provisionado de Kinesis Data Streams, as leituras não são cobradas se os dados tiverem menos de sete dias. No caso de funções do Lambda que consumam Kinesis Data Streams, o Lambda sonda cada fragmento em seu Kinesis Stream para identificar registros a uma taxa básica de uma vez por segundo.

Estratégias de otimização de custos para os dois tipos de uso do Streams

Filtragem de eventos para consumidores do AWS Lambda

A filtragem de eventos do Lambda permite que você descarte eventos com base em um critério de filtro para que não estejam disponíveis no lote de invocação da função do Lambda. Isso otimiza os custos do Lambda para processar ou descartar registros de fluxo indesejados dentro da lógica de funções do consumidor. Para saber mais sobre como configurar a filtragem de eventos e escrever seus critérios de filtragem, consulte Filtragem de eventos do Lambda.

Ajustando os consumidores do AWS Lambda

Os custos podem ser otimizados ainda mais ajustando os parâmetros de configuração do Lambda, como aumentar o `BatchSize` para processar mais por invocação, permitindo que `BisectBatchOnFunctionError` evite o processamento de duplicatas (o que gera custos adicionais) e configurando `MaximumRetryAttempts` para não executar muitas tentativas. Por padrão, as invocações do Lambda do consumidor que tiveram falha são repetidas infinitamente até que o registro expire do fluxo, o que dura cerca de 24 horas para o DynamoDB Streams e pode ser configurado de 24 horas a até 1 ano para o Kinesis Data Streams. As opções adicionais de configuração do Lambda disponíveis, incluindo as mencionadas acima para consumidores do DynamoDB Stream, estão no Guia do desenvolvedor do AWS Lambda.

Avaliar a capacidade provisionada para o provisionamento do tamanho certo de uma tabela do DynamoDB

Esta seção apresenta uma visão geral de como avaliar o provisionamento adequado para a tabela do DynamoDB. À medida que sua workload evolui, você deve modificar seus procedimentos operacionais adequadamente, especialmente quando sua tabela do DynamoDB está configurada no modo provisionado e você corre o risco de provisionar demais ou subprovisionar suas tabelas.

Os procedimentos descritos abaixo exigem informações estatísticas que devem ser capturadas das tabelas do DynamoDB que oferecem suporte à sua aplicação de produção. Para entender o comportamento da sua aplicação, você deve definir um período de tempo significativo o suficiente para capturar a sazonalidade dos dados da aplicação. Por exemplo, se a aplicação mostrar padrões semanais, usar um período de três semanas deve fornecer espaço suficiente para analisar as necessidades de throughput da aplicação.

Se não souber por onde começar, use pelo menos um mês de uso de dados para os cálculos abaixo.

Ao avaliar a capacidade, as tabelas do DynamoDB podem configurar unidades de capacidade de leitura (RCUs) e unidades de capacidade de gravação (WCU) de forma independente. Se as suas tabelas tiverem algum Índice Secundário Global (GSI) configurado, você deverá especificar o throughput que ela consumirá, que também será independente das RCUs e WCUs da tabela-base.

> **Note**
>
> Os índices secundários locais (LSI) consomem a capacidade da tabela-base.

Tópicos

- Como recuperar métricas de consumo em suas tabelas do DynamoDB
- Como identificar tabelas do DynamoDB subprovisionadas
- Como identificar tabelas superprovisionadas do DynamoDB

Como recuperar métricas de consumo em suas tabelas do DynamoDB

Para avaliar a tabela e a capacidade do GSI, monitore as seguintes métricas do CloudWatch e selecione a dimensão apropriada para recuperar as informações da tabela ou do GSI:

Unidades de capacidade de leitura	Unidades de capacidade de gravação
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

Você pode fazer isso por meio da AWS CLI ou do AWS Management Console.

AWS CLI

Antes de recuperarmos as métricas de consumo da tabela, precisaremos começar capturando alguns pontos de dados históricos usando a API do CloudWatch.

Comece criando dois arquivos:`write-calc.json` e `read-calc.json`. Esses arquivos representarão os cálculos de uma tabela ou GSI. Você deverá atualizar alguns dos campos, conforme indicado na tabela abaixo, para corresponder ao seu ambiente.

Nome do campo	Definição	Exemplo
`<table-name>`	Nome da tabela que você analisará	SampleTable
`<period>`	O período de tempo que você usará para avaliar a utilização prevista, com base em segundos	Por um período de uma hora, você deve especificar: 3600
`<start-time>`	O início do seu intervalo de avaliação, especificado no formato ISO8601	2022-02-21T23:00:00
`<end-time>`	O final do intervalo de avaliação, especificado no formato ISO8601	2022-02-22T06:00:00

O arquivo de cálculos de gravação recuperará o número de WCU provisionado e consumido no período de tempo para o intervalo de datas especificado. Também gerará uma porcentagem de utilização que será usada para análise. O conteúdo completo do arquivo `write-calc.json` deve ser semelhante a este:

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
```

```
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<ent-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

O arquivo de cálculos de leitura usa um arquivo semelhante. Esse arquivo recuperará quantas RCUs foram provisionadas e consumidas durante o período para o intervalo de datas especificado. O conteúdo do arquivo `read-calc.json` deve ser semelhante a este:

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
```

```
    },
    {
      "Id": "m1",
      "Expression": "consumedRCU/PERIOD(consumedRCU)",
      "Label": "Consumed RCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedRCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

Depois de criar os arquivos, você poderá começar a recuperar os dados de utilização.

1. Para recuperar dados de utilização de gravação, emita o seguinte comando:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
   ```

2. Para recuperar dados de utilização de leitura, emita o seguinte comando:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
   ```

O resultado de ambas as consultas será uma série de pontos de dados no formato JSON que serão usados para análise. Seu resultado dependerá do número de pontos de dados que você especificou, do período e de seus próprios dados específicos da workload. Isso pode ser semelhante a:

```
{
    "MetricDataResults": [
        {
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
```

```
                "2022-02-22T05:00:00+00:00",
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
                24.9496875,
                40.94725694444445,
                25.61819444444444,
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

> **Note**
>
> Se você especificar um período curto e um longo intervalo de tempo, talvez seja necessário modificar o `MaxDatapoints` que, por padrão, está definido como 24 no script. Isso representa um ponto de dados por hora e 24 por dia.

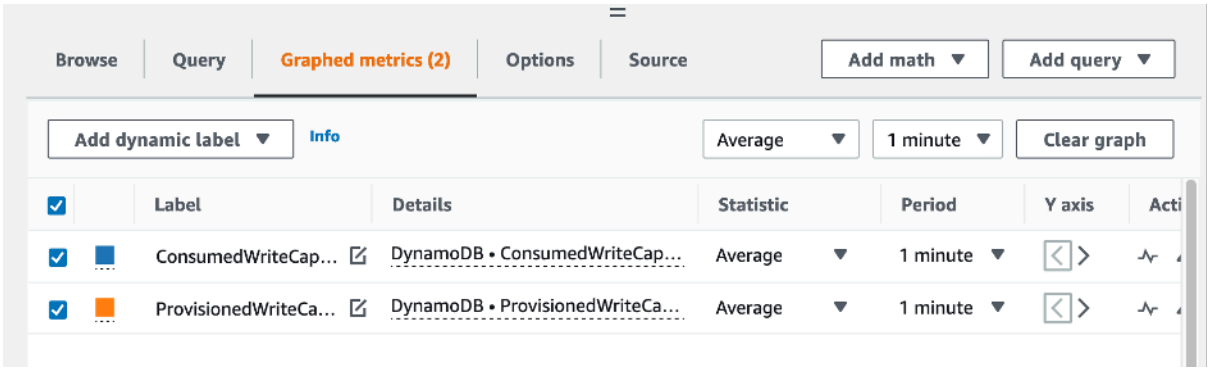
AWS Management Console

1. Faça login no AWS Management Console e acesse a página do serviço CloudWatch. Selecione uma Região da AWS adequada, se necessário.
2. Localize a seção Métricas na barra de navegação à esquerda e selecione Todas as métricas.
3. Isso abrirá um painel com dois painéis. O painel superior mostrará o gráfico e o painel inferior apresentará as métricas que você deseja plotar. Escolha DynamoDB.
4. Selecione Métricas de tabela. Isso mostrará as tabelas em sua região atual.

5. Use a caixa de pesquisa para pesquisar o nome da tabela e escolher as métricas da operação de gravação: `ConsumedWriteCapacityUnits` e `ProvisionedWriteCapacityUnits`.

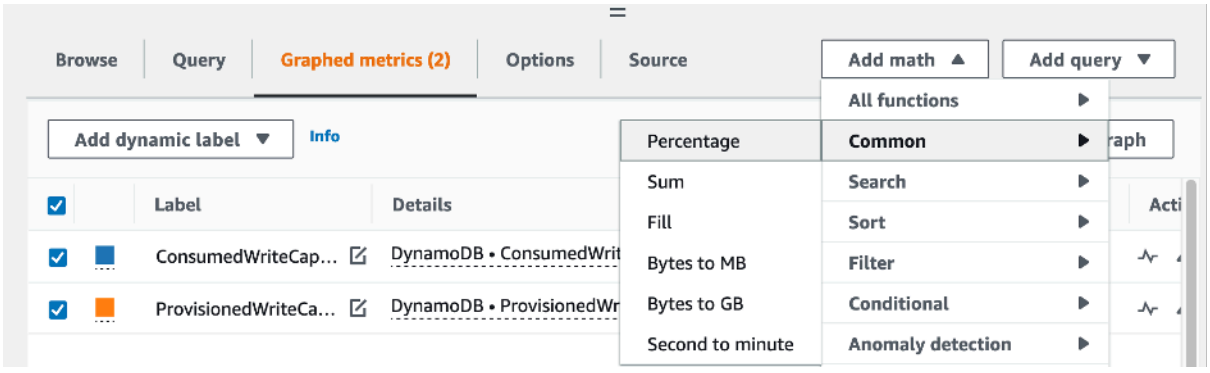
> **Note**
>
> Este exemplo fala sobre métricas da operação de gravação, mas você também pode usar essas etapas para representar graficamente as métricas da operação de leitura.

6. Selecione a guia Métricas em gráfico (2) para modificar as fórmulas. Por padrão, o CloudWatch seleciona a função estatística Média para os gráficos.

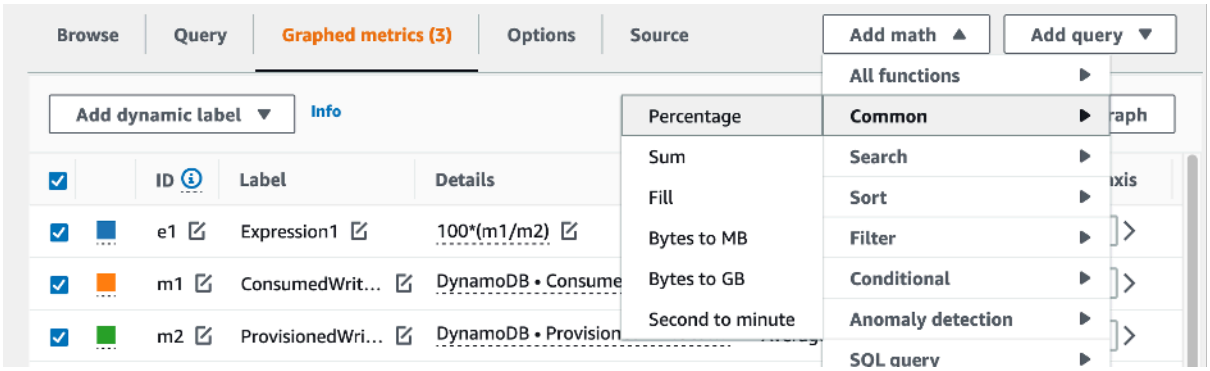


7. Com as duas métricas em gráfico selecionadas (a caixa de seleção à esquerda), selecione o menu Add math (Adicionar matemática), seguido por Common (Comum), e selecione a função Percentage (Porcentagem). Repita o procedimento duas vezes.

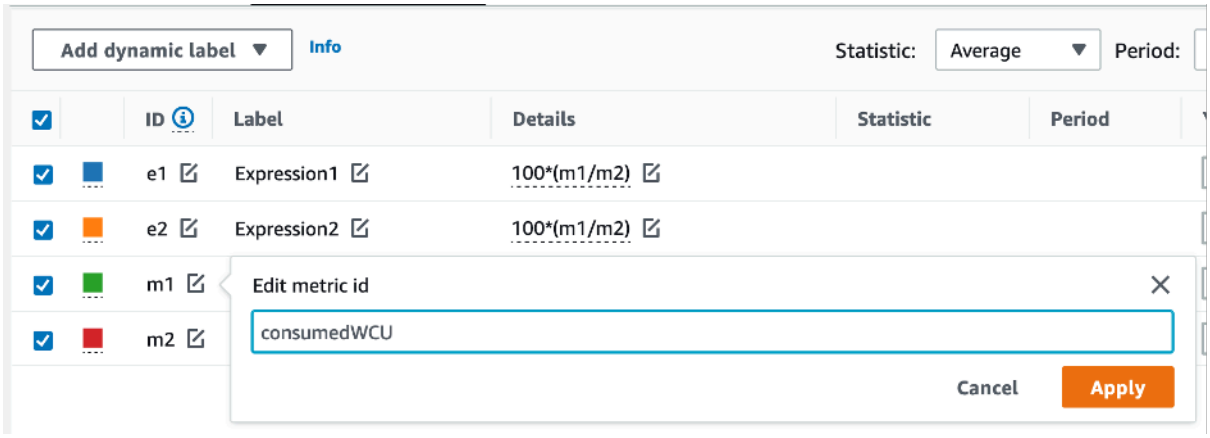
Primeira vez selecionando a função Percentage (Porcentagem):



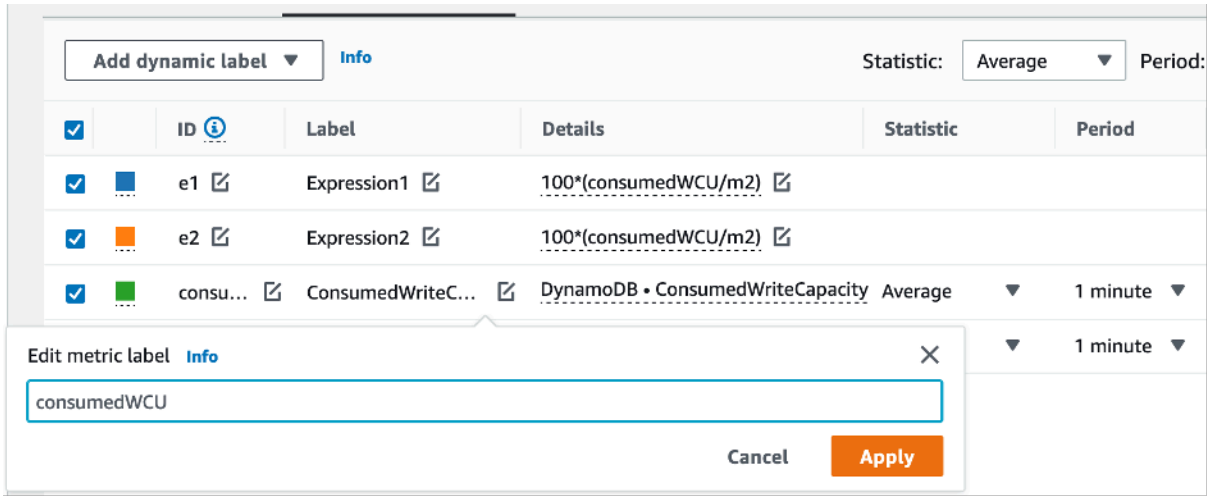
Pela segunda vez, selecionando a função Percentage (Porcentagem):



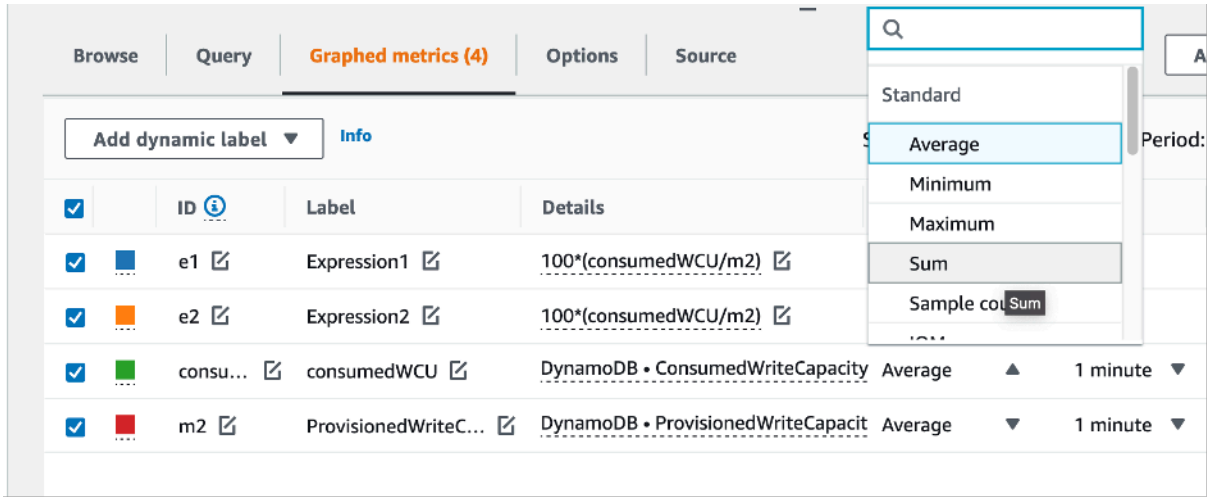
8. Nesse ponto, você deve ter quatro métricas no menu inferior. Vamos trabalhar no cálculo de `ConsumedWriteCapacityUnits`. Para sermos consistentes, precisamos combinar os nomes dos que usamos na seção da AWS CLI. Clique no ID m1 e altere esse valor para consumedWCU.

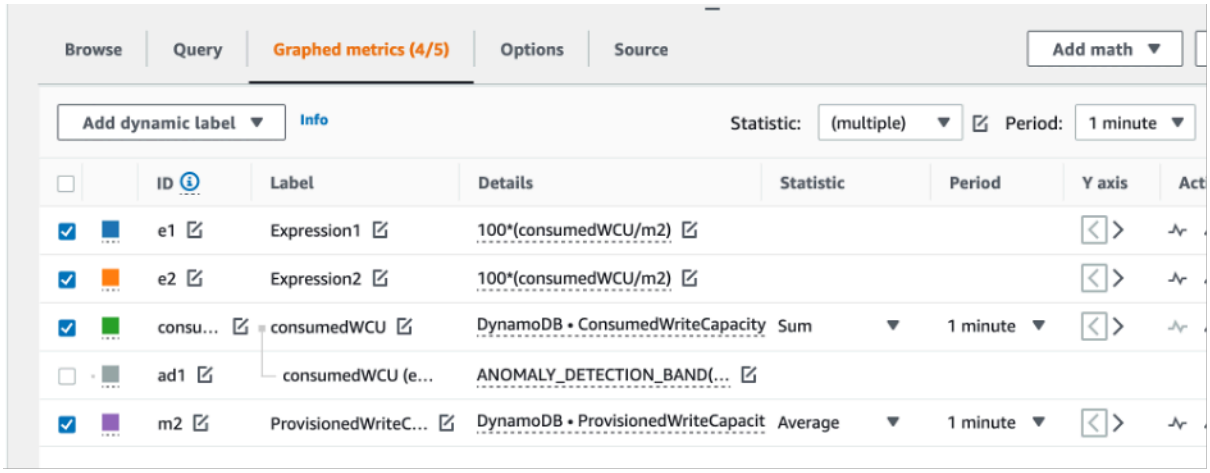


Renomeie o rótulo ConsumedWriteCapacityUnit como **`consumedWCU`**.

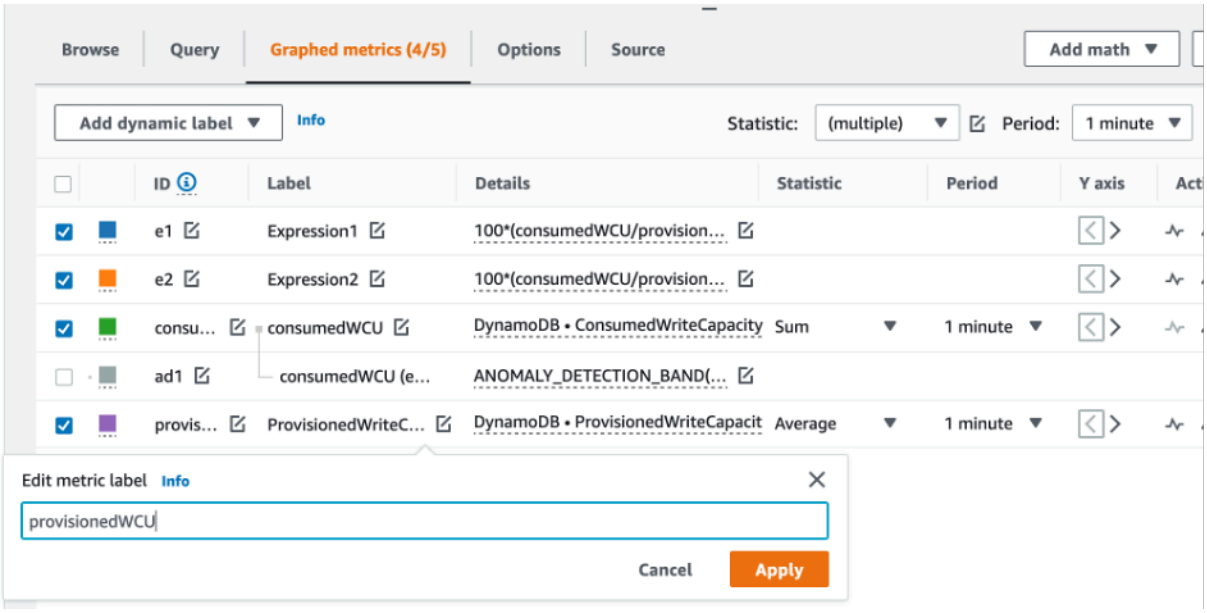


9. Altere a estatística de Average (Média) para Sum (Soma). Essa ação criará automaticamente outra métrica chamada ANOMALY_DETECTION_BAND. Para saber o escopo desse procedimento, vamos ignorá-lo removendo a caixa de seleção na métrica ad1 recém-gerada.

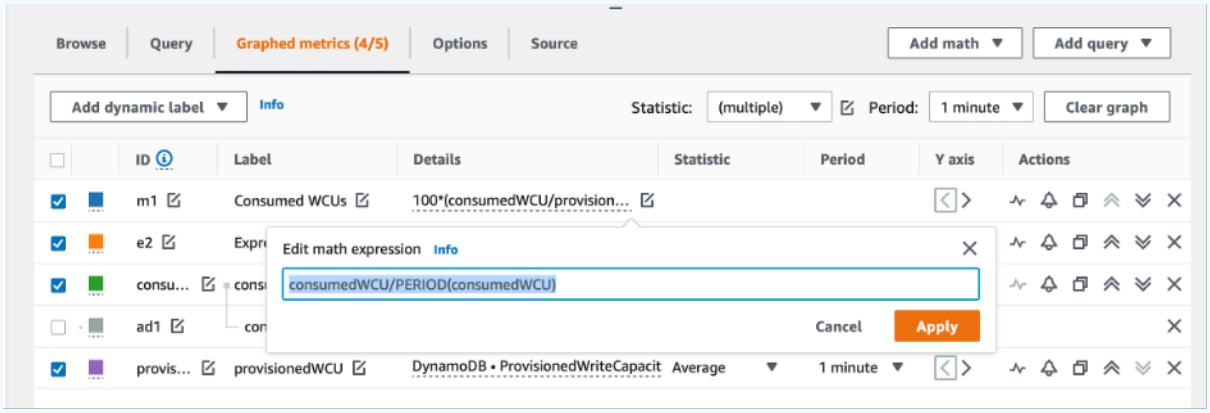




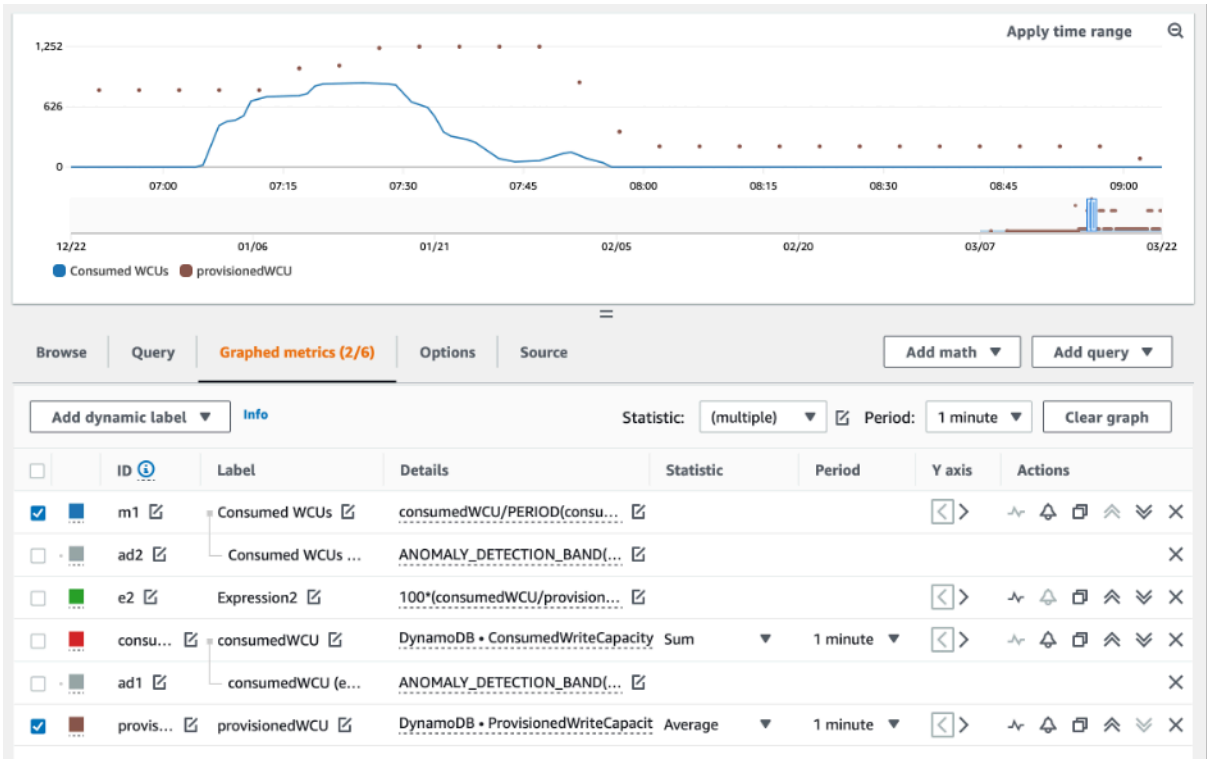
10. Repita a etapa 8 para renomear o ID m2 como provisionedWCU. Deixe a estatística definida como Average (Média).



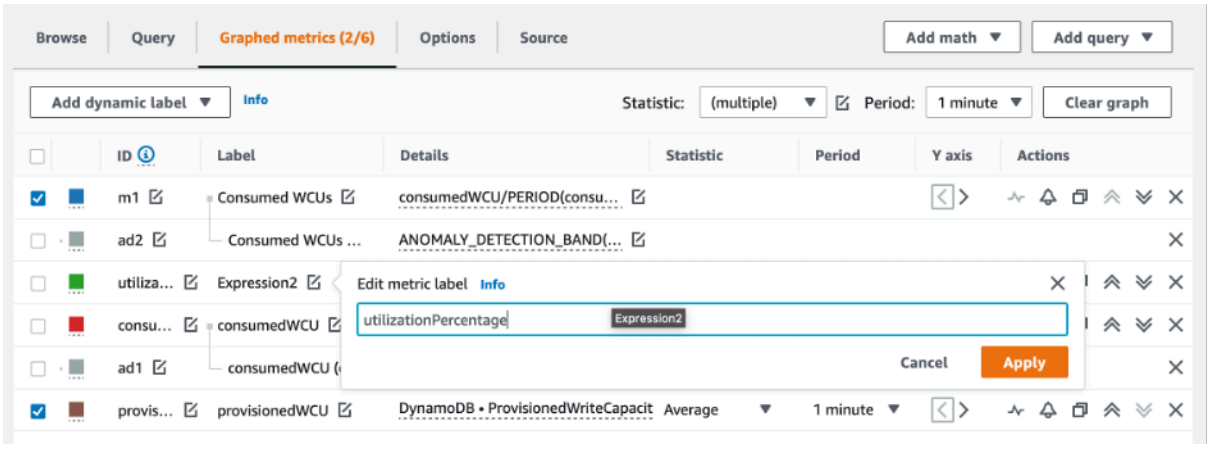
11. Selecione o rótulo Expression1 e atualize o valor para m1 e o rótulo para Consumed WCUs (WCUs consumidas).

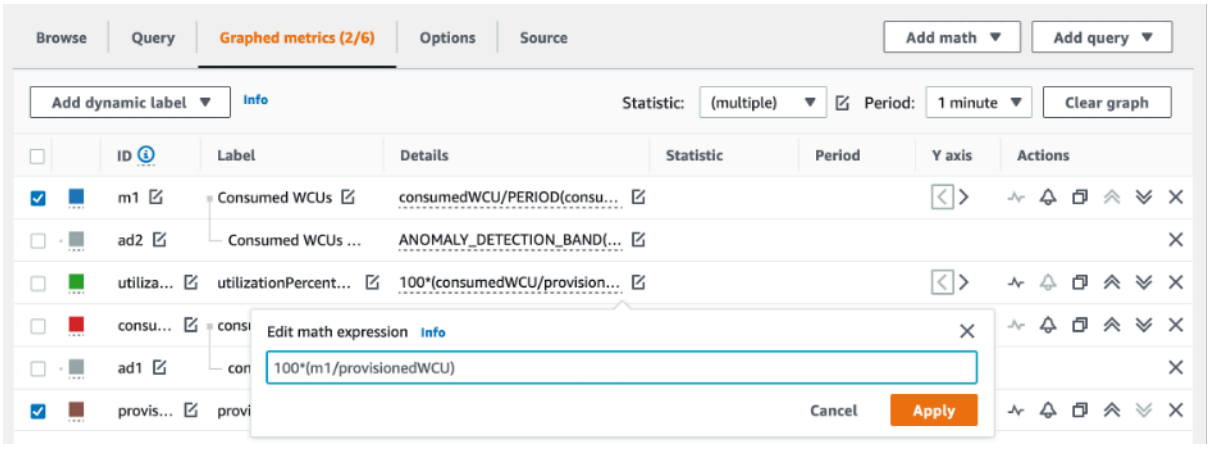
> **Note**
>
> Certifique-se de ter selecionado somente m1 (caixa de seleção à esquerda) e provisionedWCU para visualizar corretamente os dados. Atualize a fórmula clicando em Details (Detalhes) e alterando a fórmula para consumedWCU/PERIOD(consumedWCU). Essa etapa também pode gerar outra métrica ANOMALY_DETECTION_BAND, mas, para o escopo desse procedimento, podemos ignorá-la.
>
> 
>

12. Agora você deve ter dois gráficos: um que indica suas WCUs provisionadas na tabela e outro que indica as WCUs consumidas. A forma do gráfico pode ser diferente da figura abaixo, mas você pode usá-lo como referência:



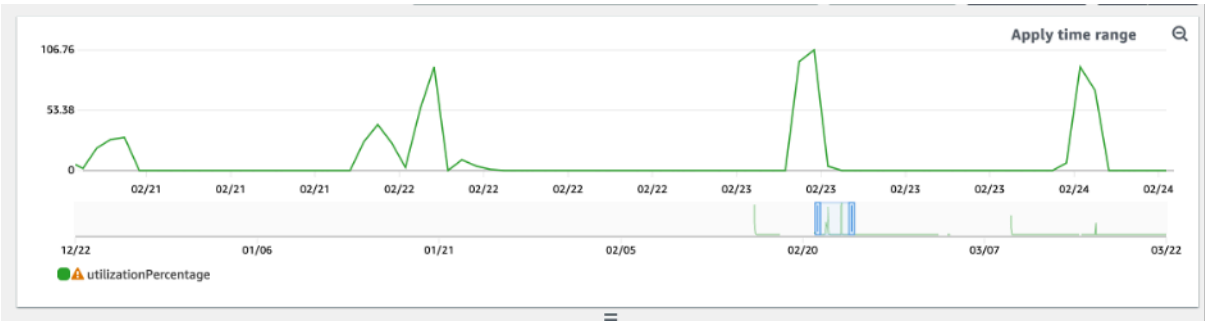
13. Atualize a fórmula de porcentagem selecionando o gráfico Expression2 (e2). Renomeie os rótulos e IDs para utilizationPercentage. Renomeie a fórmula para corresponder a 100*(m1/provisionedWCU).





14. Remova a caixa de seleção de todas as métricas, exceto utilizationPercentage, para visualizar seus padrões de utilização. O intervalo padrão é definido como um minuto, mas fique à vontade para modificá-lo conforme necessário.

Aqui está uma visão de um longo período de tempo, bem como de um período maior que uma hora. É possível ver que há alguns intervalos em que a utilização foi superior a 100%, mas essa workload específica tem intervalos mais longos com zero utilização.



Nesse momento, você pode ter resultados diferentes das imagens deste exemplo. Tudo depende dos dados da sua workload. Intervalos com mais de 100% de utilização estão sujeitos a eventos de controle de utilização. O DynamoDB oferece capacidade de expansão, mas assim que a capacidade de expansão for concluída, qualquer coisa acima de 100% terá controle de utilização.

Como identificar tabelas do DynamoDB subprovisionadas

Para a maioria das workloads, uma tabela é considerada subprovisionada quando consome constantemente mais de 80% de sua capacidade provisionada.

A capacidade de expansão é um recurso do DynamoDB que permite que os clientes consumam temporariamente mais RCUS/WCUs do que o provisionado originalmente (mais do que o throughput provisionado por segundo definido na tabela). A capacidade de expansão foi criada para absorver aumentos repentinos no tráfego devido a eventos especiais ou picos de uso. Essa capacidade de expansão não dura para sempre. Assim que as RCUs e WCUs não utilizadas forem esgotadas, você terá controle de utilização se tentar consumir mais capacidade do que a provisionada. Quando o tráfego do sua aplicação está se aproximando da taxa de utilização de 80%, o risco de controle de utilização é significativamente maior.

A regra da taxa de utilização de 80% varia com a sazonalidade de seus dados e com o crescimento do tráfego. Considere os seguintes cenários:

- Se o seu tráfego se manteve estável com uma taxa de utilização de ~90% nos últimos 12 meses, sua tabela tem a capacidade certa
- Se o tráfego da sua aplicação estiver crescendo a uma taxa de 8% ao mês em menos de 3 meses, você chegará a 100%
- Se o tráfego da sua aplicação estiver crescendo a uma taxa de 5% em pouco mais de 4 meses, você ainda chegará a 100%

Os resultados das consultas acima fornecem uma imagem da sua taxa de utilização. Use-os como um guia para avaliar melhor outras métricas que podem ajudar você a escolher aumentar a capacidade da tabela conforme necessário (por exemplo: uma taxa de crescimento mensal ou semanal). Trabalhe com sua equipe de operações para definir qual é uma boa porcentagem para sua workload e suas tabelas.

Há cenários especiais em que os dados são distorcidos quando os analisamos diariamente ou semanalmente. Por exemplo, com aplicações sazonais que têm picos de uso durante o horário comercial (mas depois caem para quase zero fora do horário comercial), é possível se beneficiar do ajuste de escala automático programado, em que você especifica as horas do dia (e os dias da semana) para aumentar a capacidade provisionada e quando reduzi-la. Em vez de buscar maior capacidade para cobrir as horas de pico, também é possível se beneficiar das configurações de ajuste de escala automático de tabelas do DynamoDB se a sua sazonalidade for menos acentuada.

> **Note**
>
> Ao criar uma configuração de Auto Scaling do DynamoDB para sua tabela-base, lembre-se de incluir outra configuração para qualquer GSI associado à tabela.

Como identificar tabelas superprovisionadas do DynamoDB

Os resultados da consulta obtidos dos scripts acima fornecem os pontos de dados necessários para realizar algumas análises iniciais. Se o seu conjunto de dados apresentar valores inferiores a 20% de utilização em vários intervalos, sua tabela pode estar superprovisionada. Para definir melhor se você precisa reduzir o número de WCUs e RCUS, revise as outras leituras nos intervalos.

Quando suas tabelas contêm vários intervalos de uso baixos, é possível se beneficiar do uso de políticas de ajuste de escala automático, seja programando o ajuste de escala automático ou simplesmente configurando as políticas de ajuste de escala automático padrão para a tabela, com base na utilização.

Se você tem uma workload com baixa utilização e alta taxa de controle de utilização (Max(ThrottleEvents)/Min(ThrottleEvents) no intervalo), isso pode acontecer quando você tem uma workload com muitos picos, em que o tráfego aumenta muito durante alguns dias (ou horas), mas, em geral, o tráfego é consistentemente baixo. Nesses cenários, pode ser benéfico usar o ajuste de escala automático programado.

O AWS Well-Architected Framework ajuda os arquitetos de nuvem a criar uma infraestrutura de alta performance que é segura, resiliente e eficiente para uma variedade de aplicações e workloads. Baseado em seis pilares (Excelência operacional, Segurança, Confiabilidade, Eficiência de performance, Otimização de custos e Sustentabilidade), o AWS Well-Architected oferece uma abordagem consistente para que clientes e parceiros avaliem arquiteturas e implementem projetos escaláveis.

O recurso Lentes do AWS Well-Architected estende a orientação oferecida pelo AWS Well-Architected para domínios setoriais e tecnológicos específicos. O recurso Lentes do Well-Architected para Amazon DynamoDB concentra-se em workloads do DynamoDB. Ele fornece práticas recomendadas, princípios de design e perguntas para avaliar e analisar uma workload do DynamoDB. A conclusão de uma análise do Lentes do Well-Architected para Amazon DynamoDB fornecerá instruções e orientações sobre os princípios de design recomendados que estão relacionados a cada um dos pilares do AWS Well-Architected. Essa orientação baseia-se em nossa experiência de trabalho com clientes em vários setores, segmentos, portes e regiões geográficas.

Como resultado direto da análise do Lentes do Well-Architected, você receberá um resumo das recomendações práticas para otimizar e melhorar workloads do DynamoDB.

Realizar a análise do Lentes do Well-Architected para Amazon DynamoDB

A análise do Lentes do Well-Architected para DynamoDB geralmente é realizada por um arquiteto de soluções da AWS com o cliente, mas também pode ser realizada pelo próprio cliente. Embora seja recomendável analisar todos os seis pilares do Well-Architected como parte do Lentes do Well-Architected para Amazon DynamoDB, você também pode optar por priorizar um ou mais pilares.

Informações e instruções adicionais para conduzir uma análise do Lentes do Well-Architected para Amazon DynamoDB estão disponíveis neste vídeo e na página do Lentes do Well-Architected para DynamoDB no GitHub.

Pilares do Lentes do Well-Architected para Amazon DynamoDB

O Lentes do Well-Architected para Amazon DynamoDB baseia-se em seis pilares:

Pilar Eficiência de performance

O pilar Eficiência de performance inclui a capacidade de usar recursos de computação de maneira eficiente para atender aos requisitos do sistema e manter essa eficiência à medida que a demanda muda e as tecnologias evoluem.

Os princípios de design mais importantes do DynamoDB para esse pilar giram em torno de modelagem dos dados, escolha de chaves de partição e chaves de classificação, e da definição de índices secundários com base nos padrões de acesso à aplicação. Outras considerações incluem escolha do modo de throughput ideal para a workload, ajuste do AWS SDK e, quando apropriado, uso de uma estratégia de armazenamento em cache ideal. Para saber mais sobre esses princípios de design, assista a este vídeo aprofundado sobre o pilar Eficiência de performance do Lentes do Well-Architected para DynamoDB.

Pilar Otimização de custos

O foco do pilar Otimização de custos é evitar custos desnecessários.

Os principais tópicos incluem entender e controlar onde o dinheiro está sendo gasto, selecionar o número mais adequado e correto de tipos de recurso, analisar os gastos ao longo do tempo, projetar modelos de dados para otimizar o custo de padrões de acesso específicos da aplicação e escalar para atender às necessidades dos negócios sem gastar demais.

Os princípios de design mais importantes de otimização de custos para o DynamoDB giram em torno de escolher o modo de capacidade e a classe de tabela mais apropriados para suas tabelas e evitar o excesso de capacidade usando o modo de capacidade sob demanda ou o modo de capacidade provisionada com escalabilidade automática. Outras considerações a serem feitas são as seguintes: a modelagem e a consulta de dados eficientes para reduzir a capacidade consumida, a reserva de partes da capacidade consumida por um preço reduzido, a redução do tamanho dos itens, a identificação e a remoção dos recursos não utilizados e o uso de TTL para excluir automaticamente dados obsoletos sem nenhum custo. Para saber mais sobre esses princípios de design, assista a este vídeo aprofundado sobre o pilar Otimização de custos do Lentes do Well-Architected para DynamoDB.

Consulte Otimização de custos para obter mais informações sobre as práticas recomendadas de otimização de custos para o DynamoDB.

Pilar Excelência operacional

O foco do pilar Excelência operacional está na execução e no monitoramento de sistemas para agregar valor empresarial e melhorar continuamente processos e procedimentos. Os principais tópicos incluem automatizar mudanças, responder a eventos e definir padrões para gerenciar operações diárias.

Os principais de design mais importantes de excelência operacional do DynamoDB incluem monitorar as métricas do DynamoDB por meio do Amazon CloudWatch e do AWS Config e alertar e corrigir automaticamente quando limites predefinidos são violados ou quando regras não compatíveis são detectadas. Outras considerações são definir os recursos do DynamoDB via infraestrutura como código e utilizar as etiquetas para organizar, identificar e contabilizar melhor os custos de seus recursos do DynamoDB. Para saber mais sobre esses princípios de design, assista a este vídeo aprofundado sobre o pilar Excelência operacional do Lentes do Well-Architected para DynamoDB.

Pilar Confiabilidade

O foco do pilar Confiabilidade é em garantir que uma workload desempenhe a função pretendida de maneira correta e consistente sempre que isso for esperado. Uma workload resiliente recupera-se rapidamente de falhas para atender a demandas da empresa e dos clientes. Os principais tópicos incluem design de sistema distribuído, planejamento de recuperação e como lidar com mudanças.

Os princípios essenciais de design de confiabilidade do DynamoDB giram em torno de escolher a estratégia de backup e a retenção com base em seus requisitos de RPO e RTO, usar tabelas globais do DynamoDB para workloads multirregionais ou cenários de recuperação de desastres entre

regiões com baixo RTO, implementar a lógica de repetição com recuo exponencial na aplicação ao configurar e usar esses recursos no AWS SDK, monitorar as métricas do DynamoDB por meio do Amazon CloudWatch e alertar e corrigir automaticamente quando limites predefinidos são violados. Para saber mais sobre esses princípios de design, assista a este vídeo aprofundado sobre o pilar Confiabilidade do Lentes do Well-Architected para DynamoDB.

Pilar Segurança

O foco do pilar Segurança está na proteção de informações e sistemas. Os principais tópicos incluem confidencialidade e integridade dos dados, identificação e gerenciamento de quem pode fazer o quê com o gerenciamento de privilégios, proteção de sistemas e estabelecimento de controles para detectar eventos de segurança.

Os principais de design mais importantes de segurança do DynamoDB são criptografar dados em trânsito com HTTPS, escolher o tipo de chave para criptografia de dados em repouso e definir políticas e perfis do IAM para autenticar, autorizar e fornecer acesso refinado aos recursos do DynamoDB. Considerações adicionais incluem a auditoria das operações do ambiente de gerenciamento e do plano de dados do DynamoDB por meio do AWS CloudTrail. Para saber mais sobre esses princípios de design, assista a este vídeo aprofundado sobre o pilar Segurança do Lentes do Well-Architected para DynamoDB.

Para obter informações de segurança adicionais para o DynamoDB, consulte Segurança.

Pilar Sustentabilidade

O foco do pilar Sustentabilidade está em minimizar os impactos ambientais da execução de workloads na nuvem. Os principais tópicos incluem um modelo de responsabilidade compartilhada pela sustentabilidade, compreensão do impacto e maximização da utilização para minimizar os recursos necessários e reduzir os impactos posteriores.

Os princípios de design mais importantes de sustentabilidade do DynamoDB incluem identificar e remover recursos não utilizados do DynamoDB, evitar o provisionamento excessivo por meio do uso do modo de capacidade sob demanda ou do modo de capacidade provisionada com autoescalabilidade, consultas eficientes para reduzir a quantidade de capacidade consumida e redução do espaço de armazenamento por meio da compactação de dados e da exclusão de dados obsoletos usando TTL. Para saber mais sobre esses princípios de design, assista a este vídeo aprofundado sobre o pilar Sustentabilidade do Lentes do Well-Architected para DynamoDB.

Práticas recomendadas para projetar e usar chaves de partição de maneira eficaz no DynamoDB

A chave primária que identifica exclusivamente cada item em uma tabela do Amazon DynamoDB pode ser simples (apenas uma chave de partição) ou composta (uma chave de partição combinada com uma chave de classificação).

É necessário projetar a aplicação para atividade uniforme em todas as chaves de partição lógica na tabela e nos respectivos índices secundários. É possível determinar os padrões de acesso que a aplicação exige e as unidades de leitura e gravação que cada tabela e índice secundário requerem.

> **Note**
>
> A capacidade adaptável aplica-se ao modo sob demanda e à capacidade provisionada.

Cada partição em uma tabela do DynamoDB foi projetada para fornecer uma capacidade máxima de três mil unidades de leitura por segundo e mil unidades de gravação por segundo. Uma unidade de leitura representa uma leitura altamente consistente por segundo, ou duas operações de leitura final consistente por segundo, para um item com até 4 KB de tamanho. Uma unidade de gravação representa uma operação de gravação por segundo para um item com até 1 KB de tamanho.

É necessário levar em conta o tamanho do item ao avaliar os limites de throughput da partição para a tabela. Por exemplo, se a tabela tiver um tamanho de item de 20 KB, uma única operação de leitura consistente consumirá 5 unidades de leitura. Isso significa que é possível realizar simultaneamente seiscentas operações de leitura consistentes por segundo nesse único item antes de atingir os limites da partição. O throughput total em todas as partições na tabela pode ser restringido pelo throughput provisionado no modo provisionado ou pelo limite de throughput em nível de tabela no modo sob demanda. Para obter mais informações, consulte Service Quotas.

Tópicos

- Projetar chaves de partição para distribuir a workload no DynamoDB
- Usar a fragmentação de gravação para distribuir workloads uniformemente em uma tabela do DynamoDB
- Distribuir a atividade de gravação de maneira eficiente durante o upload de dados no DynamoDB

Projetar chaves de partição para distribuir a workload no DynamoDB

A parte de chave de partição da chave primária de uma tabela determina as partições lógicas em que os dados de uma tabela são armazenados. Isso afeta, por sua vez, as partições físicas subjacentes. Um design de chave de partição que não distribui as solicitações de E/S de maneira eficaz pode criar partições “dinâmicas” que resultam em controle de utilização e usam a capacidade provisionada de E/S de maneira ineficiente.

O uso ideal do throughput provisionado de uma tabela depende não apenas dos padrões da workload de itens individuais, mas também do design da chave de partições. Isso não significa que você deve acessar todos os valores de chaves de partição para atingir um nível eficiente de throughput ou que a porcentagem de valores de chaves de partição acessados deve ser alta. Isso significa que quanto mais diferentes forem os valores de chave de partição que sua workload acessa, mais essas solicitações serão espalhadas entre o espaço particionado. Em geral, você utilizará o throughput provisionado de modo mais eficiente à medida que a proporção entre os valores de chave de partição acessados e o número total de valores de chave de partição aumentar.

Veja a seguir uma comparação da eficiência do throughput provisionado de alguns esquemas comuns de chave de partição.

Valor da chave de partição	Uniformidade
ID de usuário, no qual o aplicativo tem muitos usuários.	Bom
Código de status, no qual existem apenas alguns códigos de status possíveis.	Ruim
Data de criação do item, arredondada para o período mais próximo (para, por exemplo, dia, hora ou minuto).	Ruim
ID do dispositivo, no qual cada dispositivo acessa dados em intervalos relativamente semelhantes.	Bom
ID do dispositivo, no qual, mesmo que haja muitos dispositivos rastreados, um deles é muito mais popular do que todos os outros.	Ruim

Se uma única tabela tiver apenas um número pequeno de valores de chaves de partição, considere distribuir suas operações de gravação entre valores de chaves de partição mais distintos. Em outras palavras, estruture os elementos de chave primária para evitar um valor chave de partição "quente" (intensamente solicitada) que diminua a performance geral.

Por exemplo, considere uma tabela com uma chave primária composta. A chave de partição representa a data de criação do item, arredondada para o próximo dia. A chave de classificação é um identificador de item. Em um determinado dia, digamos `2014-07-09`, todos os novos itens são gravados naquele valor de chave de partição único (e na partição física correspondente).

Se a tabela se encaixar inteiramente em uma única partição (considerando o crescimento dos seus dados ao longo do tempo) e se os requisitos de throughput de leitura e gravação do seu aplicativo não excederem os recursos de leitura e gravação de uma única partição, seu aplicativo não deverá se deparar com limitações inesperadas resultantes do particionamento.

Para usar o NoSQL Workbench para DynamoDB a fim de ajudar a visualizar o design da chave de partição, consulte Criar modelos de dados com o NoSQL Workbench.

Usar a fragmentação de gravação para distribuir workloads uniformemente em uma tabela do DynamoDB

Uma forma de distribuir melhor as gravações entre o espaço de chaves de uma partição no Amazon DynamoDB é aumentar o espaço. É possível fazer isso de várias formas diferentes. Você pode adicionar um número aleatório aos valores de chaves de partição para distribuir os itens entre partições. Ou pode usar um número calculado com base em algo que você está consultando.

Fragmentação usando sufixos aleatórios

Uma estratégia para a distribuição mais uniforme de cargas em um espaço de chave de partição é adicionar um número aleatório ao final dos valores de chave de partição. Então, você randomiza as gravações no espaço maior.

Por exemplo, para uma chave de partição que represente a data de hoje, você pode escolher um número aleatório entre `1` e `200` e concatená-lo como um sufixo à data. Isso produz valores de chave de partição como `2014-07-09.1`, `2014-07-09.2` e assim por diante, até `2014-07-09.200`. Como você está randomizando a chave de partição, as gravações na tabela em cada dia são espalhadas uniformemente entre várias partições. Isso resulta em melhor paralelismo e throughput geral mais alto.

Contudo, para ler todos os itens de um determinado dia, você teria que consultar os itens quanto a todos os sufixos e mesclar os resultados. Por exemplo, você primeiro emite uma solicitação de `Query` do valor da chave de partição `2014-07-09.1`. Depois emite outra `Query` de `2014-07-09.2`, e assim por diante, por meio de `2014-07-09.200`. Por fim, seu aplicativo precisaria mesclar os resultados de todas essas solicitações `Query`.

Fragmentação usando sufixos calculados

Uma estratégia de randomização pode melhorar bastante o throughput de gravação. Mas é difícil ler um item específico, porque você não sabe que valor de sufixo foi usado na gravação do item. Para facilitar a leitura de itens individuais, você pode usar uma estratégia diferente. Em vez de usar um número aleatório para distribuir os itens entre as partições, use um número calculado com base em algo que você deseja consultar.

Considere o exemplo anterior, em que uma tabela usa a data de hoje na chave de partição. Agora considere que cada item conta com um atributo `OrderId` acessível e que você precisa com frequência de localizar os itens por ID de pedidos, além da data. Antes que o aplicativo grave o item na tabela, ele pode calcular um sufixo hash com base no ID do pedido e anexá-lo à data da chave de partição. O cálculo pode gerar um número entre 1 e 200 que é distribuído uniformemente, semelhante à estratégia aleatória.

Bastaria um cálculo simples, como o produto dos valores de pontos de código UTF-8 para os caracteres no ID do pedido, módulo 200, + 1. O valor da chave de partição seria então a data concatenada ao resultado do cálculo.

Com essa estratégia, as gravações são distribuídas uniformemente entre os valores de chaves de partição e, portanto, entre as partições físicas. É possível executar uma operação `GetItem` facilmente para um determinado item e data, pois você pode calcular o valor de chave da partição de um valor `OrderId` específico.

Para ler todos os itens de um determinado dia, você ainda precisa realizar uma operação `Query` em cada uma das chaves `2014-07-09.N` (em que `N` é de 1 a 200), e sua aplicação então deve mesclar todos os resultados. O benefício é evitar de ter um valor de chave de partição "hot" único consumindo toda a workload.

> **Note**
>
> Para obter uma estratégia mais eficiente, projetada especificamente para lidar com um alto volume de dados de séries temporais, consulte Dados de séries temporais.

Distribuir a atividade de gravação de maneira eficiente durante o upload de dados no DynamoDB

Normalmente, quando você carrega dados de outras fontes de dados, o Amazon DynamoDB particiona seus dados de tabela em vários servidores. Ao fazer upload de dados em uma tabela, você terá uma melhor performance se o upload de dados for feito em todos os servidores alocados simultaneamente.

Por exemplo, suponha que você queira fazer upload das mensagens dos usuários em uma tabela do DynamoDB que usa uma chave primária composta com `UserID` como a chave de partição e `MessageID` como a chave de classificação.

Ao carregar os dados, uma abordagem que você pode adotar é carregar todos os itens de mensagem para cada usuário, um usuário após o outro:

UserID	MessageID
U1	1
U1	2
U1	...
U1	... até 100
U2	1
U2	2
U2	...
U2	... até 200

Nesse caso, o problema é que as solicitações de gravação não estão sendo distribuídas para o DynamoDB entre seus valores de chaves de partição. Você está usando um valor de chave de partição de cada vez e fazendo upload de todos os seusitens antes de passar para o próximo valor de chave de partição e fazer o mesmo.

Nos bastidores, o DynamoDB está particionando os dados nas suas tabelas em vários servidores. Para utilizar plenamente toda a capacidade de throughput provisionado para a tabela, você precisa distribuir sua workload entre seus valores de chaves de partição. Ao direcionar uma quantidade irregular de trabalho de upload para os itens com o mesmo valor de chave de partição, você talvez não seja capaz de utilizar plenamente todos os recursos que o DynamoDB provisionou para sua tabela.

Você pode distribuir seu trabalho de upload usando a chave de classificação para carregar um item de cada valor de chave de partição, depois outro item de cada valor de chave de partição e assim por diante:

UserID	MessageID
U1	1
U2	1
U3	1
...	...
U1	2
U2	2
U3	2
...	...

Cada upload nesta sequência usa um valor de chave de partição diferente, mantendo mais servidores do DynamoDB ocupados ao mesmo tempo e melhorando sua performance de throughput.

Práticas recomendadas para usar chaves de classificação para organizar dados no DynamoDB

Em uma tabela do Amazon DynamoDB, a chave primária que identifica exclusivamente cada item na tabela pode ser composta de uma chave de partição e uma chave de classificação.

As chaves de classificação bem-projetadas tem dois principais benefícios:

- Elas reúnem informações relacionadas em um só lugar para consulta eficiente. O design cuidadoso da chave de classificação permite recuperar grupos de itens relacionados comumente necessários, usando consultas de intervalo com operadores como `begins_with`, `between`, `>`, `<` e assim por diante.
- Chaves de classificação compostas permitem definir os relacionamentos hierárquicos (um para muitos) entre os dados que podem ser consultados em qualquer nível da hierarquia.

  Por exemplo, em uma tabela que lista as localizações geográficas, você pode estruturar a chave de classificação desta forma.

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

  Isso permite fazer consultas de intervalos eficientes de uma lista de localizações em qualquer um desses níveis de agregação, de `country` a `neighborhood` e tudo o que estiver nesse intervalo.

Usar chaves de classificação para controle de versões

Muitos aplicativos precisam manter um histórico de revisões no nível do item para fins de auditoria ou conformidade e para conseguir recuperar a versão mais recente com facilidade. Há um padrão de design eficiente que faz isso por meio do uso de prefixos de chave de classificação:

- Para cada novo item, crie duas cópias do item: uma cópia deve ter um prefixo de número de versão zero (como `v0_`) no início da chave de classificação, e a outra deve ter um prefixo de número de versão um (como `v1_`).
- Sempre que o item for atualizado, use o próximo prefixo de versão mais alto na chave de classificação da versão atualizada e copie o conteúdo atualizado para o item com o prefixo de versão zero. Isso significa que a versão mais recente de qualquer item pode ser localizada facilmente usando o prefixo zero.

Por exemplo, o fabricante de uma peça pode usar um esquema como esse ilustrado abaixo.

Primary Key		Data-Item Attributes...				
Partition Key *Equipment_ID*	Sort Key *(varies)*	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...
Equipment_1	Details	Name: Biphasic Cardiometer *(equipment name)*	Factory_ID: S14_Tukwila *(factory where manufactured)*	Line_ID R_7 *(assembly-line ID)*		
	v0_Audit	Auditor: Padma *(name of the auditor)*	Latest: 3 *(most recent audit version)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	...etc.
	v1_Audit	Auditor: Rick *(name of the auditor)*	Time: 2018-03-14T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943922EKG14 *(detailed problem report in S3)*	...etc.
	v2_Audit	Auditor: George *(name of the auditor)*	Time: 2018-03-18T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943923EKG15 *(detailed problem report in S3)*	...etc.
	v3_Audit	Auditor: Padma *(name of the auditor)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	Report: x792 *(pass confirmation report)*	...etc.

O item `Equipment_1` passa por uma sequência de auditorias realizadas por vários auditores. Os resultados de cada nova auditoria são capturados em um novo item na tabela, começando com o número de versão um e depois incrementando o número para cada revisão sucessiva.

Quando cada nova revisão é adicionada, a camada do aplicativo substitui o conteúdo do item de versão zero (com chave de classificação igual a `v0_Audit`) com conteúdo da nova revisão.

Sempre que o aplicativo precisar recuperar para o status de auditoria mais recente, ele poderá consultar o prefixo da chave de classificação `v0_`.

Se o aplicativo precisar recuperar o histórico de revisão inteiro, poderá consultar todos os itens na chave de partição do item e filtrar o item `v0_`.

Esse design também funcionará para auditorias em várias peças de uma parte do equipamento, se você incluir IDs de peças individuais na chave de classificação após o prefixo de chave de classificação

Práticas recomendadas para uso de índices secundários no DynamoDB

Os índices secundários são frequentemente essenciais para suporte aos padrões de consulta que o aplicativo exige. Ao mesmo tempo, o uso em excesso os índices secundários ou o uso ineficiente deles pode adicionar custo e reduzir o desempenho desnecessariamente.

Sumário

- Diretrizes gerais para índices secundários no DynamoDB
  - Usar índices eficientemente

  - Escolher as projeções com cuidado
  - Otimizar as consultas frequentes para evitar buscas
  - Preste atenção aos limites de tamanho de coleção de itens ao criar índices secundários locais
- Aproveitar índices esparsos
  - Exemplos de índices esparsos no DynamoDB
- Usar índices secundários globais para consultas de agregação materializadas no DynamoDB
- Sobrecarga de índices secundários globais no DynamoDB
- Uso de fragmentação de gravação do índice secundário global para consultas de tabelas seletivas no DynamoDB
  - Padrão de design
  - Estratégia de fragmentação
  - Consultar o GSI fragmentado
  - Considerações sobre a execução de consultas em paralelo
  - Exemplo de código
- Usar índices secundários globais para criar uma réplica final consistente no DynamoDB

Diretrizes gerais para índices secundários no DynamoDB

O Amazon DynamoDB oferece suporte a dois tipos de índices secundários:

- Índice secundário global (GSI): um índice com uma chave de partição e uma chave de classificação que podem ser diferentes daquelas contidas na tabela base. Um índice secundário global é considerado “global” porque as consultas no índice podem abranger todos os dados em uma tabela base, além de todas as partições. Um índice secundário global não tem nenhuma limite de tamanho e tem suas próprias configurações de throughput provisionado para a atividade de leitura e gravação que são separadas dessas configurações da tabela.
- Índice secundário local (LSI): um índice com a mesma chave de partição que a tabela base mas uma chave de classificação diferente. Um índice secundário local é “local” no sentido de que cada partição de um índice secundário local tem a determinação de escopo para uma partição de tabela base com o mesmo valor de chave de partição. Como resultado, o tamanho total de itens indexados para qualquer valor de chave de partição não pode exceder 10 GB. Além disso, um índice secundário local compartilha configurações de throughput provisionado para atividade de leitura e gravação com a tabela que ele está indexando.

Cada tabela no DynamoDB pode ter até 20 índices secundários globais (cota padrão) e 5 índices secundários locais.

Os índices secundários globais geralmente são mais úteis do que os índices secundários locais. A determinação do tipo de índice a ser usado também dependerá dos requisitos da aplicação. Para ver uma comparação entre índices secundários globais e índices secundários locais e receber mais informações sobre como escolher entre eles, consulte the section called “Trabalhar com índices”.

A seguir estão alguns princípios gerais e padrões de design para serem lembrados na criação de índices no DynamoDB:

Tópicos

- Usar índices eficientemente
- Escolher as projeções com cuidado
- Otimizar as consultas frequentes para evitar buscas
- Preste atenção aos limites de tamanho de coleção de itens ao criar índices secundários locais

Usar índices eficientemente

Mantenha o número de índices no mínimo. Não crie índices secundários em atributos que você não consulta frequentemente. Os índices usados raramente contribuem para aumentar os custos de armazenamento e de E/S, sem melhorar o desempenho do aplicativo.

Escolher as projeções com cuidado

Como os índices secundários consomem armazenamento e throughput provisionado, você deve manter o índice no menor tamanho possível. Além disso, quanto menor o índice, maior será a vantagem de desempenho em comparação com a consulta de toda a tabela. Se as suas consultas geralmente retornarem apenas um pequeno subconjunto de atributos, e o tamanho total dos atributos for muito menor que o item inteiro, projete apenas os atributos que você solicita regularmente.

Se você estima muitas atividades de gravação em uma tabela, em comparação com as atividades de leitura, siga estas práticas recomendadas:

- Considere projetar menos atributos para reduzir o tamanho dos itens gravados no índice. Contudo, isso se aplicará somente se o tamanho dos atributos projetados for maior do que uma única unidade de capacidade de gravação (1 KB). Por exemplo, se o tamanho de uma entrada de índice

for apenas 200 bytes, o DynamoDB o arredondará para 1 KB. Em outras palavras, contanto que os itens de índice sejam pequenos, você pode projetar mais atributos sem nenhum custo extra.

- Evite projetar os atributos que serão raramente necessários nas consultas. Sempre que você atualizar um atributo projetado em um índice, haverá um custo extra também para a atualização do índice. Você pode ainda recuperar atributos não projetados em uma `Query` a um custo maior de throughput provisionado, mas o custo da consulta pode ser significativamente menor do que o custo da atualização frequente do índice.
- Especifique `ALL` somente se você deseja que suas consultas retornem itens da tabela inteira classificados por uma chave de classificação diferente. Proteger todos os atributos elimina a necessidade de buscas da tabela, mas, na maioria dos casos, isso dobra os custos das atividades de armazenamento e gravação.

Equilibre a necessidade de manter seus índices o menor possível e a necessidade de manter o mínimo de buscas, como explicado a próxima seção.

Otimizar as consultas frequentes para evitar buscas

Para obter consultas mais rápidas com a menor latência possível, projete todos os atributos que você espera que essas consultas retornem. Especificamente, se você consultar um índice secundário local quanto a atributos que não estão planejados, o DynamoDB buscará automaticamente esses atributos na tabela, o que requer a leitura do item inteiro na tabela. Isso pode gerar operações adicionais de E/S e latência que você pode evitar.

Lembre-se de que as consultas “ocasionais” podem se transformar em consultas “principais”. Se houver mais atributos que não se pretende projetar, porque você estima que as consultas serão apenas ocasionais, considere se as condições podem ser alteradas e se você pode lamentar não ter projetado os atributos.

Para obter mais informações sobre buscas de tabela, consulte Considerações sobre throughput provisionado para índices secundários locais.

Preste atenção aos limites de tamanho de coleção de itens ao criar índices secundários locais

Uma coleção de itens compreende todos os itens em uma tabela e seus índices secundários locais que têm a mesma chave de partição. Nenhuma coleção de itens pode exceder 10 GB. Por isso, é possível que o espaço se esgote para um determinado valor de chave de partição.

Quando você adiciona ou atualiza um item de tabela, o DynamoDB atualiza todos os índices secundários locais que são afetados. Se os atributos indexados forem definidos na tabela, os índices secundários locais também crescerão.

Ao criar um índice secundário local, pense no volume de dados que serão gravados nele e quantos desses itens de dados terão o mesmo valor de chave de partição. Se você espera que a soma dos itens da tabela e do índice de um determinado valor de chave de partição exceda 10 GB, considere se é possível evitar a criação do índice.

Se você não puder evitar a criação do índice secundário local, deverá prever o limite de tamanho da coleção de itens e executar ações antes que ele seja excedido. Como prática recomendada, você deve utilizar o parâmetro `ReturnItemCollectionMetrics` ao gravar itens para monitorar e alertar sobre tamanhos de conjuntos de itens que se aproximam do limite de 10 GB. Se o tamanho máximo do conjunto de itens for excedido, as tentativas de gravação serão malsucedidas. É possível reduzir os problemas de tamanho do conjunto de itens monitorando e emitindo alertas sobre os tamanhos antes que eles afetem a aplicação.

> **Note**
>
> Não é possível excluir um índice secundário local depois que ele é criado.

Para estratégias sobre como trabalhar dentro do limite e tomar uma ação corretiva, consulte Limite de tamanho de conjunto de itens.

Aproveitar índices esparsos

Para qualquer item em uma tabela, o DynamoDB grava uma entrada de índice correspondente somente se o valor da chave de classificação do índice está presente no item. Se a chave de classificação não aparecer em todos os itens da tabela ou se a chave de partição do índice não estiver presente no item, o índice será considerado esparso.

Os índices esparsos são úteis para consultas em uma subseção pequena de uma tabela. Por exemplo, suponha que você tem uma tabela que armazene todos os pedidos de clientes, com os seguintes atributos chaves:

- Chave de partição: `CustomerId`
- Chave de classificação: `OrderId`

Para acompanhar os pedidos em aberto, você poderá inserir um atributo chamado `isOpen` aos itens de pedidos que ainda não foram enviados. Então, quando o pedido for enviado, você poderá excluir o atributo. Se você criar um índice em `CustomerId` (chave de partição) e em `isOpen` (chave de cada), somente os pedidos definidos com `isOpen` serão exibidos. Quando você tem milhares de pedidos e apenas um pequeno número deles está em aberto, é mais rápido e barato consultar o índice de pedidos em aberto do que analisar a tabela inteira.

Em vez de usar um tipo booleano atributo como `isOpen`, você poderia usar um atributo com um valor que resulta em um pedido de classificação útil no índice. Por exemplo, você pode usar um atributo `OrderOpenDate` definido como a data em que cada pedido foi feito e excluí-lo após o atendimento do pedido. Dessa forma, quando você consulta o índice esparso, os itens são retornados classificados de acordo com a data na qual cada pedido foi feito.

Exemplos de índices esparsos no DynamoDB

Os índices secundários globais são esparsos por padrão. Ao criar um índice secundário global, você especifica uma chave de partição e, opcionalmente, uma chave de classificação. Somente os itens na tabela base que contenham esses atributos aparecem no índice.

Ao projetar um índice secundário global como esparso, você pode provisioná-lo com um throughput de gravação menor do que a da tabela base, ainda obtendo um excelente desempenho.

Por exemplo, um aplicativo de jogos pode acompanhar todas as pontuações de cada usuário, mas geralmente só precisa consultar algumas pontuações altas. O seguinte design lida com esse cenário de modo eficiente:

Table	Primary Key		Data Attributes...		
	Partition Key	*Sort Key*			
	Player_ID	Game_ID	Attribute 1	Attribute 2	Attribute 3
	Rick	Game_1	Score: 36,750 *(game score)*	Date: 2017-11-14 *(date of game)*	
		Game_2	Score: 69,450 *(game score)*	Date: 2017-12-31 *(date of game)*	
		Game_3	Score: 135,900 *(game score)*	Date: 2018-01-19 *(date of game)*	Award: Champ *(type of award)*
	Padma	Game_4	Score: 25,350 *(game score)*	Date: 2018-01-27 *(date of game)*	
		Game_5	Score: 69,450 *(game score)*	Date: 2028-01-19 *(date of game)*	
		Game_6	Score: 147,300 *(game score)*	Date: 2018-02-02 *(date of game)*	Award: Champ *(type of award)*
		Game_7	Score: 169,100 *(game score)*	Date: 2018-03-10 *(date of game)*	Award: Champ *(type of award)*

Aqui, Rick jogou três jogos e obteve o status `Champ` em um deles. Padma jogou quatro jogos e obteve o status `Champ` em dois deles. Observe que o atributo `Award` está presente apenas nos itens em que o usuário obteve um award. O índice secundário global associado é semelhante ao seguinte:

GSI	Primary Key	Projected Attributes...			
	Partition Key				
	Award	Player_ID	Game_ID	Score	Date
	Champ	Rick	Game_3	135,900	2018-01-19
		Padma	Game_6	147,300	2018-02-02
		Padma	Game_7	169,100	2018-03-10

O índice secundário global contém apenas as pontuações altas que são consultadas com frequência e representadas por um pequeno subconjunto de itens na tabela base.

Usar índices secundários globais para consultas de agregação materializadas no DynamoDB

Manter as agregações quase em tempo real e as métricas de chaves relacionadas aos dados em constante mudança é uma operação cada vez mais valiosa para a rápida tomada de decisões na

empresa. Por exemplo, uma biblioteca de músicas pode querer apresentar suas músicas que mais foram obtidas por download quase em tempo real.

Considere o seguinte layout de tabela da biblioteca de músicas:

Music Library Table

Primary Key		Data-Item Attributes...			
Partition Key	Sort Key	Attribute 1	Attribute 2	Attribute 3	
Song-129 *(song ID)*	Details	Title: Wild Love *(song title)*	Artist: Argyboots *(artist or band name)*	Downloads: 15,314,822 *(lifetime total downloads)*	...etc.
	Month-2018-01	GSI Primary Key — Month: 2018-01 *(download month)*	GSI Secondary Key — MonthTotal: 1,746,992 *(month total downloads)*		
	Dld-9349823681	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823682	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823683	Time: 2018-01-01T00:00:07 *(download timestamp)*			

A tabela neste exemplo armazena músicas com `songID` como a chave de partição. Você pode habilitar Amazon DynamoDB Streams nessa tabela e conectar uma função do Lambda aos fluxos de modo que, à medida que o download de cada música for feito, uma entrada seja adicionada à tabela com `Partition-Key=SongID` e `Sort-Key=DownloadID`. Conforme essas atualizações são feitas, elas ativam uma função do Lambda nos DynamoDB Streams. A função do Lambda pode agregar e agrupar os downloads por `songID` e atualizar o item de nível superior, `Partition-Key=songID` e `Sort-Key=Month`. Tenha em mente que se uma execução do Lambda falhar logo após escrever o novo valor agregado, ela pode ser repetida e agregar o valor mais de uma vez, deixando você com um valor aproximado.

Para ler as atualizações quase em tempo real, com latência de milissegundo de um único dígito, use o índice secundário global com condições de consulta `Month=2018-01`, `ScanIndexForward=False`, `Limit=1`.

Outra otimização de chave usada aqui é que o índice secundário global é um índice esparso e está disponível somente nos itens que precisam ser consultados para recuperar os dados em tempo real. O índice secundário global pode atender aso fluxos de trabalho adicionais que precisam de informações sobre as 10 primeiras músicas que eras populares, ou qualquer música transferida naquele mês.

Sobrecarga de índices secundários globais no DynamoDB

Embora o Amazon DynamoDB tenha uma cota padrão de 20 índices secundários globais por tabela, na prática é possível indexar muito mais do que 20 campos de dados. Ao contrário de uma tabela em um sistema de banco de dados relacional (RDBMS) em que o esquema é uniforme, uma tabela no DynamoDB pode guardar muitos tipos diferentes de itens de dados ao mesmo tempo. Além disso, os mesmos atributos em diferentes itens podem conter tipos de informações totalmente diferentes.

Considere o seguinte exemplo de um layout de tabela do DynamoDB que salva vários tipos diferentes de dados.

Primary Key		Data-Item Attributes...		
Partition Key	Sort Key	Attribute 1	Attribute 2	...
HR-974 *(employee ID)*	Employee_Name	**Data:** Murphy, John *(employee name)*	**Start:** 2008-11-08 *(start date)*	...etc.
	YYYY-Q1	**Data:** $5,477 *(order totals in USD)*	**Name:** Murphy, John *(employee name)*	
	HR_confidential	**Data:** 2008-11-08 *(hire date)*	**Name:** Murphy, John *(employee name)*	...etc.
	Warehouse_01	**Data:** Murphy, John *(employee name)*		
	v0_Job_title	**Data:** Operator-1 *(job title)*	**Start:** 2008-11-08 *(start date)*	...etc.
	v1_Job_title	**Data:** Operator-2 *(job title)*	**Start:** 2016-11-04 *(start date)*	...etc.
	v2_Job_title	**Data:** Supervisor-1 *(job title)*	**Start:** 2017-11-01 *(start date)*	...etc.

O atributo `Data`, que é comum para todos os itens, tem conteúdo diferente dependendo do item pai. Se você criar um índice secundário global para a tabela que usa a chave de classificação da tabela como sua chave de partição e o atributo `Data` como sua chave de classificação, poderá fazer

uma variedade de consultas diferentes usando esse único índice secundário global. Essas consultas podem incluir o seguinte:

- Procure um funcionário por nome no índice secundário global usando `Employee_Name` como o valor da chave de partição e o nome do funcionário (por exemplo, `Murphy, John`) como valor de chave de classificação.
- Use o índice secundário global para encontrar todos os funcionários que trabalham em um determinado armazém procurando um ID de armazém (como `Warehouse_01`).
- Obtenha uma lista de contratações recentes consultando o índice secundário global em `HR_confidential` como um valor de chave de partição e usando um intervalo de datas como o valor da chave de classificação.

Uso de fragmentação de gravação do índice secundário global para consultas de tabelas seletivas no DynamoDB

Quando você precisa consultar dados recentes em um prazo específico, a exigência do DynamoDB de fornecer uma chave de partição para a maioria das operações de leitura pode ser um desafio. Para abordar esse cenário, você pode implementar um padrão de consulta eficaz usando uma combinação de fragmentação de gravação e um índice secundário global (GSI).

Essa abordagem permite que você recupere e analise dados sensíveis ao tempo de forma eficiente sem realizar verificações completas da tabela, o que pode consumir muitos recursos e ser caro. Ao projetar estrategicamente a estrutura e a indexação da tabela, é possível criar uma solução flexível que seja compatível com a recuperação de dados baseada no tempo, mantendo o desempenho ideal.

Tópicos

- Padrão de design
- Estratégia de fragmentação
- Consultar o GSI fragmentado
- Considerações sobre a execução de consultas em paralelo
- Exemplo de código

Padrão de design

Ao trabalhar com o DynamoDB, é possível superar os desafios de recuperação de dados baseada no tempo implementando um padrão sofisticado que combina fragmentação de gravação e índices secundários globais para permitir consultas flexíveis e eficientes em janelas de dados recentes.

Estrutura da tabela

- Chave de partição (PK): “Username”

Estrutura do GSI

- Chave de partição do GSI (PK_GSI): “ShardNumber#”
- Chave de classificação do GSI (SK_GSI): carimbo de data/hora em ISO 8601 (por exemplo, “2030-04-01T12:00:00Z”)

[TABLE] - TimeBounded Metadata Actions

PK	SK_GSI	Address	PK_GSI
Trenton69	2023-12-05T06:40:31.312Z	117 Hudson Divide	ShardNumber#0
Scot_Langworth-Prosacco	2024-10-09T14:44:45.819Z	84021 Herzog Skyway	ShardNumber#6
Juliet.Morissette	2025-03-28T07:20:56.538Z	4871 N Broadway	ShardNumber#4
Kay.Von71	2024-11-23T16:19:26.763Z	14554 Odell Throughway	ShardNumber#2
Kiarra43	2023-11-15T16:21:57.078Z	8471 London Road	ShardNumber#1
Marcella91	2024-12-17T09:02:31.623Z	10271 Nick Crescent	ShardNumber#6
Bernard.Lemke36	2025-09-09T00:18:34.482Z	5438 Douglas Groves	ShardNumber#2
Daisha83	2024-01-24T06:38:08.482Z	8786 Armstrong Radial	ShardNumber#3
Marcos_Schmitt58	2024-04-06T17:38:58.551Z	510 S Center Street	ShardNumber#3
Jeremie_VonRueden-Mills	2025-04-06T04:24:23.701Z	157 Koch Drives	ShardNumber#1
Verna28	2025-10-10T18:42:21.059Z	70040 Baylee Trafficway	ShardNumber#1
Trycia.Doyle	2024-01-10T17:00:08.360Z	9511 Tillman Extensions	ShardNumber#2

Estratégia de fragmentação

Supondo que você decida usar 10 fragmentos, os números de fragmentos podem variar de 0 a 9. Ao registrar uma atividade, você calcularia o número do fragmento (por exemplo, usando uma função

hash no ID do usuário e, depois, pegando o módulo do número de fragmentos) e o adicionaria como prefixo à chave de partição do GSI. Esse método distribui as entradas em diferentes fragmentos, mitigando o risco de partições sobrecarregadas.

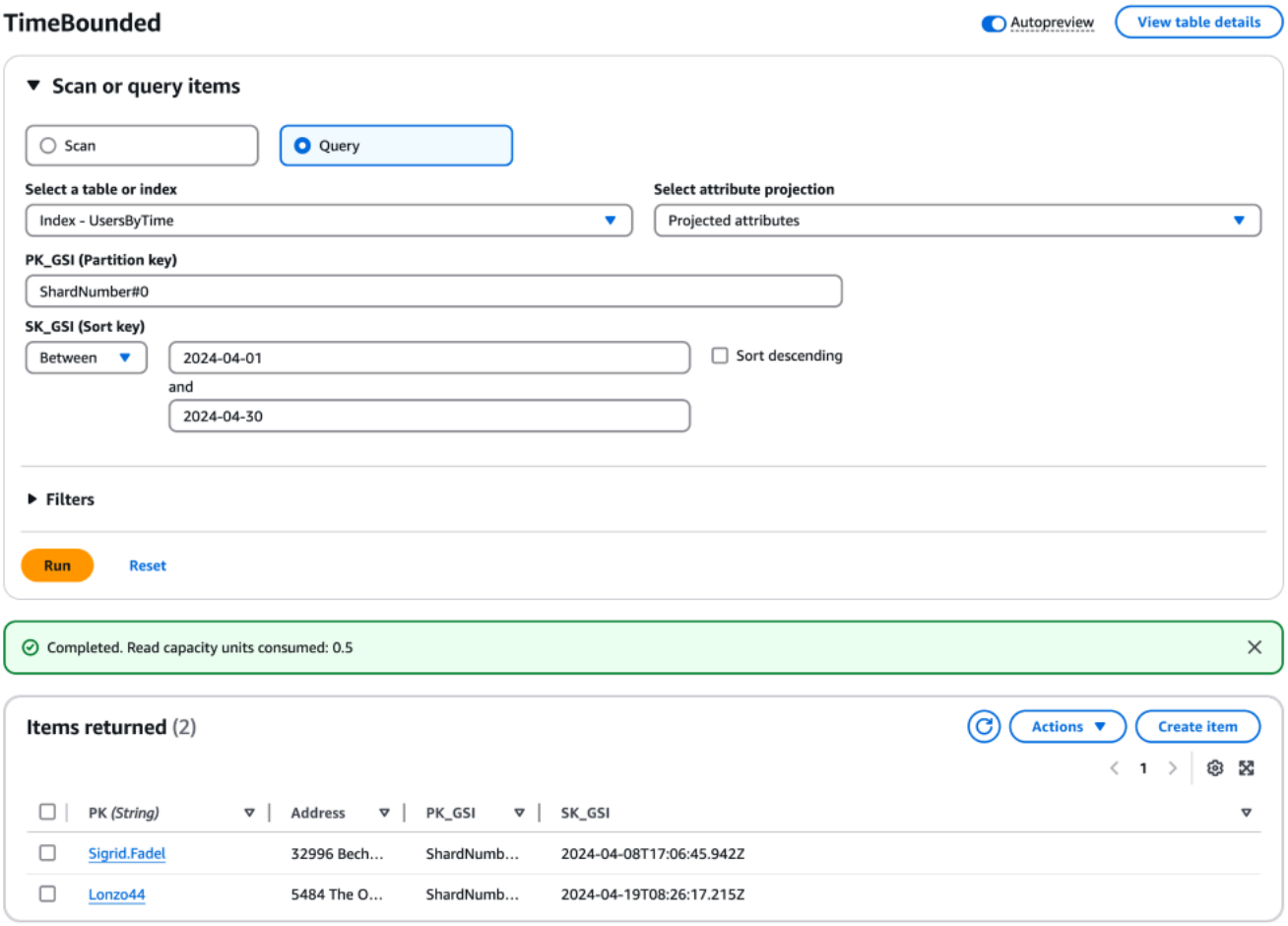
Consultar o GSI fragmentado

Consultar itens dentro de determinado intervalo de tempo em todos os fragmentos em uma tabela do DynamoDB, na qual os dados são fragmentados em várias chaves de partição, requer uma abordagem diferente da consulta de uma única partição. Como as consultas do DynamoDB estão limitadas a uma única chave de partição por vez, não é possível consultar diretamente vários fragmentos com uma única operação de consulta. No entanto, você pode obter o resultado desejado por meio da lógica no nível da aplicação executando várias consultas, cada uma visando um fragmento específico e, depois, agregando os resultados. O procedimento a seguir explica como fazer isso.

Como consultar e agregar fragmentos

1. Identifique o intervalo de números de fragmentos usado na estratégia de fragmentação. Por exemplo, se você tiver 10 fragmentos, os números de fragmentos variarão de 0 a 9.
2. Para cada fragmento, crie e execute uma consulta para buscar itens dentro do intervalo de tempo desejado. Essas consultas podem ser executadas em paralelo para melhorar a eficiência. Use a chave de partição com o número do fragmento e a chave de classificação com o intervalo de tempo para essas consultas. Aqui está um exemplo de consulta para um único fragmento:

```
aws dynamodb query \
    --table-name "YourTableName" \
    --index-name "YourIndexName" \
    --key-condition-expression "PK_GSI = :pk_val AND SK_GSI BETWEEN :start_date
 AND :end_date" \
    --expression-attribute-values '{
        ":pk_val": {"S": "ShardNumber#0"},
        ":start_date": {"S": "2024-04-01"},
        ":end_date": {"S": "2024-04-30"}
    }'
```



Você replicaria essa consulta para cada fragmento, ajustando a chave de partição adequadamente (por exemplo, “ShardNumber#1”, “ShardNumber#2”, …, “ShardNumber#9”).

3. Agregue os resultados de cada consulta após a conclusão de todas elas. Execute essa agregação no código da aplicação, combinando os resultados em um único conjunto de dados que representa os itens de todos os fragmentos dentro do intervalo de tempo especificado.

Considerações sobre a execução de consultas em paralelo

Cada consulta consome a capacidade de leitura da tabela ou do índice. Se você estiver usando throughput provisionado, a tabela deverá ser provisionada com capacidade suficiente para lidar com o pico de consultas paralelas. Se você estiver usando capacidade sob demanda, atente-se às possíveis implicações de custo.

Exemplo de código

Para executar consultas paralelas entre fragmentos no DynamoDB usando Python, é possível usar a biblioteca boto3, que é o SDK da Amazon Web Services para Python. Esse exemplo pressupõe que você tenha o boto3 instalado e configurado com as credenciais da AWS apropriadas.

O código Python a seguir demonstra como realizar consultas paralelas em vários fragmentos para determinado intervalo de tempo. Ele usa concurrent futures para executar consultas em paralelo, reduzindo o tempo geral de execução em comparação com a execução sequencial.

```
import boto3
from concurrent.futures import ThreadPoolExecutor, as_completed

# Initialize a DynamoDB client
dynamodb = boto3.client('dynamodb')

# Define your table name and the total number of shards
table_name = 'YourTableName'
total_shards = 10  # Example: 10 shards numbered 0 to 9
time_start = "2030-03-15T09:00:00Z"
time_end = "2030-03-15T10:00:00Z"

def query_shard(shard_number):
    """
    Query items in a specific shard for the given time range.
    """
    response = dynamodb.query(
        TableName=table_name,
        IndexName='YourGSIName',  # Replace with your GSI name
        KeyConditionExpression="PK_GSI = :pk_val AND SK_GSI BETWEEN :date_start
 AND :date_end",
        ExpressionAttributeValues={
            ":pk_val": {"S": f"ShardNumber#{shard_number}"},
            ":date_start": {"S": time_start},
            ":date_end": {"S": time_end},
        }
    )
    return response['Items']

# Use ThreadPoolExecutor to query across shards in parallel
with ThreadPoolExecutor(max_workers=total_shards) as executor:
    # Submit a future for each shard query
```

```
    futures = {executor.submit(query_shard, shard_number): shard_number for
 shard_number in range(total_shards)}

    # Collect and aggregate results from all shards
    all_items = []
    for future in as_completed(futures):
        shard_number = futures[future]
        try:
            shard_items = future.result()
            all_items.extend(shard_items)
            print(f"Shard {shard_number} returned {len(shard_items)} items")
        except Exception as exc:
            print(f"Shard {shard_number} generated an exception: {exc}")

# Process the aggregated results (e.g., sorting, filtering) as needed
# For example, simply printing the count of all retrieved items
print(f"Total items retrieved from all shards: {len(all_items)}")
```

Antes de executar esse código, substitua `YourTableName` e `YourGSIName` pelos nomes reais da tabela e do GSI da configuração do DynamoDB. Além disso, ajuste as variáveis `total_shards`, `time_start` e `time_end` de acordo com seus requisitos específicos.

Esse script consulta cada fragmento em busca de itens dentro do intervalo de tempo especificado e agrega os resultados.

Usar índices secundários globais para criar uma réplica final consistente no DynamoDB

Você pode usar um índice secundário global para criar uma réplica com consistência final de uma tabela. A criação de uma réplica pode permitir a você:

- Definir diferentes capacidades de leitura provisionadas para diferentes leitores. Por exemplo, suponha que você tenha duas aplicações: uma lida com consultas de alta prioridade e precisa dos mais altos níveis de performance de leitura, enquanto a outra lida com consultas de baixa prioridade que podem tolerar limitações nas atividades de leitura.

  Se ambas as aplicações forem lidas da mesma tabela, uma carga de leitura intensa da aplicação de baixa prioridade poderá consumir toda a capacidade de leitura disponível para a tabela. Isso aceleraria a atividade de leitura da aplicação de alta prioridade.

Em vez disso, é possível criar uma réplica por meio de um índice secundário global cuja capacidade de leitura você pode definir separadamente da própria tabela. Em seguida, você pode fazer com que sua aplicação de baixa prioridade consulte a réplica em vez da tabela.

- Eliminar totalmente as leituras de uma tabela. Por exemplo, talvez você tenha uma aplicação que capta um alto volume de registros de cliques de um site e não queira arriscar que as leituras interfiram nisso. Você pode isolar esta tabela e impedir leituras por outras aplicações (consulte Uso de condições de política do IAM para controle de acesso refinado) ao mesmo tempo que permite que outras aplicações leiam uma réplica criada usando um índice secundário global.

Para criar uma réplica, configure um índice secundário global que tenha o mesmo esquema de chaves que a tabela principal, com alguns (ou todos) os atributos não chaves projetados. Nas aplicações, é possível direcionar parte ou toda a atividade de leitura para esse índice secundário global, em vez de para a tabela principal. Em seguida, você pode ajustar a capacidade de leitura provisionada do índice secundário global para lidar com essas leituras sem alterar a capacidade de leitura provisionada da tabela principal.

Há sempre um breve atraso de propagação entre uma gravação na tabela principal e o momento em que os dados gravados surgem no índice. Em outras palavras, suas aplicações devem levar em conta que a réplica de índice secundário global é final consistente somente com a tabela principal.

Você pode criar várias réplicas do índice secundário global para oferecer suporte a diferentes padrões de leitura. Ao criar as réplicas, projete somente os atributos que cada padrão de leitura realmente requer. Uma aplicação pode então consumir menos capacidade de leitura provisionada para obter apenas os dados necessários, em vez de precisar ler o item da tabela principal. Essa otimização pode resultar em uma economia significativa com o passar do tempo.

Práticas recomendadas para armazenar itens e atributos grandes no DynamoDB

O Amazon DynamoDB limita o tamanho de cada item que pode ser armazenado em uma tabela a 400 KB (consulte Cotas no Amazon DynamoDB). Se a aplicação precisa armazenar mais dados em um item do que o limite de tamanho permitido pelo DynamoDB, você poderá tentar compactar um ou mais atributos grandes ou dividir o item em vários itens (indexados devidamente por chaves de classificação). Você pode armazenar o item como um objeto no Amazon Simple Storage Service (Amazon S3) e armazenar o identificador do objeto do Amazon S3 no item do DynamoDB.

Como prática recomendada, é necessário utilizar o parâmetro `ReturnConsumedCapacity` ao gravar itens para monitorar e alertar sobre tamanhos de itens que se aproximam do limite máximo de 400 KB. Se o tamanho máximo de itens for excedido, as tentativas de gravação serão malsucedidas. O DynamoDB exibirá um erro ValidationException. É possível reduzir os problemas de tamanho de item monitorando e emitindo alertas sobre os tamanhos antes que eles afetem a aplicação.

Compactar valores de atributos grandes

A compactação de valores de atributos grandes pode permitir que eles se ajustem aos limites do item no DynamoDB e reduzam os custos de armazenamento. Os algoritmos de compactação, como GZIP ou LZO, geram resultado binário que, depois, você pode armazenar em um tipo de atributo `Binary` dentro do item.

Por exemplo, pense em uma tabela que armazene mensagens gravadas por usuários do fórum. Essas mensagens geralmente contêm longas strings de texto, que são candidatas à compactação. Embora a compactação possa reduzir o tamanho dos itens, a desvantagem é que os valores dos atributos compactados não são úteis para filtragem.

Para obter um código de exemplo que demonstra como compactar essas mensagens no DynamoDB, consulte:

- Exemplo: tratar atributos do tipo binário usando a API de documento do AWS SDK para Java
- Exemplo: tratar atributos do tipo binário usando a API de baixo nível do AWS SDK para .NET

Particionamento vertical

Uma solução alternativa para lidar com itens grandes é dividi-los em blocos menores de dados e associar todos os itens relevantes pelo valor da chave de partição. Depois, é possível usar uma string de chave de classificação para identificar as informações associadas armazenadas ao lado dela. Ao fazer isso e ter vários itens agrupados pelo mesmo valor de chave de partição, você está criando um conjunto de itens.

Para ter mais informações sobre essa abordagem, consulte:

- Use vertical partitioning to scale data efficiently in Amazon DynamoDB
- Implement vertical partitioning in Amazon DynamoDB using AWS Glue

Armazenar valores de atributos grandes no Amazon S3

Conforme já mencionado, você também pode usar o Amazon S3 para armazenar valores de atributos grandes que não podem caber em um item do DynamoDB. Você pode armazená-los como um objeto no Amazon S3 e armazenar o identificador de objeto no item do DynamoDB.

Você também pode usar o suporte de metadados do objeto no Amazon S3 para fornecer um link para o item pai no DynamoDB. Armazene o valor da chave primária do item como metadados de objeto do Amazon S3 no Amazon S3. Fazer isso normalmente ajuda com a manutenção de objetos do Amazon S3.

Por exemplo, considere a tabela `ProductCatalog`. Os itens nessa tabela armazenam informações sobre preço do item, descrição, autores de livros e dimensões para outros produtos. Se você quisesse armazenar uma imagem de cada produto muito grande para se ajustar em um item, poderia armazenar as imagens no Amazon S3, em vez de no DynamoDB.

Ao implementar essa estratégia, lembre-se sempre do seguinte:

- O DynamoDB não oferece suporte a transações que cruzam o Amazon S3 e o DynamoDB. Portanto, sua aplicação deve lidar com quaisquer falhas, o que poderia incluir a limpeza de objetos órfãos do Amazon S3.
- O Amazon S3 limita o tamanho dos identificadores de objetos. Então, você deve organizar seus dados de um modo que não permita a geração excessiva de identificadores de objeto nem viole outras restrições do Amazon S3.

Para obter mais informações sobre como usar o Amazon S3, consulte o Guia do usuário do Amazon Simple Storage Service.

Práticas recomendadas para lidar com dados de séries temporais no DynamoDB

Os princípios gerais de design no Amazon DynamoDB recomendam manter o número mínimo de tabelas em uso. Para a maioria dos aplicativos, uma tabela única é tudo que você precisa. No entanto, no caso de dados de séries temporais, é possível lidar melhor com isso usando uma tabela por aplicativo por período.

Padrão de design para dados de séries temporais

Considere um cenário típico de séries temporais, em que você deseja monitorar um alto volume de eventos. O padrão de acesso de gravação é que todos os eventos sendo registrados tenham a data de hoje. O padrão de acesso de leitura pode ser: ler os eventos de hoje com mais frequência, os eventos de ontem com menos frequência e, então, os eventos mais antigos com muito menos frequência. Uma forma de lidar com isso é por meio da criação de data e hora atuais na chave primária.

O seguinte padrão de design normalmente lida com esse tipo de cenário de modo efetivo:

- Crie uma tabela por período, provisionada com a capacidade de leitura e gravação necessária e os índices necessários.
- Antes do final de cada período, crie previamente a tabela para o próximo período. Assim que o período atual terminar, direcione o tráfego de eventos para a nova tabela. É possível atribuir nomes a essas tabelas que especifiquem os períodos em que foram registradas.
- Quando uma tabela não estiver mais sendo gravada, reduza sua capacidade de gravação provisionada para um valor mais baixo (por exemplo, 1 WCU) e provisione a capacidade de leitura que for mais apropriada. Reduz a capacidade de leitura provisionada de tabelas anteriores conforme elas envelhecem. Você pode optar por arquivamento ou exclusão das tabelas cujo conteúdo raramente ou nunca é necessário.

A ideia é alocar os recursos necessários para o período atual que receberá o maior volume de tráfego e diminui o provisionamento para tabelas mais velhas que não são usadas ativamente, diminuindo assim os custos. Dependendo das necessidades de seus negócios, você pode considerar estilhaçar a gravação para distribuir o tráfego de forma uniforme para a chave de partição lógica. Para obter mais informações, consulte Usar a fragmentação de gravação para distribuir workloads uniformemente em uma tabela do DynamoDB.

Exemplos de tabelas de séries temporais

Veja a seguir um exemplo de dados de séries temporais no qual a tabela atual é provisionada com uma capacidade mais alta de leitura/gravação e as tabelas mais antigas são reduzidas porque não são acessadas com frequência:

Current table — Provisioned at: WCU=750 and RCU=300

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-15	00:00:00.002	17.372 W/Sr	713 nm	...
2018-03-15	00:00:00.004	17.385 W/Sr	712 nm	...
2018-03-15	00:00:00.005	17.478 W/Sr	708 nm	...
2018-03-15	00:00:00.007	19.172 W/Sr	674 nm	...
• • •	• • •	• • •	• • •	...

Previous table — Provisioned at: WCU=1 and RCU=100

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-14	00:00:00.001	16.473 W/Sr	512	...
2018-03-14	00:00:00.003	16.489 W/Sr	519	...
2018-03-14	00:00:00.004	16.814 W/Sr	522	...
2018-03-14	00:00:00.006	16.719 W/Sr	506	...
• • •	• • •	• • •	• • •	...

Older table — Provisioned at: WCU=1 and RCU=1

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-10	00:00:00.001	13.669 W/Sr	456	...
2018-03-10	00:00:00.002	13.522 W/Sr	459	...
2018-03-10	00:00:00.004	13.596 W/Sr	457	...
2018-03-10	00:00:00.005	15.721 W/Sr	425	...
• • •	• • •	• • •	• • •	...

Práticas recomendadas para gerenciar relações de muitos para muitos no DynamoDB

As listas de adjacências são um padrão de design útil para modelar as relações muitos para muitos no Amazon DynamoDB. De modo mais geral, elas oferecem uma forma de representar os dados gráfico (nós e bordas) no DynamoDB.

Padrão de design da lista de adjacências

Quando entidades diferentes de um aplicativo têm uma relação muitos para muitos entre elas, a relação pode ser modelada como uma lista de adjacências. Nesse padrão, todas as entidades de nível superior (sinônimos para nós no modelo de gráfico) são representadas usando a chave de partição. Qualquer relação com outras entidades (bordas em um gráfico) é representada como um item na partição configurando o valor da chave de classificação como o ID da entidade de destino (nó de destino).

As vantagens desse padrão incluem a duplicação mínima de dados e padrões simplificados de consulta para encontrar todas as entidades (nós) relacionadas a uma entidade de destino (o ponto em um nó de destino).

Como exemplo real, esse padrão foi útil em um sistema de faturamento em que as faturas continham várias contas. Uma conta pode pertencer a várias faturas. A chave de partição nesse exemplo é um `InvoiceID` ou um `BillID`. As partições `BillID` têm todos os atributos específicos para as contas. As partições `InvoiceID` têm um item que armazena os atributos específicos à fatura e um item para cada `BillID` implementado para a fatura.

O esquema é semelhante ao seguinte.

Table	Primary Key		Data Attributes...	
	Partition Key	*Sort Key (and GSI PK)*		
	`Invoice-92551`	**Inv_ID:** Invoice-92551 *(invoice ID)*	**Dated:** 2018-02-07 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	Attributes of this bill *in this invoice*..
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Invoice-92552`	**Inv_ID:** Invoice-92552 *(invoice ID)*	**Dated:** 2018-03-04 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Bill-4224663`	**Bill_ID:** Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	More attributes of this bill...
	`Bill-4224687`	**Bill_ID:** Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	More attributes of this bill...

Usando o esquema anterior, você pode ver que todas as contas para uma fatura podem ser consultadas usando a chave primária na tabela. Para pesquisar todas as faturas que contêm uma parte de uma conta, crie um índice secundário global na chave de classificação da tabela.

As projeções do índice secundário global são semelhantes ao seguinte.

GSI	Primary Key *Partition Key*	Projected Attributes...	
	Bill-4224663	**Bill_ID:** Bill-4224663 *(table primary key)*	Attributes of this bill...
		Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
	Bill-4224687	**Bill_ID:** Bill-4224687 *(table primary key)*	Attributes of this bill...
		Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
		Inv_ID: Invoice-92552 *(table primary key)*	Attributes of this bill *in this invoice*..
	Invoice-92551	**Inv_ID:** Invoice-92551 *(table primary key)*	Attributes of this invoice...
	Invoice-92552	**Inv_ID:** Invoice-92552 *(table primary key)*	Attributes of this invoice...

Padrão de gráficos materializados

Muitos aplicativos são criados com base no entendimento das classificações entre os colegas, nas relações comuns entre entidades, no estado da entidade vizinha e em outros tipos de fluxos de trabalho de estilo de gráficos. Para esses tipos de aplicativos, considere o seguinte padrão de design de esquema.

TABLE	Primary Key		Attributes		
	PK (NodeId)	SK (TypeTarget, GSI 2 SK)	Data	GSI PK	Graph Projections
	1	DATE|2|BIRTH	1980-12-19	Hash(Person.Data)	
		PERSON|1	Data (GSI1 SK)	GSI PK	
			John Doe	Hash(Person.Data)	
		PERSON|5|FRIEND	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PLACE|4|BIRTH	Data	GSI PK	
			USA|Texas|Austin	Hash(Person.Data)	
		SKILL|6	Data	GSI PK	
			Java Developer|Senior	Hash(Person.Data)	
	2	DATE|2	Data	GSI PK	
			1980-12-19	0	
	3	PLACE|3	Data	GSI PK	
			UK|England|London	0	
	4	PLACE|4	Data	GSI PK	
			USA|Texas|Austin	0	...
	5	DATE|2|BIRTH	Data	GSI PK	
			1980-12-19	Hash(Person.Data)	
		PERSON|5	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PERSON|1|FRIEND	Data	GSI PK	
			John Doe	Hash(Person.Data)	
		PLACE|3|BIRTH	Data	GSI PK	
			UK|England|London	Hash(Person.Data)	
		SKILL|7	Data	GSI PK	
			Guitar|Advanced	Hash(Person.Data)	
	6	SKILL|6	Data	GSI PK	
			Java Developer	0	
	7	SKILL|7	Data	GSI PK	
			Guitar	0	

	Primary Key		Attributes		
	GSI PK	GSI 1 SK (Data)			
GSI 1	O-N	1980-12-19	NodeID	TypeTarget	Graph Projections
			2	DATE|2	...
			NodeID	TypeTarget	
			1	DATE|2|BIRTH	
			NodeID		
			5		
		Guitar	NodeID	TypeTarget	
			7	SKILL|7	
		Guitar|Advanced	NodeID		
			5		
		Jane Smith	NodeID	TypeTarget	
			5	Person|5	
			NodeID	TypeTarget	
			1	Person|5|FRIEND	
		Java Developer	NodeID	TypeTarget	
			6	SKILL|6	
		Java Developer|Senior	NodeID		
			1		
		John Doe	NodeID	TypeTarget	
			1	Person|1	
			NodeID	TypeTarget	
			5	Person|1|FRIEND	
		UK|England|London	NodeID	TypeTarget	
			3	PLACE|3	
			NodeID	TypeTarget	
			1	PLACE|3|BIRTH	
		USA|Texas|Austin	NodeID	TypeTarget	
			4	PLACE|4	
			NodeID	TypeTarget	
			5	PLACE|4|BIRTH	

	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)			
GSI 2	O-N	DATE|2	NodeID	Data	Graph Projections
			2	1980-12-19	...
		DATE|2|BIRTH	NodeID		
			1		
			NodeID		
			5		
		PERSON|1	NodeID	Data	
			1	John Doe	
		PERSON|1|FRIEND	NodeID		
			5		
		PERSON|5	NodeID	Data	
			5	Jane Smith	
		PERSON|5|FRIEND	NodeID		
			1		
		PLACE|3	NodeID	Data	
			3	UK|England|London	
		PLACE|3|BIRTH	NodeID		
			5		
		PLACE|4	NodeID	Data	
			4	USA|texas|Austin	
		PLACE|4|BIRTH	NodeID		
			1		
		SKILL|6	NodeID	Data	
			6	Java Developer	
			NodeID	Data	
			1	Java Developer|Senior	
		SKILL|7	NodeID	Data	
			7	Guitar	
			NodeID	Data	
			5	Guitar|Advanced	

O esquema anterior mostra uma estrutura de dados em gráfico que é definida por um conjunto de partições de dados contendo os itens que definem as bordas e os nós do gráfico. Os itens da borda contêm um atributo `Target` e um `Type`. Esses atributos são usados como parte do nome de uma chave composto “TypeTarget” para identificar o item em uma partição na tabela principal ou em outro índice secundário global.

O primeiro índice secundário global é criado no atributo `Data`. Esse atributo usa a sobrecarga do índice secundário global conforme descrito anteriormente para indexar vários tipos diferentes de atributo, a saber `Dates`, `Names`. `Places` e `Skills`. Aqui, um índice secundário global está indexando efetivamente quatro atributos diferentes.

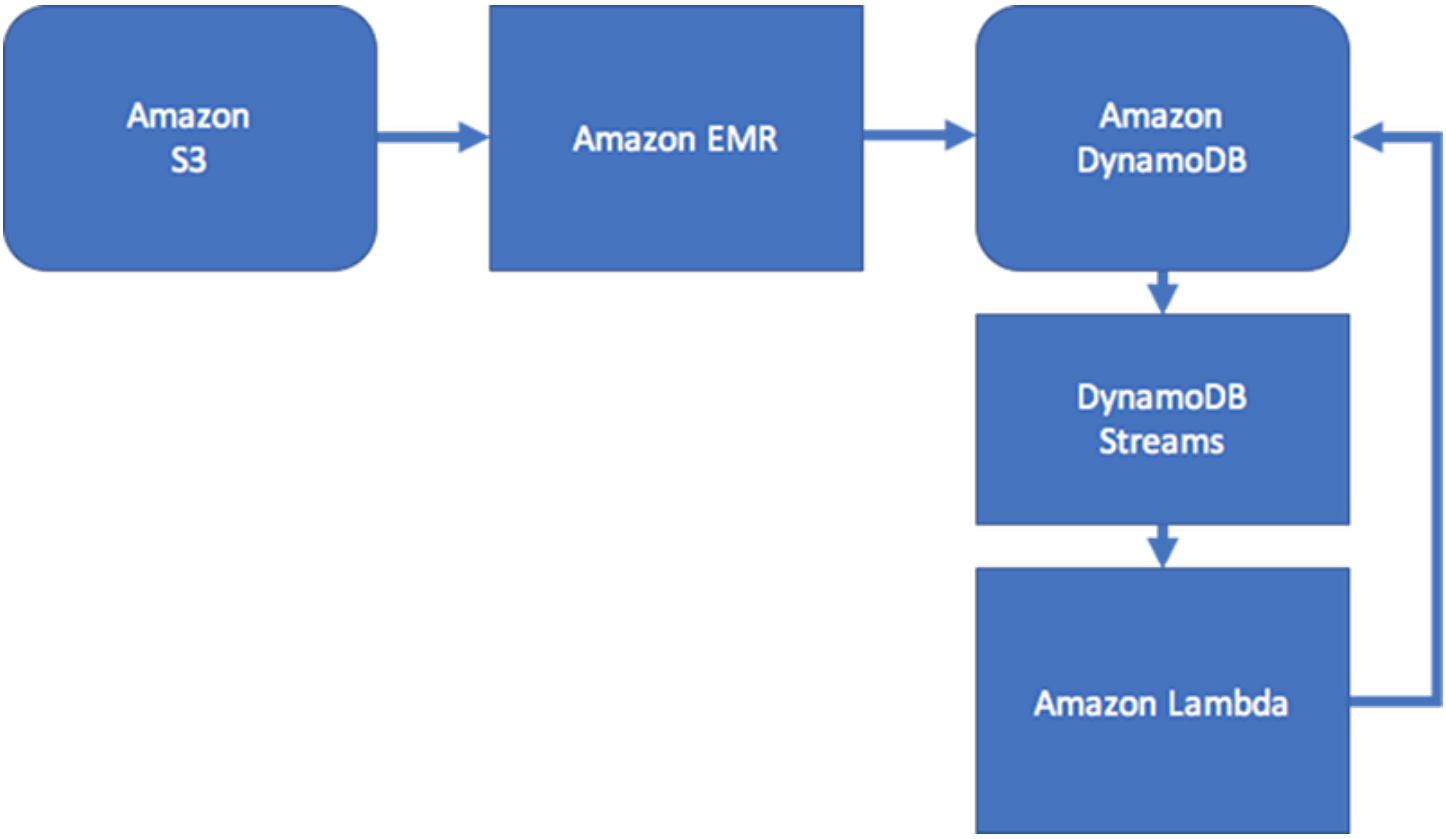
À medida que você inserir itens na tabela, poderá usar uma estratégia de fragmentação inteligente para distribuir os conjuntos de item com grandes agregações (data de nascimento, habilidade) nas partições lógicas nos índices secundários globais que forem necessárias para evitar problemas de leitura/gravação dinâmicas.

O resultado dessa combinação de padrões de design é um datastore sólido para fluxos de trabalho de gráfico altamente eficientes e em tempo real. Esses fluxos de trabalho podem fornecer consultas

de alto desempenho a agregação de borda e ao estado da entidade vizinha para mecanismos de recomendação, aplicativos de rede social, classificações de nós, agregações de subárvore e outros casos de uso comuns de gráficos.

Se seu caso de uso não for sensível a consistência de dados em tempo real, você poderá usar um processo do Amazon EMR programado para preencher as bordas com agregações relevantes de resumo de gráfico para seus fluxos de trabalho de um modo econômico. Se o aplicativo não precisar saber imediatamente quando uma borda é adicionada ao gráfico, será possível usar um processo programado para agregar resultados.

Para manter algum nível de consistência, o design poderia incluir Amazon DynamoDB Streams e AWS Lambdaparar processar atualizações da borda. Ele também poderia usar um trabalho do Amazon EMR para validar os resultados em um intervalo regular. Essa abordagem é ilustrada pelo diagrama a seguir. Ela é usada comumente em aplicativos de rede social, onde o custo de uma consulta em tempo real é alto, e a necessidade de saber imediatamente as atualizações de usuário individual é baixa.



Os aplicativos de segurança e gerenciamento de serviços de TI (ITSM - IT service-management) geralmente precisa responder em tempo real às alterações de estado da entidade compostas de agregações complexas de borda. Esses aplicativos precisam de um sistema que seja compatível

com várias agregações de nó em tempo real de relações de segundo e terceiros níveis ou de percursos complexos de borda. Se o seu caso de uso exigir esses tipos de fluxos de trabalho de consulta de gráficos em tempo real, recomendamos que você considere o uso do Amazon Neptune para gerenciar esses fluxos de trabalho.

> **Note**
>
> Se você precisar consultar conjuntos de dados altamente conectados ou executar consultas que precisem passar por vários nós (também conhecidas como consultas multi-hop) com latência de milissegundos, considere a possibilidade de usar o Amazon Neptune. O Amazon Neptune é um mecanismo de banco de dados de grafos com propósito específico e alta performance, otimizado para armazenar bilhões de relacionamentos e consultar grafos com latência de milissegundos.

Práticas recomendadas para consulta e varredura de dados no DynamoDB

Esta seção aborda algumas práticas recomendadas para o uso de operações `Query` e `Scan` no Amazon DynamoDB.

Considerações sobre desempenho para verificações

Em geral, as operações `Scan` são menos eficientes do que outras operações no DynamoDB. A operação `Scan` sempre verifica toda a tabela ou o índice secundário. Em seguida, ela filtra valores para fornecer o resultado desejado, adicionando, essencialmente, a etapa extra de remover os dados do conjunto de resultados.

Se possível, evite usar uma operação `Scan` em uma tabela ou índice grande com um filtro que remove muitos resultados. Além disso, conforme uma tabela ou índice cresce, a operação `Scan` torna-se mais lenta. A operação `Scan` examina todos os itens com os valores solicitados e pode usar o throughput provisionado para uma tabela ou índice grande em uma única operação. Para tempos de resposta mais rápidos, crie suas tabelas e índices de forma que suas aplicações possam usar `Query`, em vez de `Scan`. (Para tabelas, você também pode considerar usar `GetItem` e as APIs `BatchGetItem`.)

Como alternativa, é possível criar sua aplicação para usar operações `Scan` de uma forma que minimize o impacto na taxa de solicitações. Isso pode incluir modelagem quando for mais eficiente

usar um índice secundário global em vez de uma operação `Scan`. Há mais informações sobre esse processo no vídeo a seguir.

Modelagem de padrões de acesso de baixa velocidade

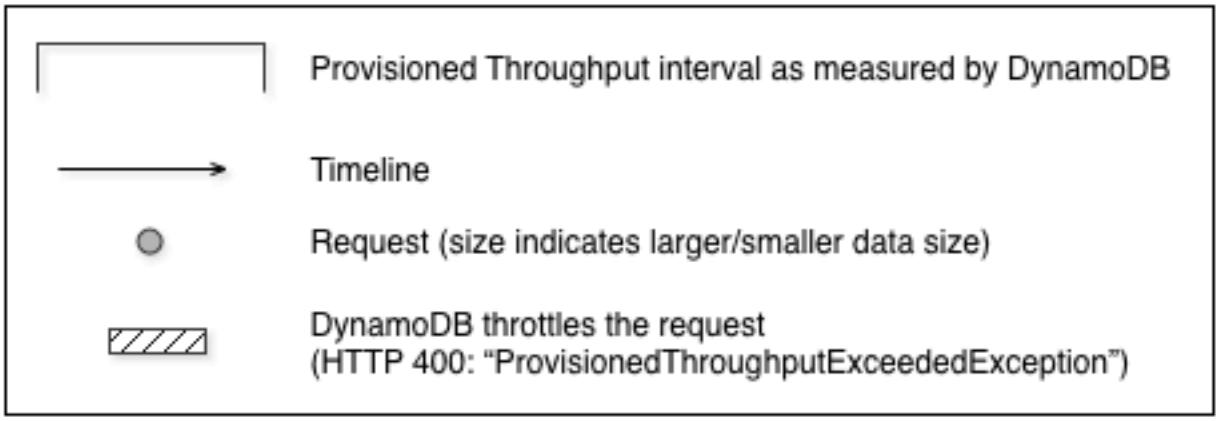
Evitar picos súbitos na atividade de leitura

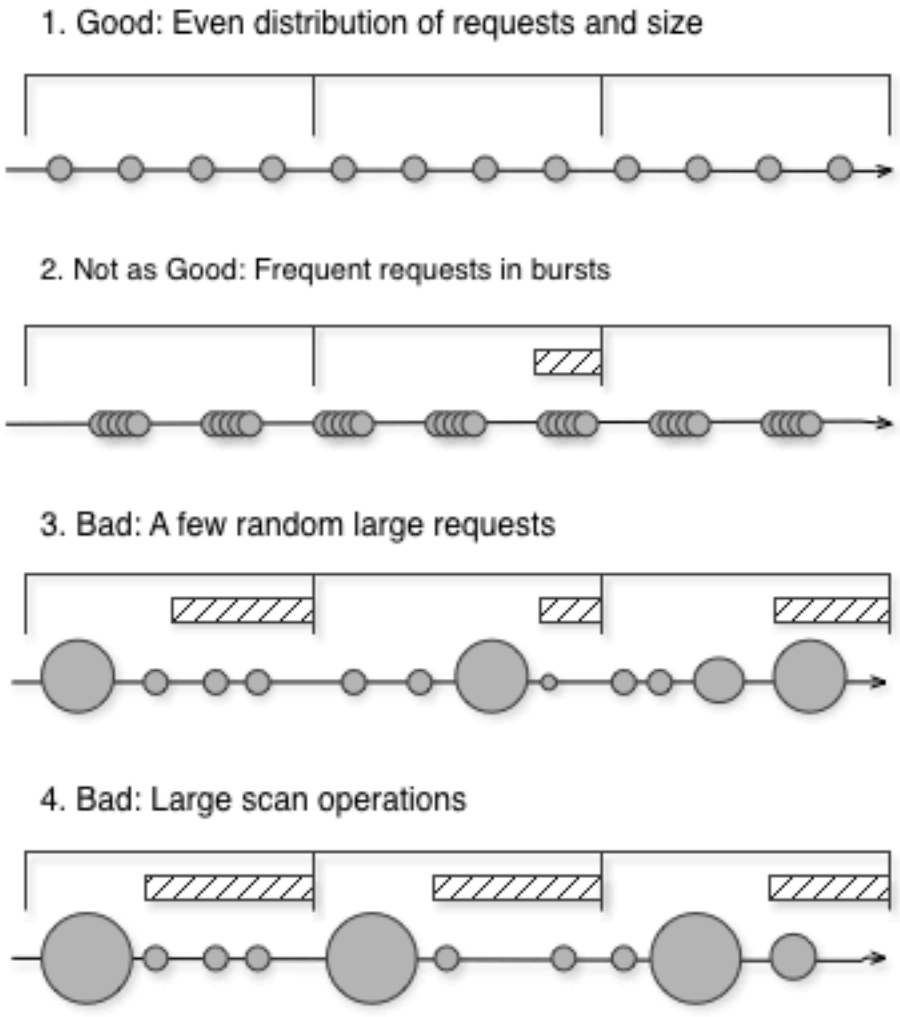
Ao criar uma tabela, você define seus requisitos de unidade de capacidade de leitura e gravação. Para leituras, as unidades de capacidade são expressas como o número de solicitações de leitura de dados de 4 KB fortemente consistentes por segundo. Para leituras finais consistentes, uma unidade de capacidade de leitura é duas solicitações de leitura de 4 KB por segundo. Uma operação `Scan` realiza leituras finais consistentes por padrão, e pode retornar até 1 MB (uma página) de dados. Portanto, uma única solicitação `Scan` pode consumir (tamanho de página de 1 MB/tamanho de item de 4KB)/2 (leituras finais consistentes) = 128 operações de leitura. Se você solicitasse leituras altamente consistentes em vez disso, a operação `Scan` consumiria duas vezes mais throughput provisionado (256 operações de leitura).

Isso representa um pico súbito no uso em comparação com a capacidade de leitura configurada para a tabela. Esse uso de unidades de capacidade por uma verificação impede que outras solicitações potencialmente mais importante para a mesma tabela use as unidades de capacidade disponíveis. Como resultado, você provavelmente receberá uma exceção `ProvisionedThroughputExceeded` para essas solicitações.

O problema não é apenas o aumento repentino nas unidades de capacidade usadas pelo `Scan`. A verificação provavelmente consumirá todas as suas unidades de capacidade da mesma partição, pois a verificação solicita itens de leitura que estão próximos uns dos outros na partição. Isso significa que a solicitação está alcançando a mesma partição, fazendo com que todas as suas unidades de capacidade sejam consumidas, e controlando a utilização de outras solicitações para essa partição. Se a solicitação para leitura de dados estiver dividida entre várias partições, a operação não limitará uma partição específica.

O diagrama a seguir ilustra o impacto de um pico súbito no uso de unidades de capacidade pelas operações `Query` e `Scan`, e seu impacto nas outras solicitações para a mesma tabela.





Como ilustrado aqui, o pico de uso pode afetar o throughput provisionado da tabela de várias maneiras:

1. Bom: distribuição uniforme de solicitações e tamanho

2. Não tão bom: pedidos frequentes em rajadas
3. Ruim: algumas solicitações grandes aleatórias
4. Ruim: operações de verificação grandes

Em vez de usar uma operação `Scan` grande, você pode usar as técnicas a seguir para minimizar o impacto de uma verificação no throughput provisionado de uma tabela.

- Reduzir o tamanho da página

  Como uma operação Scan lê uma página inteira (por padrão, 1 MB), você pode reduzir o impacto da operação Scan configurando um tamanho de página menor. A operação `Scan` fornece um parâmetro Limit que você pode usar para definir o tamanho da página para sua solicitação. Cada solicitação `Query` ou `Scan` com um tamanho de página menor usa menos operações de leitura e cria uma "pausa" entre cada solicitação. Por exemplo, suponha que cada item tem 4 KB e você definiu o tamanho da página para 40 itens. Uma solicitação `Query` consumiria apenas 20 operações de leitura final consistente ou 40 operações de leitura fortemente consistente. Um número maior de operações `Query` ou `Scan` menores permitiria que suas outras solicitações críticas tivessem êxito sem controle de utilização.

- Operações de verificação isoladas

  O DynamoDB foi projetado para facilitar a escalabilidade. Como resultado, uma aplicação pode criar tabelas para fins distintos e, possivelmente, duplicar conteúdo em várias tabelas. Você deseja executar verificações em uma tabela que não está recebendo tráfego "essencial à missão". Algumas aplicações lidam com essa carga roteando tráfego entre duas tabelas por hora – uma para tráfego crítico e outra para fins de registro. Outras aplicações podem fazer isso, executando todas as gravações em duas tabelas: uma tabela de "missão crítica" e uma tabela de "sombra".

Você deve configurar sua aplicação para tentar novamente qualquer solicitação que receba um código de resposta que indica que você excedeu o throughput provisionado. Ou, aumente o throughput provisionado para sua tabela usando a operação `UpdateTable`. Se houver picos temporárias em sua workload que fazem com que o throughput exceda, ocasionalmente, o nível provisionado, tente novamente a solicitação com recuo exponencial. Para obter mais informações sobre a implementação de recuo exponencial, consulte Repetições de erro e recuo exponencial.

Aproveitar as verificações paralelas

Muitas aplicações podem se beneficiar do uso de operações `Scan` paralelas, em vez de verificações sequenciais. Por exemplo, uma aplicação que processa uma tabela de dados históricos grande pode executar uma verificação paralela muito mais rapidamente do que uma verificação sequencial. Vários threads de operadores em um processo de "varredura" em segundo plano poderiam verificar uma tabela com baixa prioridade sem afetar o tráfego de produção. Em cada um desses exemplos, uma operação `Scan` paralela é usada de forma a não enfraquecer outras aplicações de recursos de throughput provisionado.

Embora as verificações paralelas possam ser benéficas, elas podem representar uma demanda pesada sobre o throughput provisionado. Com uma verificação paralela, sua aplicação tem vários operadores que executam operações `Scan` simultâneas. Isso pode consumir rapidamente toda a capacidade de leitura provisionada da sua tabela. Nesse caso, outras aplicações que precisam acessar a tabela podem ter controle de utilização.

A verificação paralela pode ser a escolha certa se as seguintes condições forem cumpridas:

- O tamanho da tabela é 20 GB ou maior.
- O throughput de leitura provisionado da tabela não está sendo completamente utilizada.
- Operações `Scan` sequenciais são muito lentas.

Escolher TotalSegments

A melhor configuração para `TotalSegments` depende de seus dados específicos, das configurações de throughput provisionado da tabela e de seus requisitos de performance. Você provavelmente precisará experimentar até obter os resultados desejados. Recomendamos começar com uma taxa simples, como um segmento por 2 GB de dados. Por exemplo, para uma tabela de 30 GB, você poderia definir `TotalSegments` até 15 (30 GB/2 GB). Sua aplicação poderia usar 15 operadores, com cada operador verificando um segmento diferente.

Você também pode escolher um valor para `TotalSegments` que seja baseado em recursos do cliente. Você pode definir `TotalSegments` como qualquer número de 1 a 1000000, e o DynamoDB permitirá que você execute uma verificação desse número de segmentos. Por exemplo, se o cliente limitar o número de threads que podem ser executados simultaneamente, você poderá aumentar `TotalSegments` gradualmente até obter a melhor performance de `Scan` com sua aplicação.

Você precisará monitorar suas verificações paralelas para otimizar a utilização de throughput provisionado e, ao mesmo tempo, garantir que suas outras aplicações não careçam de recursos. Aumente o valor de `TotalSegments` se você não consumir todo o throughput provisionado, mas ainda experimentar algum controle de utilização em suas solicitações de `Scan`. Reduza o valor de `TotalSegments` se as solicitações de `Scan` consomem mais throughput provisionado do que você deseja usar.

Práticas recomendadas para o design de tabelas do DynamoDB

Os princípios gerais de design no Amazon DynamoDB recomendam manter o número mínimo de tabelas em uso. Na maioria dos casos, recomendamos que você considere usar uma única tabela. No entanto, se uma única tabela ou um pequeno número de tabelas não forem opções viáveis, estas diretrizes poderão ser úteis.

- O limite por conta não pode ultrapassar 10 mil tabelas por conta. Se sua aplicação precisar de mais tabelas, planeje distribuí-las em várias contas. Para obter mais informações, consulte Cotas de serviço, conta e tabelas no Amazon DynamoDB.
- Considere os limites do ambiente de gerenciamento para operações simultâneas do ambiente de gerenciamento que possam impactar seu gerenciamento de tabelas.
- Trabalhe com arquitetos de soluções da AWS para validar seus padrões de design para projetos com vários inquilinos.

Práticas recomendadas para o design de tabelas globais do DynamoDB

As tabelas globais aproveitam a presença global do Amazon DynamoDB para oferecer a você um banco de dados totalmente gerenciado, multiativo e com várias regiões que fornece performance rápida e local de leitura e gravação para aplicações globais com grande ajuste de escala. Com as tabelas globais, os dados se replicam automaticamente nas regiões da AWS escolhidas por você. Como as tabelas globais usam APIs existentes do DynamoDB, nenhuma alteração será necessária na aplicação. Não há nenhum custo ou compromisso inicial pelo uso de tabelas globais, e você paga apenas pelos recursos que usa.

Tópicos

- Orientação prescritiva para o design de tabelas globais do DynamoDB
- Fatos importantes sobre o design de tabelas globais do DynamoDB

- Casos de uso de tabelas globais do DynamoDB
- Modos de gravação com tabelas globais do DynamoDB
- Roteamento de solicitações com tabelas globais do DynamoDB
- Evacuar uma região com tabelas globais do DynamoDB
- Planejamento da capacidade de throughput para tabelas globais do DynamoDB
- Lista de verificação de preparação para tabelas globais do DynamoDB e perguntas frequentes

Orientação prescritiva para o design de tabelas globais do DynamoDB

O uso eficiente de tabelas globais exige considerações cuidadosas de alguns fatores, como seu modo de gravação preferido, modelo de roteamento e processos de evacuação. Você deve instrumentar sua aplicação em todas as regiões e se preparar para ajustar o roteamento ou realizar uma evacuação para manter a integridade global. A recompensa é ter um conjunto de dados distribuído globalmente com leituras e gravações de baixa latência e um acordo de serviço de 99,999%.

Fatos importantes sobre o design de tabelas globais do DynamoDB

- Há duas versões de tabelas globais: a versão atual Global Tables versão 2019.11.21 (atual) (às vezes chamada de “V2”) e Global Tables versão 2017.11.29 (herdada) (às vezes chamada de “V1”). Este guia se concentra exclusivamente na versão atual, V2.
- Sem o uso de tabelas globais, o DynamoDB é um serviço regional. É altamente disponível e intrinsecamente resiliente a falhas de infraestrutura em uma região, incluindo a falha de uma zona de disponibilidade (AZ) inteira. Uma tabela do DynamoDB de região única tem um Acordo de Serviço (SLA) com 99,99% de disponibilidade. https://aws.amazon.com/dynamodb/sla/
- Com o uso de tabelas globais, o DynamoDB permite que uma tabela replique seus dados entre duas ou mais regiões. Uma tabela multirregional do DynamoDB tem um SLA com 99,999% de disponibilidade. Com um planejamento adequado, as tabelas globais podem ajudar na criação de uma arquitetura que seja resiliente e resistente a falhas regionais.
- As tabelas globais empregam um modelo de replicação ativa-ativa. Do ponto de vista do DynamoDB, a tabela em cada região tem a mesma posição para aceitar solicitações de leitura e gravação. Depois de receber uma solicitação de gravação, a tabela da réplica local replicará a gravação para outras regiões participantes em segundo plano.
- Os itens são replicados individualmente. Os itens atualizados em uma única transação não podem ser replicados juntos.

- Cada partição de tabela na região de origem replica suas gravações em paralelo com todas as outras partições. A sequência de gravações na região remota pode não corresponder à sequência de gravações que ocorreram na região de origem. Para obter mais informações sobre partições de tabelas, consulte a postagem do blog Ajuste de escala do DynamoDB: como as partições, as teclas de atalho e a divisão de calor afetam a performance.
- Um item recém-gravado geralmente é propagado para todas as tabelas de réplica em questão de poucos segundos. A propagação tende a ser mais rápida nas regiões próximas.
- O Amazon CloudWatch fornece uma métrica `ReplicationLatency` para cada par de regiões. É calculada com base na análise dos itens recebidos e na comparação do horário de chegada com o tempo inicial de gravação para o cálculo de uma média. Os tempos são armazenados no CloudWatch na região de origem. A visualização dos tempos médios e máximos pode ajudar a determinar o atraso médio e o maior atraso de replicação. Não há SLA sobre essa latência.
- Se o mesmo item for atualizado aproximadamente ao mesmo tempo (dentro dessa janela de `ReplicationLatency`) em duas regiões diferentes e a segunda gravação ocorrer antes de a primeira gravação ser replicada, poderão acontecer conflitos de gravação. As tabelas globais resolvem esses conflitos com o mecanismo de último gravador vence, com base no carimbo de data/hora das gravações. A primeira gravação “perde” para a segunda gravação. Esses conflitos não são registrados no CloudWatch ou no AWS CloudTrail.
- Cada item tem um carimbo de data/hora da última gravação mantido como uma propriedade privada do sistema. A abordagem de último gravador vence é implementada usando uma gravação condicional que exige que o carimbo de data/hora do item recebido seja maior do que o carimbo de data/hora do item existente.
- Uma tabela global replicará todos os itens em todas as regiões participantes. Se você quiser ter diferentes escopos de replicação, poderá criar tabelas diferentes e definir regiões participantes diferentes para cada uma das tabelas.
- As gravações serão aceitas na região local mesmo que a região da réplica esteja off-line ou que `ReplicationLatency` cresça. A tabela local continua tentando replicar itens na tabela remota até que cada item seja bem-sucedido.
- No caso improvável de uma região ficar totalmente off-line, quando ela voltar a ficar on-line, todas as replicações pendentes de entrada e saída serão repetidas. Nenhuma ação especial é necessária para sincronizar as tabelas novamente. O mecanismo de último gravador vence garante que os dados acabem se tornando consistentes.
- Você pode adicionar uma nova região a uma tabela do DynamoDB a qualquer momento. O DynamoDB cuidará da sincronização inicial e da replicação contínua. Se uma região for removida, mesmo que seja a região original, somente a tabela dessa região será excluída.

- O DynamoDB não tem um endpoint global. Todas as solicitações são feitas para um endpoint regional, que então acessa a instância da tabela global que é local dessa região.
- As chamadas para o DynamoDB não devem acontecer entre regiões. A prática recomendada é que a camada de computação em uma região acesse diretamente somente o endpoint local do DynamoDB dessa região. Se forem detectados problemas em uma região, sejam eles na camada do DynamoDB ou na pilha circundante, o tráfego do usuário final deverá ser roteado para outra camada de computação hospedada em uma região diferente. Graças à replicação de tabelas globais, a região diferente já terá uma cópia local dos mesmos dados para trabalhar localmente. Em algumas circunstâncias, a camada de computação em uma região pode transmitir a solicitação à camada computacional de outra região para processamento, mas isso não deve acessar diretamente o endpoint remoto do DynamoDB. Para obter mais informações sobre esse caso de uso específico, consulte Roteamento de solicitações na camada de computação.

Casos de uso de tabelas globais do DynamoDB

As tabelas globais oferecem os seguintes benefícios comuns:

- Leituras com latência mais baixa. Você pode colocar uma cópia dos dados mais perto do usuário final para reduzir a latência de rede durante as leituras. O cache é mantido tão atualizado quanto o valor de `ReplicationLatency`.
- Gravações com latência mais baixa. Você pode gravar em uma região próxima para reduzir a latência de rede e o tempo necessário para realizar a gravação. O tráfego de gravação deve ser roteado cuidadosamente para garantir que não haja conflitos. As técnicas de roteamento são discutidas em mais detalhes em Roteamento de solicitações com tabelas globais do DynamoDB.
- Maior resiliência e recuperação de desastres. Você pode evacuar uma região (afastar algumas ou todas as solicitações enviadas para essa região) caso a região apresente redução de performance ou interrupção total, com um objetivo de ponto de recuperação (RPO) e um objetivo de tempo de recuperação (RTO) medidos em segundos. O uso de tabelas globais também aumenta o SLA do DynamoDB de 99,99% para 99,999%.
- Migração entre regiões sem interrupção. Você pode adicionar uma nova região, depois excluir a região antiga para migrar uma implantação de uma região para outra sem nenhum tempo de inatividade na camada de dados. Por exemplo, você pode usar tabelas globais do DynamoDB para que um sistema de gerenciamento de pedidos obtenha um processamento confiável de baixa latência e em alta escala e, ao mesmo tempo, manter a resiliência a falhas regionais e de AZ.

Modos de gravação com tabelas globais do DynamoDB

As tabelas globais são sempre ativo-ativas no nível da tabela. No entanto, talvez você queira tratá-las como ativo-passivas ao controlar a forma como roteia as solicitações de gravação. Por exemplo, você pode decidir rotear solicitações de gravação para uma única região a fim de evitar possíveis conflitos de gravação.

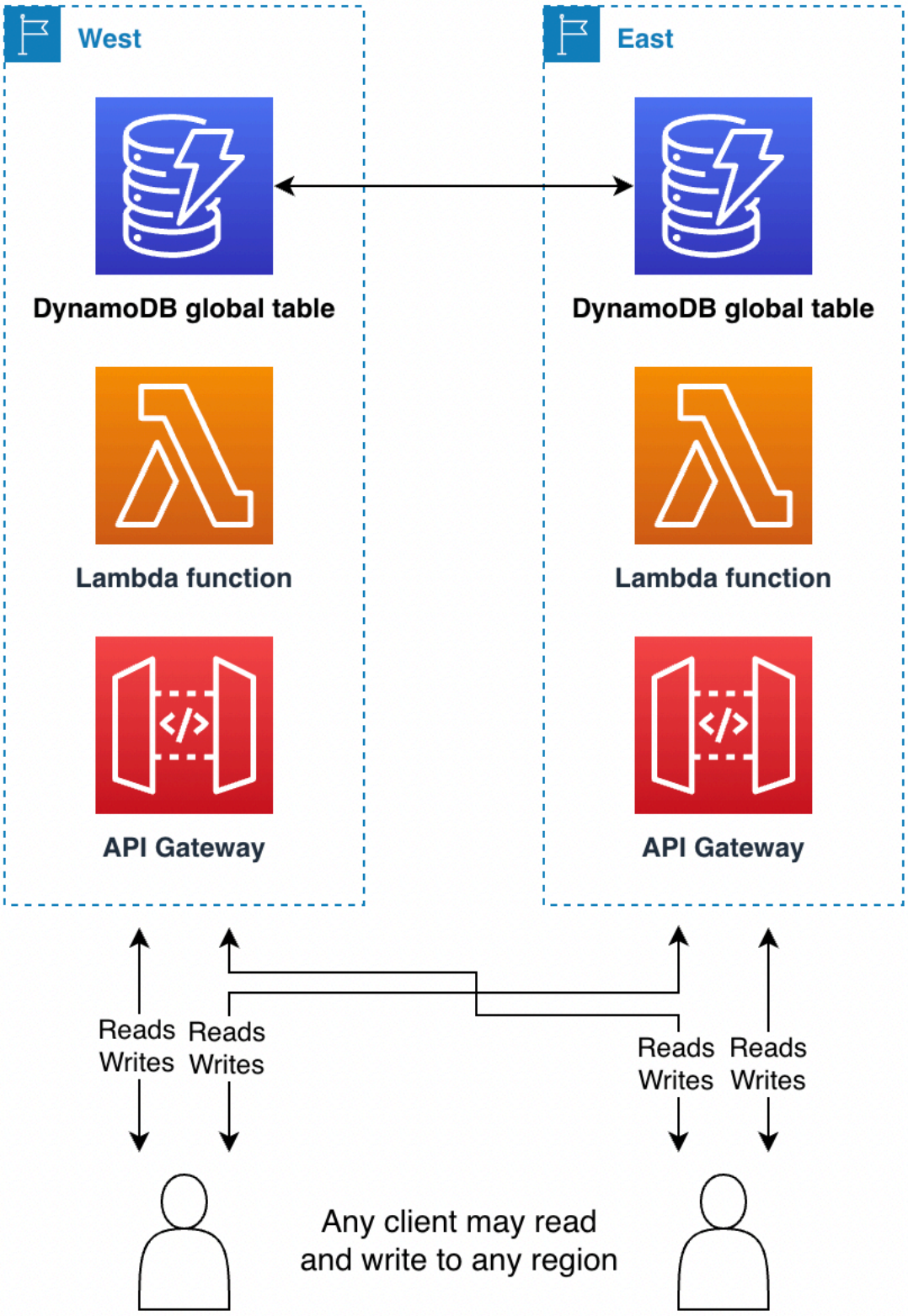
Há três categorias principais de padrões de gravação gerenciada:

- Modo de gravação em qualquer região (sem primária)
- Modo de gravação em uma região (primária única)
- Modo de gravação em sua região (primária mista)

Você deve considerar qual padrão de gravação se adapta ao seu caso de uso. Essa escolha afeta a forma como você roteia solicitações, evacua uma região e lida com a recuperação de desastres. As práticas recomendadas gerais podem ser diferentes dependendo do modo de gravação da sua aplicação.

Modo de gravação em qualquer região (sem primária)

O modo de gravação em qualquer região é totalmente ativo-ativo e não impõe nenhuma restrição sobre onde uma gravação pode ocorrer. Qualquer região pode aceitar uma gravação a qualquer momento. Esse é o modo mais simples. Pode ser usado somente com alguns tipos de aplicações. É adequado quando todos os gravadores são idempotentes e, portanto, repetíveis com segurança, para que as operações de gravação simultâneas ou repetidas entre regiões não entrem em conflito. Por exemplo, quando um usuário atualiza seus dados de contato. Esse modo também funciona bem para um caso especial de idempotência, um conjunto de dados somente para anexos em que todas as gravações são inserções exclusivas em uma chave primária determinística. Por fim, esse modo é adequado quando o risco de gravações conflitantes é aceitável.



O modo de gravação em qualquer região é a arquitetura mais simples de implementar. O roteamento é mais fácil porque qualquer região pode ser o destino de gravação a qualquer momento. O failover

é mais fácil porque qualquer gravação recente pode ser repetida várias vezes em qualquer região secundária. Sempre que possível, defina o design para usar esse modo de gravação.

Por exemplo, os serviços de streaming de vídeo geralmente usam tabelas globais para rastrear favoritos, avaliações, sinalizadores de status de exibição e assim por diante. Essas implantações podem usar o modo de gravação em qualquer região, desde que garantam que cada gravação seja idempotente e que o próximo valor correto de um item não dependa de seu valor atual. Esse será o caso para atualizações do usuário que atribuem diretamente o novo estado ao usuário, como definir um novo código de horário mais recente, atribuir uma nova avaliação ou definir um novo status do relógio. Se as solicitações de gravação do usuário forem encaminhadas para regiões diferentes, a última operação de gravação persistirá e o estado global será resolvido de acordo com a última atribuição. As operações de leitura nesse modo acabarão se tornando consistentes após serem atrasadas pelo valor mais recente de `ReplicationLatency`.

Em outro exemplo, uma empresa de serviços financeiros usa tabelas globais como parte de um sistema para manter um registro contínuo das compras com cartão de débito para cada cliente, a fim de calcular as recompensas de cashback desse cliente. Novas transações chegam do mundo inteiro e vão para várias regiões. O design atual da empresa não aproveita as tabelas globais, pois usa um único item de balancete corrente por cliente. As ações do cliente atualizam o saldo com uma expressão ADD, que não é idempotente porque o novo valor correto depende do valor atual. Isso significa que o saldo ficaria fora de sincronia se houvesse duas operações de gravação no mesmo saldo aproximadamente ao mesmo tempo em regiões diferentes.

Essa mesma empresa poderia implantar o modo de gravação em qualquer região por meio de uma reformulação cuidadosa com as tabelas globais do DynamoDB. O novo design poderia seguir um modelo de “streaming de eventos”, que, essencialmente, é um livro contábil com um fluxo de trabalho somente para anexos. Cada ação do cliente acrescenta um novo item à coleção de itens mantidos para esse cliente. A coleção de itens é o conjunto de itens que compartilham uma chave primária, com chaves de classificação diferentes. Cada ação de gravação que acrescenta a ação do cliente é uma inserção idempotente, usando o ID do cliente como chave de partição e o ID da transação como chave de classificação. Esse design torna o cálculo do saldo mais complicado, pois exige uma `Query` para extrair os itens seguida de alguns cálculos no lado do cliente. Mas a vantagem é que isso torna todas as gravações idempotentes, o que proporciona simplificações significativas de roteamento e failover. Para obter mais informações, consulte Roteamento de solicitações com tabelas globais do DynamoDB.

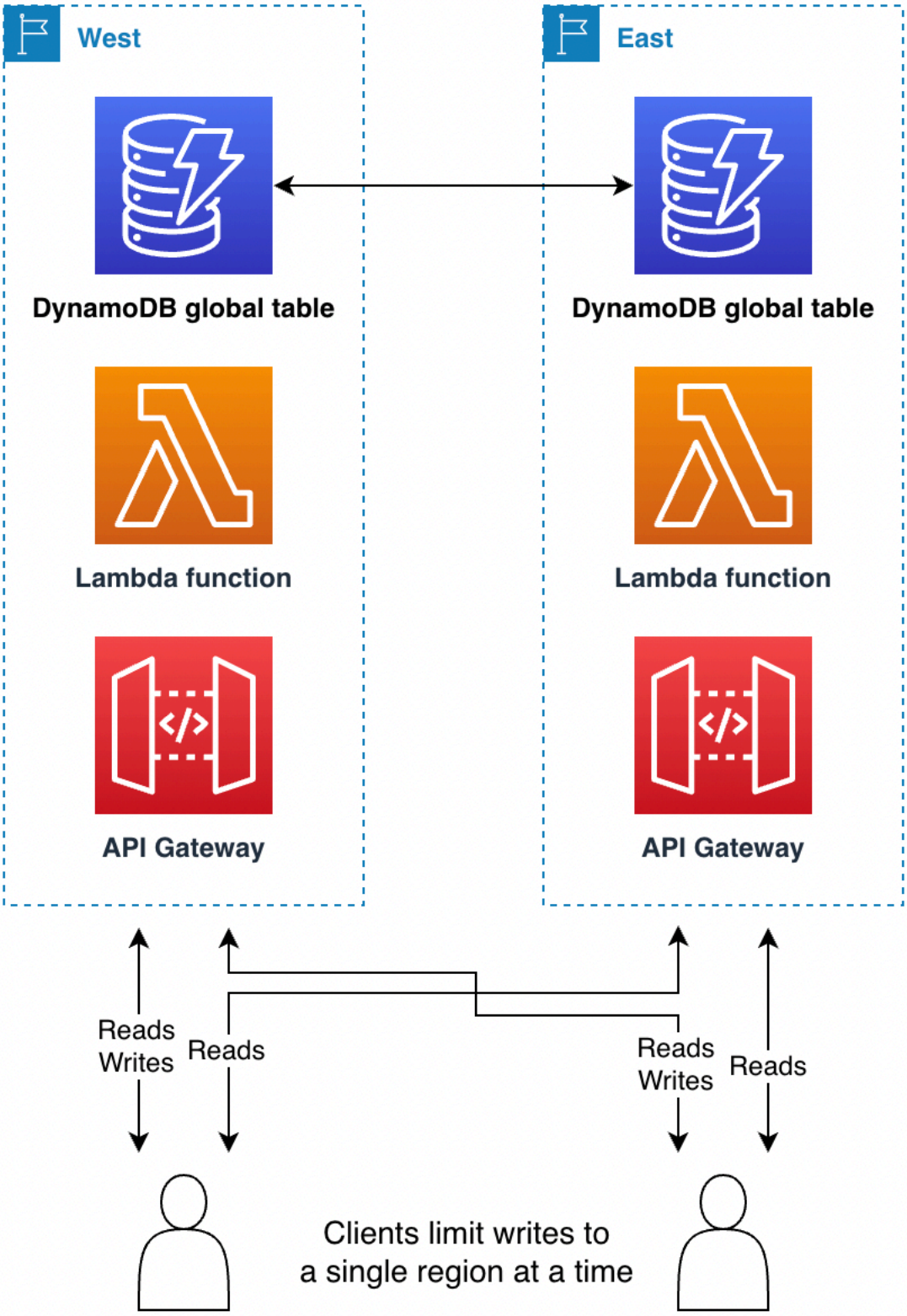
Para um terceiro exemplo, digamos que há um cliente que realiza posicionamento de anúncios on-line. O cliente decidiu que um baixo risco de perda de dados seria aceitável para obter as

simplificações de design do modo de gravação em qualquer região. Ao veicular anúncios, o cliente tem apenas alguns milissegundos para recuperar metadados suficientes para determinar qual anúncio exibir e, depois, para registrar a impressão do anúncio para que o mesmo anúncio não seja repetido para esse usuário. Com tabelas globais, o cliente pode obter leituras de baixa latência para usuários finais em todo o mundo e gravações de baixa latência. O cliente pode registrar todas as impressões de anúncios de um usuário em um único item e representar isso como uma lista crescente. Também pode usar um item em vez de anexá-lo a uma coleção de itens, pois dessa forma pode remover impressões de anúncios mais antigas como parte de cada gravação sem pagar pela exclusão. Essa operação de gravação NÃO é idempotente. Sendo assim, se o mesmo usuário final ver anúncios veiculados em várias regiões aproximadamente ao mesmo tempo, há uma chance de uma impressão de anúncio substituir a outra. Para o posicionamento de anúncios on-line, vale a pena correr o risco de um usuário ver ocasionalmente um anúncio repetido em troca de usar esse design mais simples e eficiente.

Primária única (“gravação em uma região”)

O modo de gravação em uma região é ativo-passivo e direciona todas as gravações de tabela para uma única região ativa. Observe que o DynamoDB não tem a noção de uma única região ativa; é o roteamento de aplicações fora do DynamoDB que gerencia isso. O modo de gravação em uma região evita conflitos de gravação ao garantir que as gravações fluam somente para uma região por vez. Esse modo de gravação é útil quando você deseja usar expressões ou transações condicionais, porque elas não funcionarão a menos que você saiba que está agindo com base nos dados mais recentes. Portanto, o uso de expressões e transações condicionais exige o envio de todas as solicitações de gravação para a única região com os dados mais recentes.

As leituras finais consistentes podem ir para qualquer região de réplica a fim de alcançar latências mais baixas. Leituras altamente consistentes devem ir para a única região primária.


West
East
DynamoDB global table
DynamoDB global table
Lambda function
Lambda function
API Gateway
API Gateway
Reads
Writes
Reads
Reads
Writes
Reads
Clients limit writes to
a single region at a time

Às vezes, é necessário alterar a região ativa em resposta a uma falha regional, para ajudar com os dados. Evacuar uma região com tabelas globais do DynamoDB é um exemplo desse caso de uso. Alguns clientes mudarão a região atualmente ativa em intervalos regulares, como uma implantação "follow-the-sun". Isso coloca a região ativa próxima à geografia com mais atividade, proporcionando a ela a menor latência de leituras e gravações. Também oferece a vantagem de chamar diariamente o caminho do código de alteração de região, garantindo que esteja com os testes em dia antes de qualquer recuperação de desastres.

As regiões passivas podem manter um conjunto reduzido de infraestrutura em torno do DynamoDB, que só é aumentado caso se tornem a região ativa. Para uma discussão mais aprofundada sobre os designs de luz-piloto e espera de preparo, consulte Arquitetura de recuperação de desastres (DR) na AWS, parte III: luz-piloto e espera de preparo.

O uso do modo de gravação em uma região funciona bem ao aproveitar tabelas globais para leituras de baixa latência distribuídas globalmente. Por exemplo, uma grande empresa de mídia social tem milhões de usuários e bilhões de postagens. Cada usuário é atribuído a uma região no momento da criação da conta, localizada geograficamente perto de sua localização. Todos os dados vão para essa tabela não global. A empresa usa uma tabela global separada para manter o mapeamento entre usuários e regiões de origem usando o modo de gravação em uma região. Isso mantém cópias somente para leitura em todo o mundo para ajudar a localizar os dados de cada usuário diretamente com o mínimo de latência adicional. As atualizações são raras (somente ao mudar a região de origem de um usuário) e sempre passam por uma região para gravar, para evitar qualquer chance de conflitos de gravação.

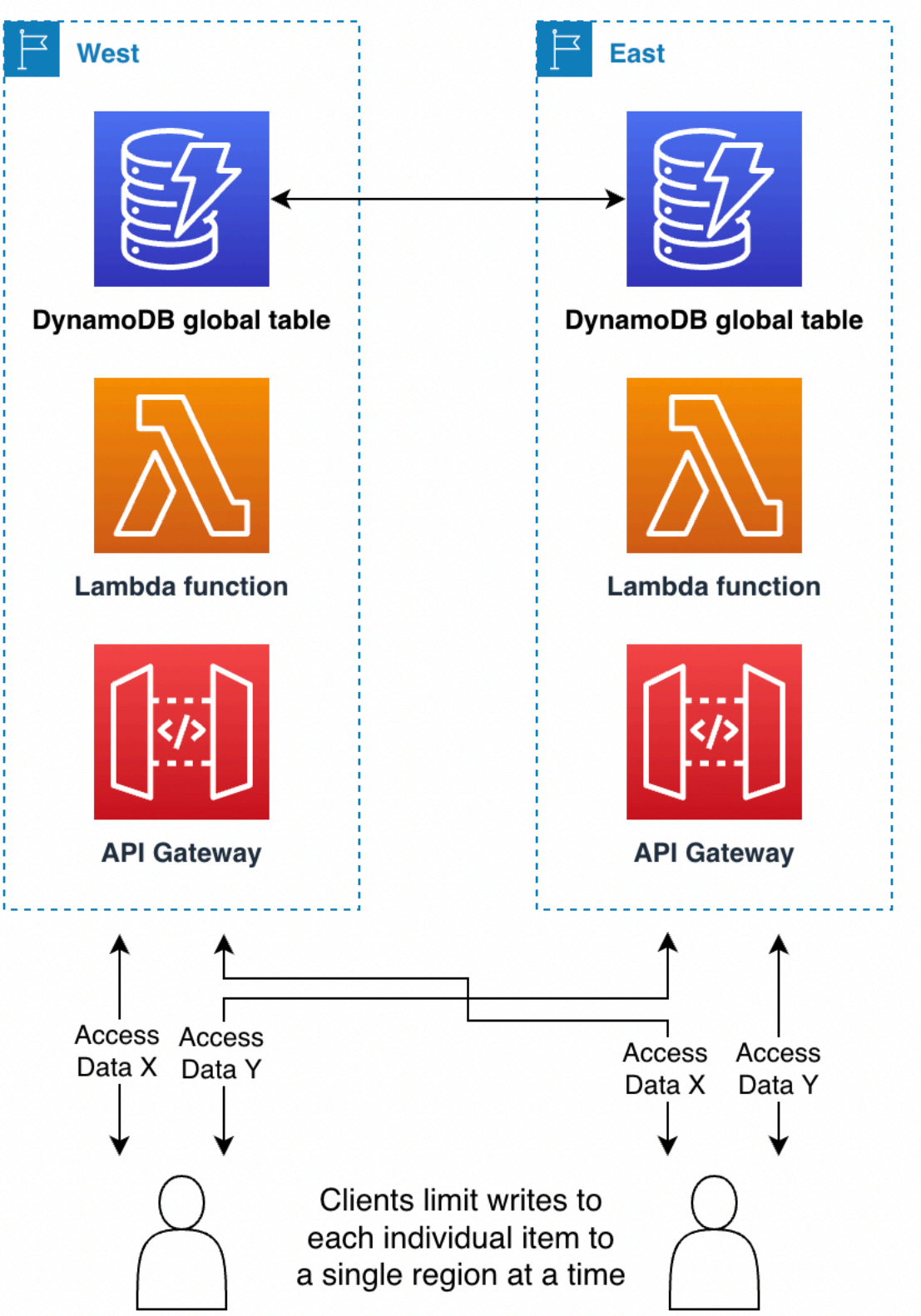
Como outro exemplo, considere um cliente de serviços financeiros que implementou um cálculo diário de cashback. O cliente usa o modo de gravação em qualquer região para calcular o saldo, mas usa o modo de gravação em uma região para rastrear os pagamentos de cashback. Se quiser recompensar um cliente com USD 0,01 por cada USD 10 gastos por dia, precisará usar `Query` para consultar todas as transações do dia anterior, calcular o total gasto, gravar a decisão de cashback em uma nova tabela, excluir o conjunto de itens consultado para marcá-los como consumidos e substituí-los por um único item armazenando qualquer valor restante que deve ser incluído nos cálculos do dia seguinte. Esse trabalho requer transações, portanto funcionará melhor com o modo de gravação em uma região. Uma aplicação pode combinar modos de gravação, até mesmo na mesma tabela, desde que as workloads não tenham chance de se sobrepor.

Primária mista ("gravação em sua região")

O modo de gravação em sua região atribui diferentes subconjuntos de dados a regiões diferentes e permite operações de gravação somente em itens na respectiva região de origem. Esse modo é

ativo-passivo, mas atribui a região ativa com base no item. Toda região é primária para seu próprio conjunto de dados não sobreposto, e as gravações devem ser protegidas para garantir a localidade adequada.

Esse modo é semelhante à gravação em uma região, exceto pelo fato de permitir gravações de latência mais baixa, pois os dados associados a cada usuário final podem ser colocados mais próximos desse usuário na rede. Também distribui a infraestrutura ao redor de forma mais uniforme entre as regiões e exige menos trabalho para desenvolver a infraestrutura durante um cenário de failover, porque todas as regiões terão uma parte de sua infraestrutura já ativa.



A determinação da região de origem dos itens pode ser feita de várias maneiras:

- Intrínseca: alguns aspectos dos dados deixam claro em qual região eles estão hospedados, como sua chave de partição. Por exemplo, um cliente e todos os dados sobre esse cliente seriam marcados nos dados do cliente como hospedados em determinada região. Essa técnica é descrita em Usar a fixação de região para definir uma região inicial para itens em uma tabela global do Amazon DynamoDB.
- Negociada: a região de origem de cada conjunto de dados é negociada de alguma forma externa, como com um serviço global separado que mantém as atribuições. A atribuição pode ter uma duração finita, após a qual estará sujeita à renegociação.
- Orientada por tabelas: em vez de uma única tabela global de replicação, tenha a mesma quantidade de tabelas globais e regiões replicadoras. O nome de cada tabela indica sua região de origem. Nas operações padrão, todos os dados são gravados na região de origem, enquanto outras regiões mantêm uma cópia somente leitura. Durante um failover, outra região adotará temporariamente as tarefas de gravação dessa tabela.

Por exemplo, imagine que você trabalha em uma empresa de jogos. Você precisa de leituras e gravações de baixa latência para todos os jogadores em todo o mundo. Você pode hospedar cada jogador na região mais próxima a ele. Essa região faz todas as leituras e gravações desse jogador, garantindo sempre uma consistência sólida de leitura após gravação. No entanto, se esse jogador viajar ou sua região de origem sofrer uma interrupção, uma cópia completa de seus dados estará disponível em regiões alternativas. Assim, o jogador pode ser atribuído a uma região de origem diferente, conforme a necessidade.

Em outro exemplo, imagine que você trabalha em uma empresa de videoconferência. Os metadados de cada teleconferência são atribuídos a uma região específica. Os chamadores podem usar a região mais próxima de si para obter a latência mais baixa. Se houver uma interrupção na região, o uso de tabelas globais permitirá uma recuperação rápida, pois o sistema poderá mover o processamento da chamada para uma região diferente, onde já existe uma cópia replicada dos dados.

Roteamento de solicitações com tabelas globais do DynamoDB

Talvez a parte mais complexa da implantação de uma tabela global seja gerenciar o roteamento de solicitações. As solicitações devem primeiro ir de um usuário final para uma região escolhida, depois devem ser roteadas de alguma forma. A solicitação encontra uma pilha de serviços nessa região, incluindo uma camada de computação que talvez consista em um balanceador de carga respaldado por uma função do AWS Lambda, um contêiner ou um nó do Amazon Elastic Compute Cloud (Amazon EC2) e, possivelmente, outros serviços, incluindo outro banco de dados. Essa

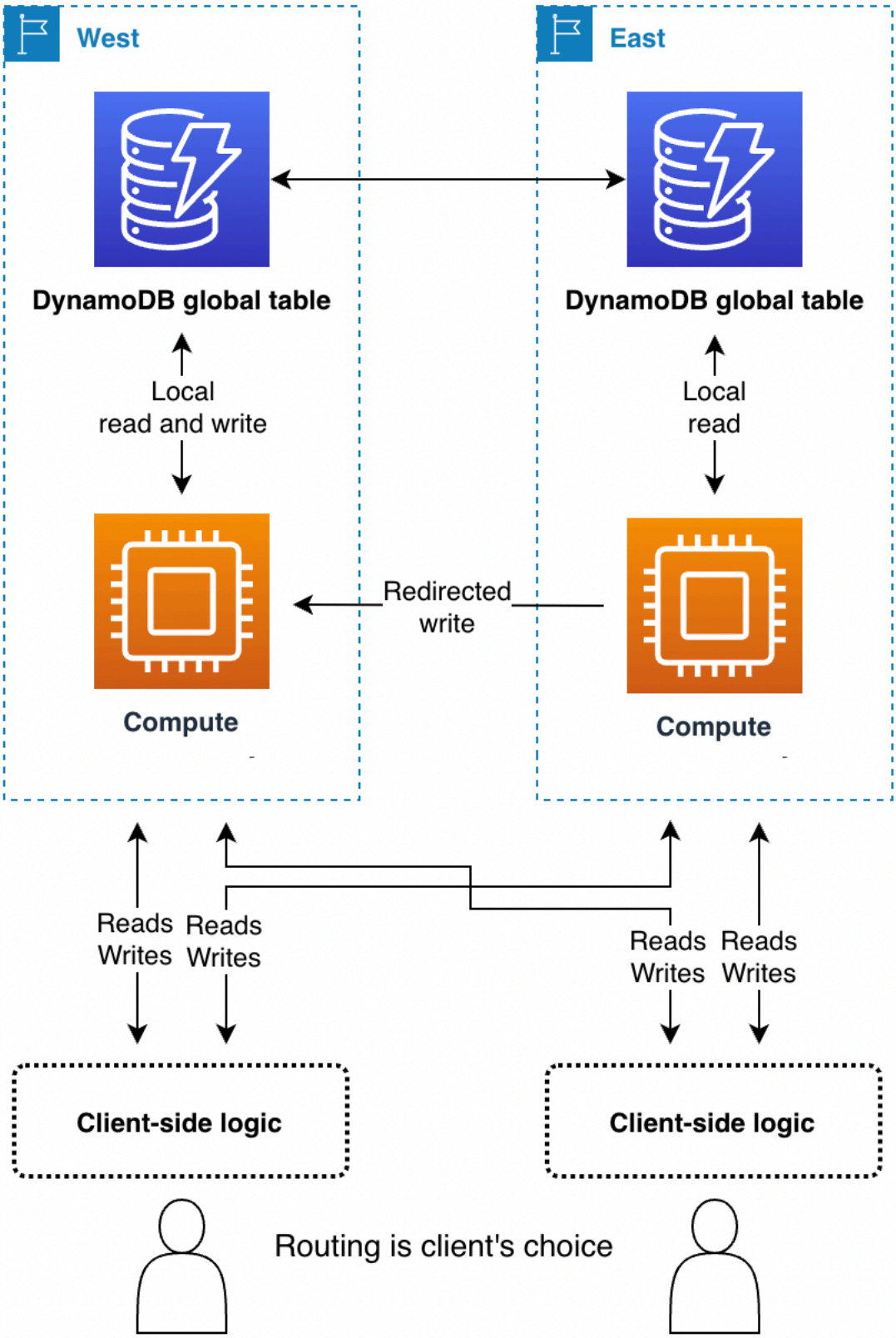
camada de computação se comunica com o DynamoDB. Ela deve fazer isso usando o endpoint local dessa região. Os dados na tabela global se replicam para todas as outras regiões participantes, e cada região tem uma pilha de serviços semelhante em torno de sua tabela do DynamoDB.

A tabela global fornece a cada pilha nas várias regiões uma cópia local dos mesmos dados. Você pode pensar em projetar uma única pilha em uma única região e antecipar a realização de chamadas remotas para o endpoint do DynamoDB de uma região secundária em caso de problemas com a tabela local do DynamoDB. Essa não é uma prática recomendada. As latências associadas ao cruzamento de regiões podem ser 100 vezes mais altas que as do acesso local. Uma série de ida e volta de cinco solicitações pode levar milissegundos quando executada localmente, mas segundos ao cruzar o globo. É melhor rotear o usuário final a outra região para processamento. Para garantir a resiliência, você precisa de replicação em várias regiões, com replicação da camada de computação e da camada de dados.

Existem várias técnicas alternativas para rotear uma solicitação do usuário final a uma região para processamento. A escolha ideal depende do modo de gravação e das considerações de failover. Esta seção aborda quatro opções: orientado pelo cliente, camada de computação, Route 53 e Global Accelerator.

Roteamento de solicitações orientado pelo cliente

Com o roteamento de solicitações orientado pelo cliente, um cliente de usuário final, como uma aplicação, uma página da web com JavaScript ou outro cliente, acompanhará os endpoints válidos da aplicação. Nesse caso, serão endpoints de aplicação, como um Amazon API Gateway, em vez de endpoints literais do DynamoDB. O cliente de usuário final usa sua própria lógica incorporada para escolher com qual região se comunicar. Essa escolha pode ser baseada em seleção aleatória, nas latências mais baixas observadas, nas medições mais altas de largura de banda observadas ou em verificações de integridade realizadas localmente.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Reads
Writes
Reads
Writes
Reads
Writes
Reads
Writes
Client-side logic
Client-side logic
Routing is client's choice

A vantagem do roteamento de solicitações orientado pelo cliente é que pode se adaptar às condições reais de tráfego público da Internet para mudar de região, caso observe redução na performance. O cliente deve estar ciente de todos os possíveis endpoints, mas o lançamento de um novo endpoint regional não ocorre com frequência.

Com o modo de gravação em qualquer região, um cliente pode selecionar unilateralmente seu endpoint preferido. Se seu acesso a uma região ficar prejudicado, o cliente poderá rotear para outro endpoint.

Com o modo de gravação em uma região, o cliente precisará de um mecanismo para rotear suas gravações para a região atualmente ativa. Isso pode ser tão básico quanto testar empiricamente qual região está aceitando gravações (observando quaisquer rejeições de gravação e recorrendo a uma alternativa) ou tão complexo quanto chamar um coordenador global para consultar o estado atual da aplicação (talvez baseado no Controlador de Recuperação de Aplicações (ARC) do Route 53, que fornece um sistema controlado por quórum de cinco regiões para manter o estado global em casos como esse). O cliente pode decidir se as leituras podem ir para qualquer região a fim de obter consistência eventual ou se devem ser roteadas para a região ativa a fim de obter uma consistência sólida. Para obter mais informações, consulte Como funciona o Route 53.

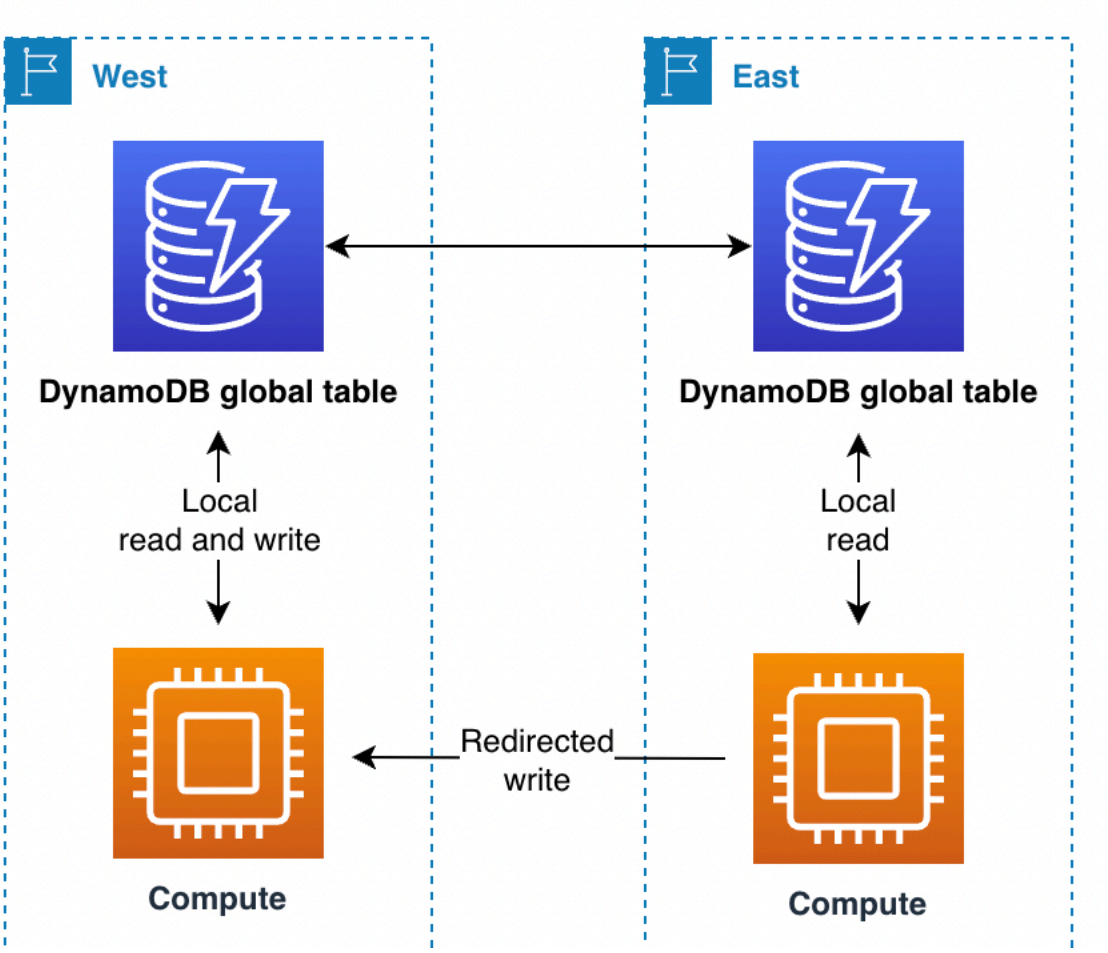
Com o modo de gravação em sua região, o cliente precisa determinar a região de origem do conjunto de dados com o qual está trabalhando. Por exemplo, se o cliente corresponder a uma conta de usuário e cada conta de usuário estiver hospedada em uma região, o cliente poderá solicitar o endpoint apropriado de um sistema de login global.

Por exemplo, uma empresa de serviços financeiros que ajuda os usuários a gerenciar suas finanças comerciais pela web pode usar tabelas globais com um modo de gravação em sua região. Cada usuário precisa fazer login em um serviço central. Esse serviço retorna as credenciais e o endpoint da região em que essas credenciais vão funcionar. As credenciais são válidas por um curto período. Depois disso, a página da web negociará automaticamente um novo login, o que oferecerá a oportunidade de redirecionar potencialmente a atividade do usuário para uma nova região.

Roteamento de solicitações na camada de computação

Com o roteamento de solicitações na camada de computação, o código em execução na camada de computação decide se deseja processar a solicitação localmente ou enviá-la para uma cópia de si mesma que está sendo executada em outra região. Quando você usa o modo de gravação em uma região, a camada de computação consegue detectar que não é a região ativa e permitir operações de leitura locais enquanto encaminha todas as operações de gravação para outra região. Esse código da camada de computação precisa estar ciente da topologia de dados e das regras

de roteamento e aplicá-las de forma confiável com base nas configurações mais recentes que especificam quais regiões estão ativas para quais dados. A pilha de software externa da região não precisa estar ciente de como as solicitações de leitura e gravação são roteadas pelo microsserviço. Em um design robusto, a região receptora valida se é a primária atual para a operação de gravação. Se não for, vai gerar um erro que indica que o estado global precisa ser corrigido. A região receptora também pode armazenar em buffer a operação de gravação por um tempo se a região primária estiver em processo de alteração. Em todos os casos, a pilha de computação em uma região grava somente em seu endpoint local do DynamoDB, mas as pilhas de computação podem se comunicar entre si.

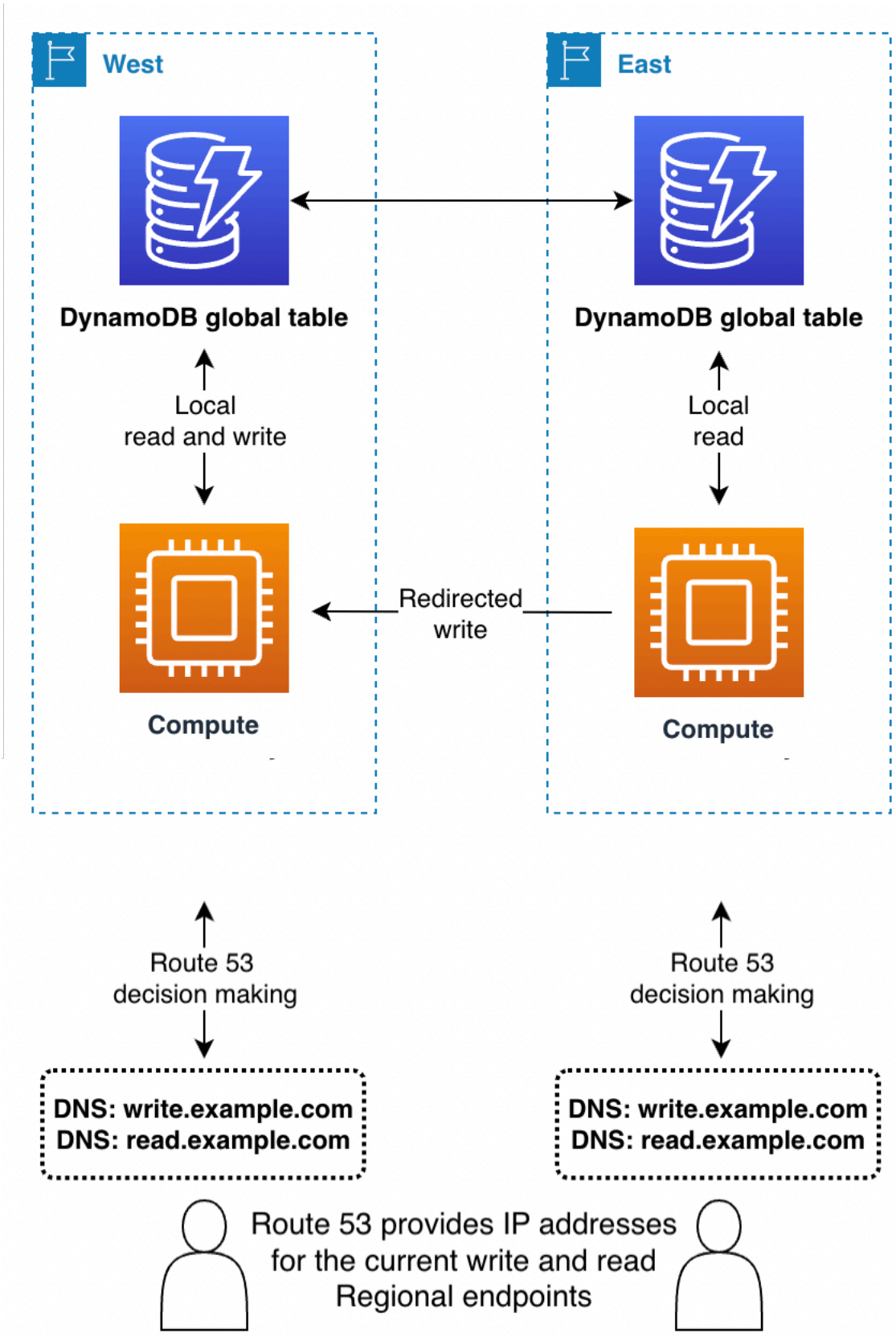


Nesse cenário, digamos que uma empresa de serviços financeiros use um modelo de primária única “follow-the-sun”. A empresa usa um sistema e uma biblioteca para esse processo de roteamento. Seu sistema geral mantém o estado global, semelhante ao controle de roteamento do ARC da AWS. A empresa usa uma tabela global para rastrear qual região é a região primária e para quando está programada a próxima mudança de primária. Todas as operações de leitura e gravação passam pela biblioteca, que coordena com seu sistema. A biblioteca permite que as operações de leitura sejam realizadas localmente, com baixa latência. Para operações de gravação, a aplicação verifica se a

região local é a região primária atual. Nesse caso, a operação de gravação é concluída diretamente. Caso contrário, a biblioteca encaminha a tarefa de gravação para a biblioteca que está na região primária atual. Essa biblioteca receptora confirma que ela também se considera como região primária e gerará um erro se não for, o que indicará um atraso na propagação com o estado global. Essa abordagem oferece um benefício de validação ao não gravar diretamente em um endpoint remoto do DynamoDB.

Roteamento de solicitações do Route 53

O Controlador de Recuperação de Aplicações (ARC) da Amazon é uma tecnologia de serviço de nomes de domínio (DNS). Com o Route 53, o cliente solicita seu endpoint pesquisando um nome de domínio DNS conhecido, e o Route 53 retorna o endereço IP correspondente aos endpoints regionais que considera mais apropriados. O Route 53 tem uma lista de políticas de roteamento que usa para determinar a região apropriada. O Route 53 também pode fazer o roteamento por failover para afastar o tráfego de regiões com falhas nas verificações de integridade.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Route 53
decision making
Route 53
decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Route 53 provides IP addresses
for the current write and read
Regional endpoints

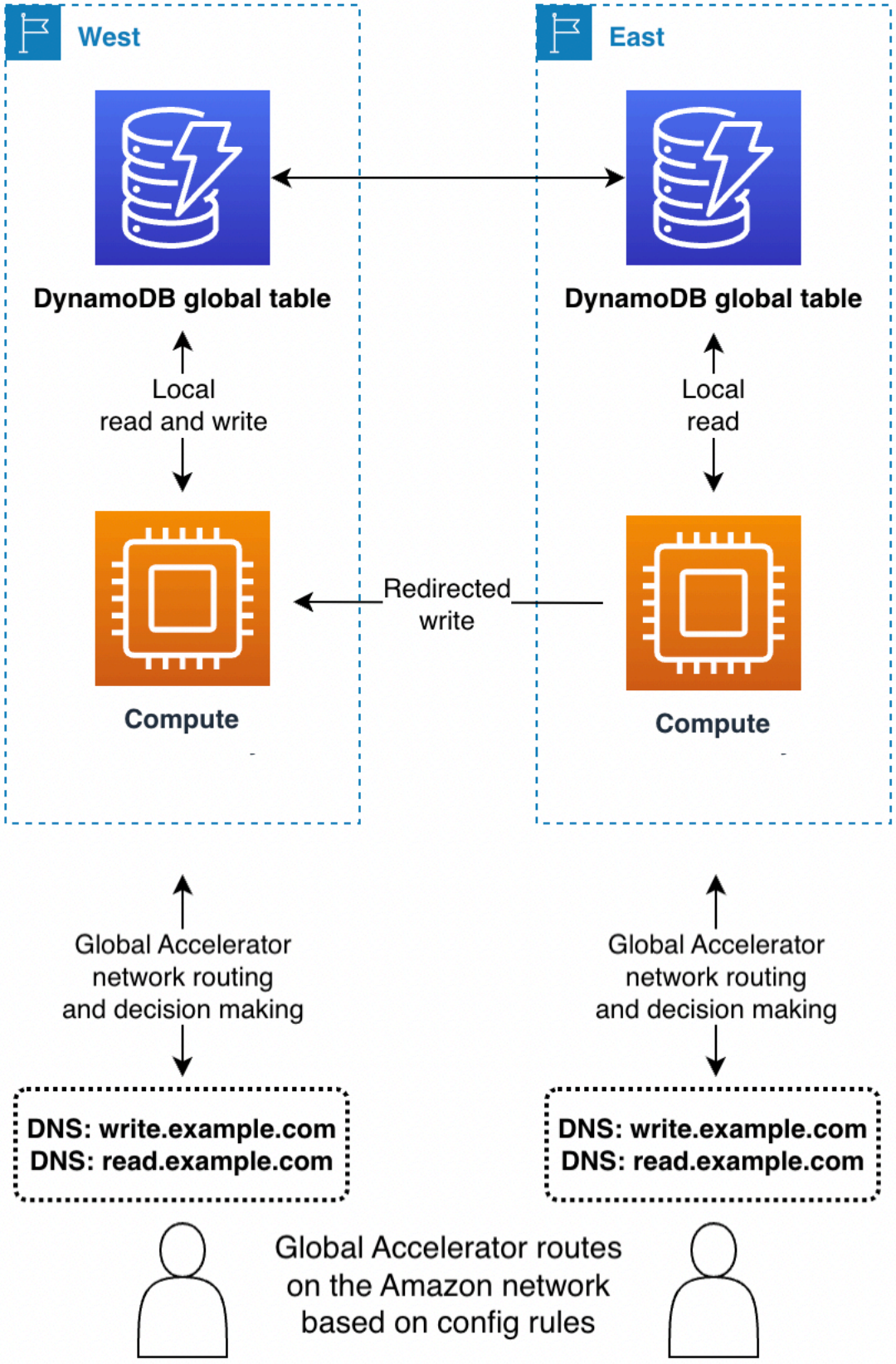
- Com o modo de gravação em qualquer região, ou quando combinado com o roteamento de solicitação na camada de computação no back-end, o Route 53 pode receber acesso total para retornar a região com base em quaisquer regras internas complexas, como a região mais próxima na rede, a mais próxima geograficamente ou qualquer outra opção.
- Com o modo de gravação em uma região, o Route 53 pode ser configurado para retornar a região atualmente ativa (usando o Route 53 ARC).

> **Note**
>
> Os clientes armazenam em cache os endereços IP na resposta do Route 53 por um tempo indicado pela configuração de vida útil (TTL) no nome de domínio. Uma TTL mais longa estende o objetivo de tempo de recuperação (RTO) para que todos os clientes reconheçam o novo endpoint. É comum um valor de 60 segundos para uso em failover. Nem todo software adere perfeitamente à expiração da TTL de DNS.

- Com o modo de gravação em sua região, é melhor evitar o Route 53, a menos que você também esteja usando o roteamento de solicitações na camada de computação.

Roteamento de solicitações no Global Accelerator

Um cliente usa o AWS Global Accelerator para pesquisar o nome de domínio conhecido no Route 53. No entanto, em vez de recuperar um endereço IP que corresponde a um endpoint regional, o cliente recebe um endereço IP estático anycast que roteia para o local da borda da AWS mais próximo. Começando nesse local da borda, todo o tráfego é roteado na rede privada da AWS e para algum endpoint (como um balanceador de carga ou o API Gateway) em uma região escolhida pelas regras de roteamento mantidas no Global Accelerator. Em comparação com o roteamento baseado nas regras do Route 53, o roteamento de solicitações no Global Accelerator tem latências mais baixas porque reduz a quantidade de tráfego na Internet pública. Além disso, como o Global Accelerator não depende da expiração da TTL de DNS para alterar as regras de roteamento, ele consegue ajustar o roteamento mais rapidamente.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Global Accelerator
network routing
and decision making
Global Accelerator
network routing
and decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Global Accelerator routes
on the Amazon network
based on config rules

- Com o modo de gravação em qualquer região, ou quando combinado com o roteamento de solicitações na camada de computação no back-end, o Global Accelerator funciona de maneira integrada. O cliente se conecta ao local da borda mais próximo e não precisa se preocupar com qual região recebe a solicitação.
- Com o modo de gravação em uma região, as regras de roteamento do Global Accelerator devem enviar solicitações para a região atualmente ativa. Você pode usar verificações de integridade que relatam artificialmente uma falha em qualquer região que não seja considerada como a região ativa pelo seu sistema global. Assim como no DNS, é possível usar um nome de domínio DNS alternativo para rotear solicitações de leitura se as solicitações puderem ser de qualquer região.
- Com o modo de gravação em sua região, é melhor evitar o Global Accelerator, a menos que você também esteja usando o roteamento de solicitações na camada de computação.

Evacuar uma região com tabelas globais do DynamoDB

Evacuar uma região é o processo de afastar a atividade de leitura e gravação dessa região. Isso geralmente envolve a atividade de gravação e, às vezes, a atividade de leitura.

Evacuar uma região ativa

Você pode decidir evacuar uma região ativa por vários motivos. A evacuação pode fazer parte da atividade comercial normal; por exemplo, se você estiver usando um modo de gravação “follow the sun” em uma região. A evacuação também pode ser por causa de uma decisão comercial de mudar a região atualmente ativa, em resposta a falhas na pilha de software fora do DynamoDB ou porque você está enfrentando problemas gerais, como latências mais altas do que o normal na região.

Com o modo de gravação em qualquer região, é fácil evacuar uma região ativa. Você pode rotear o tráfego para as regiões alternativas por meio de qualquer sistema de roteamento e permitir que as operações de gravação que já ocorreram na região evacuada sejam replicadas como de costume.

Com os modos de gravação em uma região e gravação em sua região, é necessário garantir que todas as gravações na região ativa tenham sido totalmente gravadas, processadas no fluxo e propagadas globalmente antes de iniciar as gravações na nova região ativa. Isso é necessário para garantir que as gravações future sejam comparadas à versão mais recente dos dados.

Digamos que a região A esteja ativa e a região B seja passiva (seja para a tabela inteira ou para itens hospedados na região A). O mecanismo típico para realizar uma evacuação é pausar as operações de gravação em A, esperar tempo suficiente para que essas operações sejam totalmente propagadas para B, atualizar a pilha de arquitetura para reconhecer B como ativa e, depois,

retomar as operações de gravação em B. Não há métrica que indique com certeza absoluta que a região A replicou totalmente seus dados para a região B. Se a região A estiver íntegra, pausar as operações de gravação na região A e esperar 10 vezes o valor máximo recente da métrica `ReplicationLatency` normalmente será suficiente para determinar que a replicação foi concluída. Se a região A não estiver íntegra e mostrar outras áreas com latência mais alta, escolha um múltiplo maior para o tempo de espera.

Evacuar uma região off-line

Há um caso especial a considerar: e se a região A ficar totalmente off-line sem aviso prévio? Isso é extremamente improvável, mas, mesmo assim, é prudente considerar. Se isso acontecer, todas as operações de gravação na região A que ainda não foram propagadas serão mantidas e propagadas depois que a região A voltar a ficar on-line. As operações de gravação não serão perdidas, mas a propagação será adiada indefinidamente.

A aplicação decidirá como proceder nesse caso. Para a continuidade dos negócios, talvez seja necessário prosseguir para a nova região primária B. No entanto, se um item na região B receber uma atualização enquanto houver uma propagação pendente de uma operação de gravação para esse item da região A, a propagação será suprimida no modelo último gravador vence. Qualquer atualização na região B pode suprimir uma solicitação de gravação recebida.

Com o modo de gravação em qualquer região, as leituras e gravações podem continuar na região B, confiando que os itens na região A eventualmente serão propagados para a região B e reconhecendo a possibilidade de itens perdidos até que a região A volte a ficar on-line. Quando possível, considere repetir o tráfego de gravação recente (por exemplo, usando uma fonte de eventos upstream) para preencher a lacuna da possível perda de operações de gravação e permitir que a resolução de conflitos do tipo último gravador vence suprima a eventual propagação da operação de gravação de entrada.

Com os outros modos de gravação, você deve considerar até que ponto o trabalho pode continuar com uma visão de mundo ligeiramente desatualizada. Uma pequena duração das operações de gravação, conforme monitoradas por `ReplicationLatency`, será perdida até que a região A volte a ficar on-line. Os negócios podem prosseguir? Em alguns casos de uso, sim, mas, em outros, talvez não seja possível sem mecanismos adicionais de mitigação.

Por exemplo, imagine que você precise manter um saldo de crédito disponível sem interrupções, mesmo após uma falha na região. Você pode dividir o saldo em dois itens diferentes, um hospedado na região A e outro na região B, cada um começando com metade do saldo disponível. Isso usa o modo de gravação em sua região. As atualizações transacionais processadas em cada região

são gravadas com base na cópia local do saldo. Se a região A ficar totalmente off-line, o trabalho ainda poderá prosseguir com o processamento de transações na região B, e as operações de gravação serão limitadas à parte do saldo mantida na região B. Dividir o saldo dessa forma introduz complexidades quando o saldo fica baixo ou o crédito precisa ser reajustado, mas fornece um exemplo de recuperação segura dos negócios, mesmo com operações de gravação pendentes e incertas.

Em outro exemplo, imagine que você está capturando dados de formulários da web. Você pode usar o Controle de Simultaneidade Otimista (OCC) para atribuir versões aos itens de dados e incorporar a versão mais recente ao formulário da web como um campo oculto. Em cada envio, a operação de gravação será bem-sucedida somente se a versão no banco de dados ainda corresponder à versão com a qual o formulário foi criado. Se as versões não corresponderem, o formulário da web poderá ser atualizado (ou mesclado cuidadosamente) com base na versão atual no banco de dados, e o usuário poderá prosseguir novamente. O modelo OCC geralmente protege contra a substituição e a produção de uma nova versão dos dados por outro cliente, mas também pode ajudar durante um failover, quando um cliente pode encontrar versões mais antigas dos dados.

Vamos imaginar que você está usando o carimbo de data/hora como a versão. Digamos que o formulário tenha sido criado pela primeira vez na região A às 12h, mas (após o failover) tenta gravar na região B e percebe que a versão mais recente no banco de dados é das 11h59. Nesse cenário, o cliente pode aguardar até a versão das 12h ser propagada para a região B, depois gravar em cima dessa versão, ou se basear na das 11h59 para criar uma versão das 12h01 (que, depois de ser gravada, vai suprimir a versão recebida após a recuperação da região A).

Como exemplo final, uma empresa de serviços financeiros mantém dados sobre contas de clientes e suas transações financeiras em um banco de dados do DynamoDB. Em caso de interrupção completa da região A, a empresa quer garantir que toda atividade de gravação relacionada às contas esteja totalmente disponível na região B ou, caso contrário, quer colocar suas contas em quarentena na forma em que estão até que a região A voltar a ficar on-line. Em vez de pausar toda a atividade comercial, a empresa decidiu pausar os negócios apenas na pequena fração de contas que determinaram ter transações não propagadas. Para isso, a empresa usou uma terceira região, que chamaremos de região C. Antes de processar qualquer operação de gravação na região A, a empresa incluiu um resumo sucinto dessas operações pendentes (por exemplo, uma nova contagem de transações para uma conta) na região C. Esse resumo foi suficiente para que a região B determinasse se sua visualização estava totalmente atualizada. Essa ação bloqueou efetivamente a conta desde o momento da gravação na região C até a região A aceitar as operações de gravação e a região B recebê-las. Os dados na região C não foram usados, exceto como parte de um processo de failover, após o qual a região B conseguiu comparar seus dados com a região C

para verificar se alguma conta estava desatualizada. Essas contas seriam colocadas em quarentena até que a recuperação da região A propagasse os dados parciais para a região B.

Se a região C falhasse, uma nova região D poderia ser criada para ser usada em seu lugar. Os dados na região C eram muito transitórios e, após alguns minutos, a região D já teria um registro suficientemente atualizado das operações de gravação em andamento para ser totalmente útil. Se a região B falhasse, a região A poderia continuar aceitando solicitações de gravação em cooperação com a região C. Essa empresa estava disposta a aceitar gravações com latência mais alta (em duas regiões: C e, depois, A) e teve a sorte de ter um modelo de dados em que o estado de uma conta pudesse ser resumido sucintamente.

Planejamento da capacidade de throughput para tabelas globais do DynamoDB

A migração do tráfego de uma região para outra exige uma análise cuidadosa das configurações da tabela do DynamoDB em relação à capacidade.

Algumas considerações sobre o gerenciamento da capacidade de gravação:

- Uma tabela global deve estar no modo sob demanda ou provisionada com o ajuste de escala automático ativado.
- Se provisionada com o ajuste de escala automático, as configurações de gravação (utilização mínima, máxima e pretendida) são replicadas em todas as regiões. Embora as configurações de ajuste de escala automático sejam sincronizadas, a capacidade real de gravação provisionada pode variar de maneira independente entre as regiões.
- Um motivo pelo qual você pode ver uma capacidade de gravação provisionada diferente é devido ao recurso de TTL. Ao habilitar a TTL no DynamoDB, você pode especificar um nome de atributo cujo valor indica a hora de expiração do item, em segundos no formato de hora epoch do Unix. Depois desse período, o DynamoDB poderá excluir o item sem incorrer em custos de gravação. Com tabelas globais, você pode configurar a TTL em uma região, e essa configuração é replicada automaticamente para as outras regiões associadas à tabela global. Quando um item é elegível para exclusão por meio de uma regra de TTL, esse trabalho pode ser feito em qualquer região. A operação de exclusão é executada sem consumir unidades de gravação na tabela de origem, mas as tabelas de réplica receberão uma gravação replicada dessa operação de exclusão e incorrerão em custos de unidade de gravação replicada.
- Se você estiver usando o ajuste de escala automático, garanta que a configuração de capacidade máxima de gravação provisionada seja suficientemente alta para lidar com todas as operações

de gravação, bem como com todas as possíveis operações de exclusão por TTL. O ajuste de escala automático ajusta cada região de acordo com seu consumo de gravação. As tabelas sob demanda não têm configuração de capacidade máxima de gravação provisionada, mas o limite máximo de throughput de gravação por tabela especifica a capacidade máxima de gravação sustentada que a tabela sob demanda permitirá. O limite padrão é 40 mil, mas esse limite é ajustável. Recomendamos que você o defina alto o suficiente para lidar com todas as operações de gravação (incluindo operações de gravação TTL) que a tabela sob demanda possa precisar. Esse valor deve igual em todas as regiões participantes quando você configura tabelas globais.

Algumas considerações sobre o gerenciamento da capacidade de leitura:

- É permitido que as configurações de gerenciamento da capacidade de leitura sejam diferentes entre regiões, pois presume-se que regiões diferentes possam ter padrões de leitura independentes. Ao adicionar uma réplica global a uma tabela, a capacidade da região de origem é propagada. Após a criação, você pode ajustar as configurações de capacidade de leitura, que não são transferidas para o outro lado.
- Ao usar o ajuste de escala automático do DynamoDB, certifique-se de que as configurações de capacidade máxima de leitura provisionada sejam suficientemente altas para lidar com todas as operações de leitura em todas as regiões. Durante as operações padrão, a capacidade de leitura talvez esteja distribuída entre as regiões, mas durante um failover, a tabela deve ser capaz de se adaptar automaticamente ao aumento da workload de leitura. As tabelas sob demanda não têm configuração de capacidade máxima de leitura provisionada, mas o limite máximo de throughput de leitura por tabela especifica a capacidade máxima de leitura sustentada que a tabela sob demanda permitirá. O limite padrão é 40 mil, mas esse limite é ajustável. Recomendamos que você o defina alto o suficiente para lidar com todas as operações de leitura que a tabela possa precisar se todas as operações de leitura fossem roteadas para essa única região.
- Se uma tabela em uma região não costuma receber tráfego de leitura, mas pode precisar absorver uma grande quantidade de tráfego de leitura após um failover, você pode aumentar a capacidade de leitura provisionada da tabela, esperar que a tabela termine de ser atualizada e reduzir o provisionamento da tabela novamente. Você pode deixar a tabela no modo provisionado ou alterná-la para o modo sob demanda. Isso pré-prepara a tabela para aceitar um nível mais alto de tráfego de leitura.

O ARC inclui verificações de prontidão que podem ser úteis para confirmar se as regiões do DynamoDB têm configurações de tabela e cotas de conta semelhantes, independentemente de você

usar ou não o Route 53 para rotear solicitações. Essas verificações de prontidão também podem ajudar a ajustar as cotas por conta para garantir que elas correspondam.

Lista de verificação de preparação para tabelas globais do DynamoDB e perguntas frequentes

Use a lista de verificação a seguir para decisões e tarefas ao implantar tabelas globais.

- Determine quantas e quais regiões devem participar da tabela global.
- Determine o modo de gravação da sua aplicação. Para obter mais informações, consulte Modos de gravação com tabelas globais do DynamoDB.
- Planeje sua estratégia de Roteamento de solicitações com tabelas globais do DynamoDB com base no seu modo de gravação.
- Defina seu plano de evacuação

> Evacuar uma região é o processo de afastar a atividade de leitura e gravação dessa região. Isso geralmente envolve a atividade de gravação e, às vezes, a atividade de leitura.

Evacuar uma região ativa

> Você pode decidir evacuar uma região ativa por vários motivos. A evacuação pode fazer parte da atividade comercial normal; por exemplo, se você estiver usando um modo de gravação “follow the sun” em uma região. A evacuação também pode ser por causa de uma decisão comercial de mudar a região atualmente ativa, em resposta a falhas na pilha de software fora do DynamoDB ou porque você está enfrentando problemas gerais, como latências mais altas do que o normal na região.
>
> Com o modo de gravação em qualquer região, é fácil evacuar uma região ativa. Você pode rotear o tráfego para as regiões alternativas por meio de qualquer sistema de roteamento e permitir que as operações de gravação que já ocorreram na região evacuada sejam replicadas como de costume.
>
> Com os modos de gravação em uma região e gravação em sua região, é necessário garantir que todas as gravações na região ativa tenham sido totalmente gravadas, processadas no fluxo e propagadas globalmente antes de iniciar as gravações na nova região ativa. Isso é necessário para garantir que as gravações future sejam comparadas à versão mais recente dos dados.

Digamos que a região A esteja ativa e a região B seja passiva (seja para a tabela inteira ou para itens hospedados na região A). O mecanismo típico para realizar uma evacuação é pausar as operações de gravação em A, esperar tempo suficiente para que essas operações sejam totalmente propagadas para B, atualizar a pilha de arquitetura para reconhecer B como ativa e, depois, retomar as operações de gravação em B. Não há métrica que indique com certeza absoluta que a região A replicou totalmente seus dados para a região B. Se a região A estiver íntegra, pausar as operações de gravação na região A e esperar 10 vezes o valor máximo recente da métrica `ReplicationLatency` normalmente será suficiente para determinar que a replicação foi concluída. Se a região A não estiver íntegra e mostrar outras áreas com latência mais alta, escolha um múltiplo maior para o tempo de espera.

Evacuar uma região off-line

Há um caso especial a considerar: e se a região A ficar totalmente off-line sem aviso prévio? Isso é extremamente improvável, mas, mesmo assim, é prudente considerar. Se isso acontecer, todas as operações de gravação na região A que ainda não foram propagadas serão mantidas e propagadas depois que a região A voltar a ficar on-line. As operações de gravação não serão perdidas, mas a propagação será adiada indefinidamente.

A aplicação decidirá como proceder nesse caso. Para a continuidade dos negócios, talvez seja necessário prosseguir para a nova região primária B. No entanto, se um item na região B receber uma atualização enquanto houver uma propagação pendente de uma operação de gravação para esse item da região A, a propagação será suprimida no modelo último gravador vence. Qualquer atualização na região B pode suprimir uma solicitação de gravação recebida.

Com o modo de gravação em qualquer região, as leituras e gravações podem continuar na região B, confiando que os itens na região A eventualmente serão propagados para a região B e reconhecendo a possibilidade de itens perdidos até que a região A volte a ficar on-line. Quando possível, considere repetir o tráfego de gravação recente (por exemplo, usando uma fonte de eventos upstream) para preencher a lacuna da possível perda de operações de gravação e permitir que a resolução de conflitos do tipo último gravador vence suprima a eventual propagação da operação de gravação de entrada.

Com os outros modos de gravação, você deve considerar até que ponto o trabalho pode continuar com uma visão de mundo ligeiramente desatualizada. Uma pequena duração das operações de gravação, conforme monitoradas por `ReplicationLatency`, será perdida até

que a região A volte a ficar on-line. Os negócios podem prosseguir? Em alguns casos de uso, sim, mas, em outros, talvez não seja possível sem mecanismos adicionais de mitigação.

Por exemplo, imagine que você precise manter um saldo de crédito disponível sem interrupções, mesmo após uma falha na região. Você pode dividir o saldo em dois itens diferentes, um hospedado na região A e outro na região B, cada um começando com metade do saldo disponível. Isso usa o modo de gravação em sua região. As atualizações transacionais processadas em cada região são gravadas com base na cópia local do saldo. Se a região A ficar totalmente off-line, o trabalho ainda poderá prosseguir com o processamento de transações na região B, e as operações de gravação serão limitadas à parte do saldo mantida na região B. Dividir o saldo dessa forma introduz complexidades quando o saldo fica baixo ou o crédito precisa ser reajustado, mas fornece um exemplo de recuperação segura dos negócios, mesmo com operações de gravação pendentes e incertas.

Em outro exemplo, imagine que você está capturando dados de formulários da web. Você pode usar o Controle de Simultaneidade Otimista (OCC) para atribuir versões aos itens de dados e incorporar a versão mais recente ao formulário da web como um campo oculto. Em cada envio, a operação de gravação será bem-sucedida somente se a versão no banco de dados ainda corresponder à versão com a qual o formulário foi criado. Se as versões não corresponderem, o formulário da web poderá ser atualizado (ou mesclado cuidadosamente) com base na versão atual no banco de dados, e o usuário poderá prosseguir novamente. O modelo OCC geralmente protege contra a substituição e a produção de uma nova versão dos dados por outro cliente, mas também pode ajudar durante um failover, quando um cliente pode encontrar versões mais antigas dos dados.

Vamos imaginar que você está usando o carimbo de data/hora como a versão. Digamos que o formulário tenha sido criado pela primeira vez na região A às 12h, mas (após o failover) tenta gravar na região B e percebe que a versão mais recente no banco de dados é das 11h59. Nesse cenário, o cliente pode aguardar até a versão das 12h ser propagada para a região B, depois gravar em cima dessa versão, ou se basear na das 11h59 para criar uma versão das 12h01 (que, depois de ser gravada, vai suprimir a versão recebida após a recuperação da região A).

Como exemplo final, uma empresa de serviços financeiros mantém dados sobre contas de clientes e suas transações financeiras em um banco de dados do DynamoDB. Em caso de interrupção completa da região A, a empresa quer garantir que toda atividade de gravação relacionada às contas esteja totalmente disponível na região B ou, caso contrário, quer colocar suas contas em quarentena na forma em que estão até que a região A voltar a ficar on-line. Em vez de pausar toda a atividade comercial, a empresa decidiu pausar os negócios apenas

> na pequena fração de contas que determinaram ter transações não propagadas. Para isso, a empresa usou uma terceira região, que chamaremos de região C. Antes de processar qualquer operação de gravação na região A, a empresa incluiu um resumo sucinto dessas operações pendentes (por exemplo, uma nova contagem de transações para uma conta) na região C. Esse resumo foi suficiente para que a região B determinasse se sua visualização estava totalmente atualizada. Essa ação bloqueou efetivamente a conta desde o momento da gravação na região C até a região A aceitar as operações de gravação e a região B recebê-las. Os dados na região C não foram usados, exceto como parte de um processo de failover, após o qual a região B conseguiu comparar seus dados com a região C para verificar se alguma conta estava desatualizada. Essas contas seriam colocadas em quarentena até que a recuperação da região A propagasse os dados parciais para a região B.
>
> Se a região C falhasse, uma nova região D poderia ser criada para ser usada em seu lugar. Os dados na região C eram muito transitórios e, após alguns minutos, a região D já teria um registro suficientemente atualizado das operações de gravação em andamento para ser totalmente útil. Se a região B falhasse, a região A poderia continuar aceitando solicitações de gravação em cooperação com a região C. Essa empresa estava disposta a aceitar gravações com latência mais alta (em duas regiões: C e, depois, A) e teve a sorte de ter um modelo de dados em que o estado de uma conta pudesse ser resumido sucintamente.

com base no modo de gravação e na estratégia de roteamento.

- Capture métricas sobre integridade, latência e erros em cada região. Para obter uma lista das métricas do DynamoDB, consulte a postagem do blog da AWS Monitoramento do Amazon DynamoDB para conscientização operacional para obter uma lista de métricas a serem observadas. Você também deve usar canários sintéticos (solicitações artificiais projetadas para detectar falhas, nomeadas em referência aos canários nas minas de carvão), bem como a observação ao vivo do tráfego de clientes. Nem todos os problemas aparecerão nas métricas do DynamoDB.
- Defina alarmes para qualquer aumento contínuo em `ReplicationLatency`. Um aumento pode indicar uma configuração incorreta acidental na qual a tabela global tem diferentes configurações de gravação em regiões diferentes, o que leva à falha nas solicitações replicadas e ao aumento das latências. Também pode indicar que há uma interrupção regional. Um bom exemplo é gerar um alerta se a média recente exceder 180 mil milissegundos. Você também pode observar a queda de `ReplicationLatency` para 0, o que indica uma replicação paralisada.
- Atribua configurações máximas de leitura e gravação suficientes para cada tabela global.

- Identifique os motivos para evacuar uma região com antecedência. Se a decisão envolver julgamento humano, documente todas as considerações. Esse trabalho deve ser feito com cuidado e com antecedência, não sob pressão.
- Mantenha um runbook para cada ação que deve ser tomada ao evacuar uma região. Normalmente, as tabelas globais exigem muito pouco trabalho, mas mover o restante da pilha pode ser complexo.

> **Note**
>
> É uma prática recomendada depender somente de operações do plano de dados e não das operações do ambiente de gerenciamento, pois algumas operações do ambiente de gerenciamento podem ser comprometidas durante falhas na região.

  Para obter mais informações, consulte a postagem do blog da AWS Desenvolver aplicações resilientes com tabelas globais do Amazon DynamoDB: parte 4.
- Teste todos os aspectos do runbook periodicamente, incluindo evacuações de região. Um runbook não testado é um runbook não confiável.
- Considere usar o Hub de Resiliência para avaliar a resiliência de toda a sua aplicação (incluindo tabelas globais). Ele fornece uma visão abrangente do status geral de resiliência do seu portfólio de aplicações por meio de seu painel.
- Considere usar as verificações de prontidão do ARC para avaliar a configuração atual da aplicação e rastrear quaisquer desvios das práticas recomendadas.
- Ao escrever uma verificação de integridade para uso com o Route 53 ou o Global Accelerator, não basta apenas enviar um ping para verificar se o endpoint do DynamoDB está ativo. Isso não abrange os vários modos de falha, como erros de configuração do IAM, problemas de implantação de código, falha na pilha fora do DynamoDB, latências de leitura ou gravação acima da média e assim por diante. É melhor realizar um conjunto de chamadas que exerçam um fluxo completo do banco de dados.

Perguntas frequentes (FAQ) sobre a implantação de tabelas globais

Quais são alguns princípios úteis para o uso geral das tabelas globais do DynamoDB?

As tabelas globais do DynamoDB têm poucos botões de controle, mas ainda exigem diversas considerações. Você deve determinar seu modo de gravação, modelo de roteamento e processos

de evacuação. Você deve instrumentar sua aplicação em todas as regiões e se preparar para ajustar o roteamento ou realizar uma evacuação para manter a integridade global. A recompensa é ter um conjunto de dados distribuído globalmente com leituras e gravações de baixa latência e um acordo de serviço de 99,999%.

Qual é o preço das tabelas globais?

O preço de uma tabela tradicional do DynamoDB é definido em unidades de capacidade de gravação (WCUs, para tabelas provisionadas) ou unidades de solicitação de gravação (WRUs, para tabelas sob demanda). Se você gravar um item de 5 KB, serão cobradas 5 unidades. O preço de uma gravação em uma tabela global é definido em unidades de capacidade de gravação replicada (rWCUs, para tabelas provisionadas) ou unidades de solicitação de gravação replicada (rWRUs, para tabelas sob demanda).

As rWCUs e rWRUs incluem o custo da infraestrutura de streaming necessária para gerenciar a replicação.

As unidades de gravação replicada são cobradas em todas as regiões em que o item é gravado diretamente ou gravado em replicação.

A gravação em um índice secundário global (GSI) é considerada uma gravação local e usa unidades de gravação regulares.

Não há nenhuma capacidade reservada disponível para rWCUs no momento. A compra de capacidade reservada ainda pode ser benéfica para tabelas com GSIs que consomem unidades de gravação.

O bootstrap inicial ao adicionar uma nova região a uma tabela global é cobrado como uma restauração por GB de dados restaurados, mais taxas de transferência de dados entre regiões.

Quais regiões são compatíveis com tabelas globais?

O Global Tables versão 2019.11.21 (atual) está disponível na maioria das regiões.

Como os GSIs são tratados com tabelas globais?

Em Global Tables versão 2019.11.21 (atual), ao criar um GSI em uma região, ele é criado e preenchido automaticamente em outras regiões participantes.

Como faço para interromper a replicação de uma tabela global?

Você pode excluir uma tabela de réplica da mesma forma que excluiria qualquer outra tabela. A exclusão de uma tabela global interromperá a replicação nessa região e excluirá a cópia da tabela

mantida nessa região. No entanto, você não pode interromper a replicação enquanto mantém cópias da tabela como entidades independentes nem pode pausar a replicação.

Como o DynamoDB Streams interage com as tabelas globais?

Cada tabela global produz um fluxo independente com base em todas as gravações, independentemente de onde começaram. Você pode optar por consumir o fluxo do DynamoDB em uma região ou em todas as regiões (de forma independente). Se você quiser processar operações de gravações locais, mas não gravações replicadas, poderá adicionar seu próprio atributo de região a cada item para identificar a região de gravação. Depois, você pode usar um filtro de eventos do Lambda para chamar a função do Lambda somente para gravações na região local. Isso ajuda nas operações de inserção e atualização, mas infelizmente não ajuda nas operações de exclusão.

Como as tabelas globais lidam com as transações?

As operações transacionais fornecem garantia de atomicidade, consistência, isolamento e durabilidade (ACID) somente na região em que a operação de gravação ocorreu originalmente. As transações não são compatíveis entre regiões em tabelas globais. Por exemplo, se você tiver uma tabela global com réplicas nas regiões Leste dos EUA (Ohio) e Oeste dos EUA (Oregon) e realizar uma operação `TransactWriteItems` na região Leste dos EUA (Ohio), poderá observar transações parcialmente concluídas na região Oeste dos EUA (Oregon) à medida que as alterações forem replicadas. As alterações só são replicadas para outras regiões quando forem confirmadas na região de origem.

Como as tabelas globais interagem com o cache do DynamoDB Accelerator (DAX)?

As tabelas globais ignoram o DAX ao atualizar o DynamoDB diretamente, por isso o DAX não sabe que está armazenando dados obsoletos. O cache do DAX só é atualizado quando a TTL do cache expira.

As etiquetas nas tabelas são propagadas?

Não, as etiquetas não são propagadas automaticamente.

Devo fazer backup de tabelas em todas as regiões ou em apenas uma?

A resposta depende da finalidade do backup. Se sua intenção é garantir a durabilidade dos dados, o DynamoDB já fornece essa proteção. O serviço garante a durabilidade. Se você quiser manter um snapshot para registros históricos (por exemplo, para atender aos requisitos regulatórios), o backup em uma única região deverá ser suficiente. Você pode copiar o backup para outras regiões usando

o AWS Backup. Se você quiser recuperar dados excluídos ou modificados incorretamente, use a recuperação para um ponto no tempo (PITR) do DynamoDB em uma região.

Como faço para implantar tabelas globais usando o AWS CloudFormation?

O CloudFormation representa uma tabela do DynamoDB e uma tabela global como dois recursos separados: `AWS::DynamoDB::Table` e `AWS::DynamoDB::GlobalTable`. Uma abordagem é criar todas as tabelas que possam ser globais usando a estrutura `GlobalTable`. Você pode mantê-las como tabelas independentes inicialmente, depois adicionar regiões, se necessário.

No CloudFormation, cada tabela global é controlada por uma única pilha, em uma só região, independentemente do número de réplicas. Quando seu modelo for implantado, o CloudFormation criará e atualizará todas as réplicas como parte de uma única operação de pilha. Você não deve implantar o mesmo atributo AWS::DynamoDB::GlobalTable em várias regiões. Isso não é compatível e resultará em erros. Se você implantar seu modelo de aplicação em várias regiões, poderá usar condições para criar o recurso `AWS::DynamoDB::GlobalTable` em uma única região. Se quiser, você poderá optar por definir recursos `AWS::DynamoDB::GlobalTable` em uma pilha separada da pilha de aplicações e garantir que ela seja implantada apenas em uma região.

Se você tiver uma tabela regular e quiser convertê-la em uma tabela global enquanto a mantém gerenciada pelo CloudFormation, defina a política de exclusão como Reter, remova a tabela da pilha, converta a tabela em uma tabela global no console e importe a tabela global como um novo recurso para a pilha.

No momento, a replicação entre contas não é compatível.

Práticas recomendadas para gerenciar o ambiente de gerenciamento no DynamoDB

> **Note**
>
> O DynamoDB está introduzindo um controle de utilização do ambiente de gerenciamento de 2,5 mil solicitações por segundo com a opção de uma nova tentativa. Veja mais detalhes a seguir.

As operações do ambiente de gerenciamento do DynamoDB permitem gerenciar tabelas do DynamoDB, bem como objetos que dependem de tabelas, como índices. Para obter mais informações sobre essas operações, consulte a Ambiente de gerenciamento.

Em algumas circunstâncias, talvez seja necessário realizar ações e usar os dados retornados pelas chamadas do ambiente de gerenciamento como parte de sua lógica empresarial. Por exemplo, talvez você precise conhecer o valor de `ProvisionedThroughput` retornado por `DescribeTable`. Nessas circunstâncias, siga estas práticas recomendadas:

- Não consulte excessivamente o ambiente de gerenciamento do DynamoDB.
- Não misture chamadas do ambiente de gerenciamento e do plano de dados no mesmo código.
- Lide com os controles de utilização nas solicitações do ambiente de gerenciamento e tente novamente com um recuo.
- Invoque e acompanhe as alterações em um recurso específico de um único cliente.
- Em vez de recuperar dados da mesma tabela várias vezes em intervalos curtos, armazene os dados em cache para processamento.

Práticas recomendadas para usar operações de dados em massa no DynamoDB

O DynamoDB é compatível com operações em lote, como `BatchWriteItem`, que permitem realizar até 25 solicitações `PutItem` e `DeleteItem` em conjunto. No entanto, `BatchWriteItem` não permite operações `UpdateItem`. Quando se trata de atualizações em massa, a diferença está nos requisitos e na natureza da atualização. É possível usar outras APIs do DynamoDB, como `TransactWriteItems`, para lotes de até 100. Quando mais itens estão envolvidos, você pode usar serviços como o AWS Glue, o Amazon EMR, o AWS Step Functions ou usar scripts e ferramentas personalizados, como o DynamoDB-shell, para atualizações em massa.

Tópicos

- Atualização condicional em lote
- Operações em massa eficientes

Atualização condicional em lote

O DynamoDB é compatível com operações em lote, como `BatchWriteItem`, que permitem realizar até 25 solicitações `PutItem` e `DeleteItem` em um único lote. No entanto, `BatchWriteItem` não permite operações `UpdateItem` nem expressões condicionais. Como solução alternativa, você pode usar outras APIs do DynamoDB, como `TransactWriteItems`, para lotes de até 100.

Quando mais itens estão envolvidos e uma grande parte dos dados precisa ser alterada, é possível usar serviços como o AWS Glue, o Amazon EMR, o AWS Step Functions ou usar scripts e ferramentas personalizados, como o DynamoDB-shell, para atualizações em massa eficientes.

Quando usar esse padrão

- O DynamoDB-shell não é compatível para caso de uso de produção.
- `TransactWriteItems`: até 100 atualizações individuais com ou sem condições, executadas como um pacote ACID de "tudo ou nada". As chamadas de `TransactWriteItems` também podem ser fornecidas com um `ClientRequestToken` se a aplicação exigir idempotência, o que significa que várias chamadas idênticas têm o mesmo efeito de uma única chamada. Isso garante que você não execute a mesma transação várias vezes e acabe com um estado incorreto dos dados.

  Desvantagem: throughput adicional é consumido. 2 WCUs por gravação de 1 KB em vez do padrão de 1 WCU por gravação de 1 KB.
- PartiQL `BatchExecuteStatement`: até 25 atualizações com ou sem condições. `BatchExecuteStatement` sempre retorna uma resposta bem-sucedida à solicitação geral e também retorna uma lista de respostas individuais da operação que preserva a ordem.

  Desvantagem: para lotes maiores, é necessária uma lógica adicional do lado do cliente para distribuir solicitações em lotes de 25. As respostas de erro individuais precisam ser consideradas para determinar a estratégia de nova tentativa.

Exemplos de código

Esses exemplos de código usam a biblioteca boto3, que é o AWS SDK para Python. Os exemplos pressupõem que você tenha o boto3 instalado e configurado com as credenciais da AWS apropriadas.

Suponha um banco de dados de inventário para um fornecedor de eletrodomésticos que tenha vários armazéns em cidades europeias. Como é final do verão, o fornecedor gostaria de liquidar os ventiladores de mesa para abrir espaço para outros estoques. O fornecedor deseja oferecer um desconto no preço de todos os ventiladores de mesa fornecidos pelos armazéns na Itália, mas somente se eles tiverem um estoque de reserva de 20 ventiladores de mesa. A tabela do DynamoDB é chamada inventário, ela tem um esquema de chaves com chave de partição sku, que é um identificador exclusivo para cada produto, e uma chave de classificação armazém, que é um identificador de um armazém.

O código Python a seguir demonstra como realizar essa atualização condicional em lote usando a chamada de API `BatchExecuteStatement`.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.batch_execute_statement(
        Statements=[
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITTUR1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM5'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN3'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
```

```
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

A execução produz a saída abaixo nos dados de amostra:

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '40'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '35'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '33'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '38'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '30'}, 'sku': {'S': 'F123'}}]
```

Como essa é uma operação limitada para um sistema interno, os requisitos de idempotência não foram considerados. É possível colocar barreiras de proteção adicionais, como permitir a atualização de preços somente se o preço for maior que 35 e menor que 40, para tornar as atualizações mais robustas.

Como alternativa, podemos realizar a mesma operação de atualização em lote usando `TransactWriteItems` no caso de requisitos mais rígidos de idempotência e ACID. No entanto, é importante lembrar que ou todas as operações no pacote de transações são concluídas ou o pacote inteiro falha.

Vamos supor um caso em que haja uma onda de calor na Itália e a demanda por ventiladores de mesa tenha aumentado significativamente. O fornecedor deseja aumentar o custo do ventilador de mesa que sai de cada armazém na Itália em 20 euros, mas o órgão regulador só permite esse aumento de custo se o custo atual for inferior a 70 euros em todo o estoque. É essencial que o

preço seja atualizado em todo o estoque de uma só vez e somente uma vez, e apenas se o custo for inferior a 70 euros em cada um dos armazéns.

O código Python a seguir demonstra como realizar essa atualização em lote usando a chamada de API `TransactWriteItems`.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.transact_write_items(
        ClientRequestToken='UUIDAWS124',
        TransactItems=[
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITTUR1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM5'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
```

```
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
           {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN3'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
       ],
       ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

A execução produz a saída abaixo nos dados de amostra:

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '60'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '55'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '53'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '55'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '50'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '80'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '75'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '73'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '75'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '78'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '78'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '70'}, 'sku': {'S': 'F123'}}]
```

Existem várias abordagens para realizar atualizações em lote no DynamoDB. A abordagem adequada depende de fatores como requisitos de ACID e/ou idempotência, número de itens a serem atualizados e familiaridade com as APIs.

Operações em massa eficientes

Quando usar esse padrão

Esses padrões são úteis para realizar com eficiência atualizações em massa nos itens do DynamoDB.

- O DynamoDB-shell não é compatível para caso de uso de produção.
- `TransactWriteItems`: até 100 atualizações individuais com ou sem condições, executadas como um pacote ACID de “tudo ou nada”

  Desvantagem: é consumido um throughput adicional, 2 WCUs por gravação de 1 KB.
- PartiQL `BatchExecuteStatement`: até 25 atualizações com ou sem condições

  Desvantagem: é necessária lógica adicional para distribuir solicitações em lotes de 25.
- AWS Step Functions: operações em massa com taxa limitada para desenvolvedores familiarizados com o AWS Lambda.

  Desvantagem: o tempo de execução é inversamente proporcional ao limite de taxa. Limitado pelo tempo limite máximo da função do Lambda. A funcionalidade implica que as alterações de dados que ocorrem entre a leitura e a gravação podem ser substituídas. Consulte mais informações em Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.
- AWS Glue e Amazon EMR: operação em massa com taxa limitada e paralelismo gerenciado. Para aplicações ou atualizações que não são sensíveis ao tempo, essas opções podem ser executadas em segundo plano, consumindo apenas uma pequena porcentagem do throughput. Ambos os serviços usam o emr-dynamodb-connector para realizar operações do DynamoDB. Esses serviços realizam uma grande leitura seguida por uma grande gravação de itens atualizados com a opção de limitar a taxa.

  Desvantagem: o tempo de execução é inversamente proporcional ao limite de taxa. A funcionalidade inclui que as alterações de dados que ocorrem entre a leitura e a gravação podem ser substituídas. Não é possível ler de índices secundários globais (GSIs). Consulte Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.
- DynamoDB Shell: operações em massa com taxa limitada usando consultas semelhantes a SQL. É possível ler de GSIs para ter melhor eficiência.

  Desvantagem: o tempo de execução é inversamente proporcional ao limite de taxa. Consulte Rate limited bulk operations in DynamoDB Shell.

Uso do padrão

Atualizações em massa podem ter implicações significativas de custo, especialmente se você usar o modo de throughput sob demanda. Haverá uma compensação entre velocidade e custo se você usar o modo de throughput provisionado. Definir o parâmetro de limite de taxa de forma muito restrita pode levar a um tempo de processamento muito longo. É possível determinar aproximadamente a velocidade da atualização usando o tamanho médio do item e o limite de taxa.

Como alternativa, é possível determinar a quantidade de throughput necessária para o processo com base na duração esperada do processo de atualização e no tamanho médio do item. As referências do blog compartilhadas com cada padrão fornecem detalhes sobre a estratégia, a implementação e as limitações do uso do padrão. Consulte mais informações em Cost-effective bulk processing with Amazon DynamoDB.

Existem várias abordagens para realizar atualizações em massa em uma tabela ativa do DynamoDB. A abordagem adequada depende de fatores como requisitos de ACID e/ou idempotência, número de itens a serem atualizados e familiaridade com as APIs. É importante considerar a relação entre custo e tempo. A maioria das abordagens discutidas acima oferece uma opção para limitar o throughput usado pelo trabalho de atualização em massa.

Práticas recomendadas para implementar o controle de versão no DynamoDB

Em sistemas distribuídos como o DynamoDB, o controle de versão de itens usando o bloqueio positivo evita atualizações conflitantes. Ao rastrear as versões dos itens e usar gravações condicionais, as aplicações podem gerenciar modificações simultâneas, garantindo a integridade dos dados em ambientes de alta simultaneidade.

O bloqueio positivo é uma estratégia usada para garantir que as modificações dos dados sejam aplicadas corretamente sem conflitos. Em vez de bloquear os dados quando eles são lidos (como no bloqueio negativo), o bloqueio positivo verifica se os dados foram alterados antes de gravá-los novamente. No DynamoDB, isso é obtido por meio de uma forma de controle de versão, em que cada item inclui um identificador que aumenta a cada atualização. Ao atualizar um item, a operação só será bem-sucedida se esse identificador corresponder ao esperado pela aplicação.

Quando usar esse padrão

Esse padrão é útil nos seguintes cenários:

- Vários usuários ou processos podem tentar atualizar o mesmo item simultaneamente.
- Garantir a integridade e a consistência dos dados é fundamental.
- É necessário evitar a sobrecarga e a complexidade do gerenciamento de bloqueios distribuídos.

Os exemplos incluem:

- Aplicações de comércio eletrônico em que os níveis de estoque são atualizados com frequência.
- Plataformas colaborativas em que vários usuários editam os mesmos dados.
- Sistemas financeiros em que os registros de transações devem permanecer consistentes.

Desvantagens

Embora o bloqueio positivo e as verificações condicionais forneçam uma integridade robusta dos dados, eles apresentam as seguintes desvantagens:

Conflitos de simultaneidade

: Em ambientes de alta simultaneidade, a probabilidade de conflitos aumenta, podendo gerar mais tentativas e custos de gravação.

Complexidade de implementação

: Adicionar controle de versão aos itens e lidar com verificações condicionais pode adicionar complexidade à lógica da aplicação.

Mais custos operacionais indiretos com armazenamento

: Armazenar números de versão para cada item aumenta um pouco os requisitos de armazenamento.

Padrão de design

Para implementar esse padrão, o esquema do DynamoDB deve incluir um atributo de versão para cada item. Aqui está um design de esquema simples:

- Chave de partição: um identificador exclusivo para cada item (por exemplo: `ItemId`).

- Atributos:
  - `ItemId`: o identificador exclusivo do item.
  - `Version`: um número inteiro que representa o número da versão do item.
  - `QuantityLeft`: o estoque restante do item.

Quando um item é criado pela primeira vez, o atributo `Version` é definido como 1. Com cada atualização, o número da versão é incrementado em 1.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	1	10
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

Uso do padrão

Para implementar esse padrão, siga estas etapas no fluxo da aplicação:

1. Leia a versão atual do item.

   Recupere o item atual do DynamoDB e leia o número de versão.

   ```
   def get_document(item_id):
       response = table.get_item(Key={'ItemID': item_id})
       return response['Item']

   document = get_document('Bananas')
   current_version = document['Version']
   ```

2. Incremente o número da versão na lógica da aplicação. Essa será a versão esperada para a atualização.

   ```
   new_version = current_version + 1
   ```

3. Tente atualizar o item usando uma expressão condicional para garantir que o número da versão corresponda.

```
def update_document(item_id, qty_bought, current_version):
    try:
        response = table.update_item(
            Key={'ItemID': item_id},
            UpdateExpression="set #qty = :qty, Version = :v",
            ConditionExpression="Version = :expected_v",
            ExpressionAttributeNames={
                '#qty': 'QuantityLeft'
            },
            ExpressionAttributeValues={
                ':qty': qty_bought,
                ':v': current_version + 1,
                ':expected_v': current_version
            },
            ReturnValues="UPDATED_NEW"
        )
        return response
    except ClientError as e:
        if e.response['Error']['Code'] == 'ConditionalCheckFailedException':
            print("Update failed due to version conflict.")
        else:
            print("Unexpected error: %s" % e)
        return None

update_document('Bananas', 2, new_version)
```

Se a atualização for bem-sucedida, a QuantityLeft do item será reduzida em 2.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	2	8
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

4. Gerencie os conflitos se eles ocorrerem.

   Se ocorrer um conflito (por exemplo, outro processo atualizou o item desde a última vez que você o leu), gerencie o conflito adequadamente, como repetindo a operação ou alertando o usuário.

   Isso exigirá uma leitura adicional do item para cada nova tentativa, portanto, limite o número total de tentativas permitidas antes de falhar completamente o ciclo de solicitação.

```
def update_document_with_retry(item_id, new_data, retries=3):
    for attempt in range(retries):
        document = get_document(item_id)
        current_version = document['Version']

        result = update_document(item_id, qty_bought, current_version)

        if result is not None:
            print("Update succeeded.")
            return result
        else:
            print(f"Retrying update... ({attempt + 1}/{retries})")

    print("Update failed after maximum retries.")
    return None

update_document_with_retry('Bananas', 2)
```

A implementação do controle de versão de itens usando o DynamoDB com bloqueio positivo e verificações condicionais é um padrão poderoso para garantir a integridade dos dados em aplicações distribuídas. Embora apresente alguma complexidade e possíveis desvantagens de desempenho, é inestimável em cenários que exigem um controle robusto de simultaneidade. Ao projetar cuidadosamente o esquema e implementar as verificações necessárias na lógica da aplicação, é possível gerenciar com eficiência as atualizações simultâneas e manter a consistência de dados.

Orientações e estratégias adicionais sobre formas de implementar o controle de versão dos dados do DynamoDB podem ser encontradas no Blog de banco de dados da AWS.

Práticas recomendadas para entender os relatórios de uso e faturamento da AWS no DynamoDB

Este documento explica os códigos de faturamento `UsageType` para cobranças relacionadas ao DynamoDB.

A AWS fornece Relatórios de Custos e Uso (CUR) que contêm dados dos serviços usados. É possível usar o AWS Cost and Usage Report para publicar relatórios de faturamento no Amazon S3 em formato CSV. Ao configurar o CUR, é possível optar por dividir os períodos por hora, dia ou mês, e escolher se deseja dividir o uso por ID de recurso ou não. Para ter mais detalhes sobre a geração de CUR, consulte Criar relatórios de custos e uso

Na exportação de CSV, você encontrará atributos relevantes listados para cada linha. Veja a seguir exemplos de atributos que podem ser incluídos:

- lineitem/UsageStartDate: a data e a hora de início do item de linha em UTC, inclusive.
- lineitem/UsageEndDate: a data e a hora de término do item de linha correspondente em UTC, excluindo-se esse momento.
- lineitem/ProductCode: para o DynamoDB, será “AmazonDynamoDB”.
- lineitem/UsageType: código de descrição específico para o tipo de uso, conforme enumerado neste documento
- lineitem/Operation: um nome que fornece contexto à cobrança, como o nome da operação que gerou a cobrança (opcional).
- lineitem/ResourceId: o identificador do recurso que gerou o uso. Disponível se o CUR incluir um detalhamento por ID do recurso.
- lineitem/UsageAmount: a quantidade de uso incorrida durante um período específico.
- lineitem/UnblendedCost: o custo desse uso.
- lineitem/LineItemDescription: descrição textual do item de linha.

Para ter mais informações sobre o dicionário de dados do CUR, consulte Cost and Usage Report (CUR) 2.0. Observe que os nomes exatos variam de acordo com o contexto.

`UsageType` é uma string com um valor como `ReadCapacityUnit-Hrs`, `USW2-ReadRequestUnits`, `EU-WriteCapacityUnit-Hrs` ou `USE1-TimedPITRStorage-ByteHrs`. Cada tipo de uso começa com um prefixo de região opcional. Se ausente, isso indica a região us-

east-1. Se presente, a tabela abaixo associa o código curto da região de faturamento ao código e ao nome da região convencionais.

Por exemplo, o uso chamado `USW2-ReadRequestUnits` indica unidades de solicitação de leitura consumidas em us-west-2.

Código da região de faturamento	Código da região	Nome da região
AFS1	af-south-1	África (Cidade do Cabo)
APE1	ap-east-1	Ásia-Pacífico (Hong Kong)
APN1	ap-northeast-1	Ásia-Pacífico (Tóquio)
APN2	ap-northeast-2	Ásia-Pacífico (Seul)
APN3	ap-northeast-3	Ásia-Pacífico (Osaka)
APS1	ap-southeast-1	Ásia-Pacífico (Singapura)
APS2	ap-southeast-2	Ásia-Pacífico (Sydney)
APS3	ap-south-1	Ásia-Pacífico (Mumbai)
APS4	ap-southeast-3	Ásia-Pacífico (Jacarta)
APS5	ap-south-2	Ásia-Pacífico (Hyderabad)
APS6	ap-southeast-4	Ásia-Pacífico (Melbourne)
CAN1	ca-central-1	Canadá (Central)
UE	eu-west-1	Europa (Irlanda)
EUC1	eu-central-1	Europa (Frankfurt)
EUC2	eu-central-2	Europa (Zurique)
EUN1	eu-north-1	Europa (Estocolmo)
EUS1	eu-south-1	Europa (Milão)

Código da região de faturamento	Código da região	Nome da região
EUS2	eu-south-2	Europa (Espanha)
EUW1	eu-west-1	Europa (Irlanda)
EUW2	eu-west-2	Europa (Londres)
EUW3	eu-west-3	Europa (Paris)
ILC1	Il-central-1	Israel (Tel Aviv)
MEC1	me-central-1	Oriente Médio (Emirados Árabes Unidos)
MES1	me-south-1	Oriente Médio (Bahrein)
SAE1	sa-east-1	América do Sul (São Paulo)
USE1 (padrão)	us-east-1	Leste dos EUA (Norte da Virgínia)
USE2	us-east-2	Leste dos EUA (Ohio)
UGE1	us-gov-east-1	Gov (Leste dos EUA)
UGW1	us-gov-west-1	Gov (Oeste dos EUA)
USW1	us-west-1	Oeste dos EUA (Norte da Califórnia)
USW2	us-west-2	Oeste dos EUA (Oregon)

Nas seções a seguir, usamos o padrão `REG-UsageType` ao analisar as cobranças do DynamoDB, em que REG especifica a região em que ocorreu o uso e usageType é o código para o tipo de cobrança. Por exemplo, se você vir um item de linha para `USW1- ReadCapacityUnit-Hrs` no arquivo CSV, isso significa que o uso ocorreu em US-West-1 para a capacidade de leitura provisionada. Nesse caso, a listagem será `REG-ReadCapacityUnit-Hrs`.

Tópicos

- Capacidade de throughput
- Fluxos
- Armazenamento
- Backup e restauração
- Transferência de dados
- CloudWatch Contributor Insights
- DynamoDB Accelerator (DAX)

Capacidade de throughput

Capacidade provisionada de leituras e gravações

Ao criar uma tabela do DynamoDB no modo de capacidade provisionada, você deve especificar a capacidade de leitura e gravação que espera que a aplicação exija. O tipo de uso depende da classe da tabela (Standard ou Standard-Infrequent Access). Você deve provisionar leituras e gravações com base na taxa de consumo por segundo, mas as cobranças são feitas por hora com base na capacidade provisionada.

UsageType	Unidades	Granularity	Descrição
REG-ReadCapacityUnit-Hrs	RCU-hours	Hora	Cobranças por leituras no modo de capacidade provisionada usando a classe de tabela Standard.
REG-IA-ReadCapacityUnit-Hrs	RCU-hours	Hora	Cobranças por leituras no modo de capacidade provisionada usando a classe de tabela Standard-IA.
REG-WriteCapacityUnit-Hrs	WCU-hours	Hora	Cobranças por gravações no modo

UsageType	Unidades	Granularity	Descrição
			de capacidade provisionada usando a classe de tabela Standard.
REG-IA-WriteCapacityUnit-Hrs	WCU-hours	Hora	Cobranças por gravações no modo de capacidade provisionada usando a classe de tabela Standard-IA.

Capacidade reservada de leituras e gravações

Com a capacidade reservada, você paga uma taxa única antecipada e se compromete a um nível mínimo de uso provisionado ao longo de um período. A capacidade reservada é cobrada com desconto por hora. Qualquer capacidade que você provisionar além da sua capacidade reservada será cobrada de acordo com as taxas de capacidade provisionada padrão. A capacidade reservada está disponível para unidades de capacidade de leitura e gravação provisionadas de região única (RCU e WCU) nas tabelas do DynamoDB que usam a classe de tabela padrão. A capacidade reservada de um e três anos é cobrada usando as mesmas SKUs.

UsageType	Unidades	Granularity	Descrição
REG-Heavy Usage:dynamodb.read	RCU-hours	Antecipado e depois mensalmente	Cobranças por leituras de capacidade reservada: uma cobrança antecipada única e uma cobrança mensal no início de cada mês cobrindo todas as horas de RCU confirmadas com desconto durante o mês. Terá itens de

UsageType	Unidades	Granularity	Descrição
			linha REG-ReadCapacityUnit-Hrs correspondentes e de custo zero.
REG-HeavyUsage:dynamodb.write	WCU-hours	Antecipado e depois mensalmente	Cobranças por gravações de capacidade reservada: uma cobrança antecipada única e uma cobrança mensal no início de cada mês cobrindo todas as horas de WCU confirmadas com desconto durante o mês. Terá itens de linha REG-WriteCapacityUnit-Hrs correspondentes e de custo zero.

Capacidade de leituras e gravações sob demanda

Ao criar uma tabela do DynamoDB no modo de capacidade sob demanda, você paga somente pelas leituras e gravações que sua aplicação executa. Os preços das solicitações de leitura e gravação dependem da classe da tabela.

UsageType	Unidades	Granularity	Descrição
REG-ReadRequestUnits	RRUs	Unidade	Cobranças por leituras no modo de capacidade sob demanda com a

UsageType	Unidades	Granularity	Descrição
			classe de tabela Standard.
REG-IA-ReadRequestUnits	RRUs	Unidade	Cobranças por leituras no modo de capacidade sob demanda com a classe de tabela Standard-IA.
REG-WriteRequestUnits	WRUs	Unidade	Cobranças por gravações no modo de capacidade sob demanda com a classe de tabela Standard.
REG-IA-WriteRequestUnits	WRUs	Unidade	Cobranças por gravações no modo de capacidade sob demanda com a classe de tabela Standard-IA.

Leituras e gravações de tabelas globais

O DynamoDB cobra pelo uso de tabelas globais com base nos recursos usados em cada tabela de réplica. Para tabelas globais provisionadas, as solicitações de gravação para tabelas globais são medidas em WCUs replicadas (rWCU) em vez de WCUs padrão e as gravações em índices secundários globais em tabelas globais são medidas em WCUs. Para tabelas globais sob demanda, as solicitações de gravação são medidas em WRUs replicadas (rWRU) em vez de WRUs padrão. O número de rWCUs ou rWRUs consumidas para replicação depende da versão das tabelas globais que você está usando. O preço depende da classe da tabela.

As gravações em índices secundários globais (GSIs) são cobradas usando unidades de gravação padrão (WCUs e WRUs). As solicitações de leitura e o armazenamento de dados são cobrados de forma idêntica às tabelas de uma única região.

Se você adicionar uma réplica de tabela para criar ou estender uma tabela global em novas regiões, o DynamoDB cobrará pela restauração da tabela nas regiões adicionadas por gigabyte de dados restaurados. Os dados restaurados são cobrados como REG-RestoreDataSize-Bytes. Consulte Backup e restauração para o DynamoDB para ter detalhes. A replicação entre regiões e a adição de réplicas às tabelas que contêm dados também geram cobranças pela transferência externa de dados.

Ao selecionar o modo de capacidade sob demanda para as tabelas globais do DynamoDB, você paga somente pelos recursos que a aplicação usa em cada tabela de réplica.

UsageType	Unidades	Granularity	Descrição
REG-ReplWriteCapacityUnit-Hrs	rWCU-hours	Hora	Tabela global, provisionada, classe de tabela Standard.
REG-IA-ReplWriteCapacityUnit-Hrs	rWCU-hours	Hora	Tabela global, provisionada, classe de tabela Standard-IA.
REG-ReplWriteRequestUnits	rWRU	Unidade	Tabela global, sob demanda, classe de tabela Standard.
REG-IA-ReplWriteRequestUnits	rWRU	Unidade	Tabela global, sob demanda, classe de tabela Standard-IA.

Fluxos

O DynamoDB tem duas tecnologias de streaming, DynamoDB Streams e Kinesis. Cada um tem preços distintos.

O DynamoDB Streams cobra pela leitura de dados em unidades de solicitação de leitura. Cada chamada de API `GetRecords` é cobrada como uma solicitação de leitura de fluxos. Você não recebe cobranças por chamadas de API `GetRecords` invocadas pelo AWS Lambda como parte de gatilhos do DynamoDB ou por tabelas globais do DynamoDB como parte da replicação.

UsageType	Unidades	Granularity	Descrição
REG-Streams-RequestsCount	Contagem	Unidade	Unidades de solicitação de leitura para o DynamoDB Streams.

O Amazon Kinesis Data Streams cobra em unidades de captura de dados de alterações. O DynamoDB cobra uma unidade de captura de dados de alterações para cada gravação (até 1 KB). Para itens maiores que 1 KB, unidades de captura de dados de alterações adicionais são necessárias. Você paga somente pelas gravações que a aplicação executa sem precisar gerenciar a capacidade de throughput na tabela.

UsageType	Unidades	Granularity	Descrição
REG-ChangeDataCaptureUnits-Kinesis	Unidades de CDC	Unidade	Unidades de captura de dados de alterações para o Kinesis Data Streams.

Armazenamento

O DynamoDB mede o tamanho dos dados faturáveis adicionando o tamanho do byte bruto dos dados mais uma sobrecarga de armazenamento por item que depende dos atributos ativados.

> **Note**
>
> Os valores de uso de armazenamento no CUR serão maiores em comparação com os valores de armazenamento durante o uso de `DescribeTable`, porque `DescribeTable` não inclui as despesas indiretas de armazenamento por item.

O armazenamento é calculado por hora, mas o preço é mensal, calculado por uma média das cobranças por hora.

Embora o armazenamento `UsageType` use `ByteHrs` como sufixo, o uso do armazenamento no CUR é medido em GB e precificado por GB/mês.

UsageType	Unidades	Granularity	Descrição
REG-TimedStorage-ByteHrs	GB	Mês	Quantidade de armazenamento usada por suas tabelas e índices do DynamoDB, para tabelas com a classe de tabela Standard.
REG-IA-TimedStorage-ByteHrs	GB	Mês	Quantidade de armazenamento usada por suas tabelas e índices do DynamoDB, para tabelas com a classe de tabela Standard-IA.

Backup e restauração

O DynamoDB oferece dois tipos de backups: backups para recuperação para um ponto no tempo (PITR) e backups sob demanda. Os usuários também podem restaurar esses backups em tabelas do DynamoDB. As cobranças abaixo se referem tanto a backups quanto a restaurações.

As cobranças de armazenamento de backup são feitas no primeiro dia do mês, com ajustes ao longo do mês à medida que os backups são adicionados ou removidos. Consulte o blog Understanding Amazon DynamoDB On-demand Backups and Billing para ter mais informações.

UsageType	Unidades	Granularity	Descrição
REG-TimedBackupStorage-ByteHrs	GB	Mês	O armazenamento consumido por backups sob demanda de suas tabelas e índices secundários locais do DynamoDB.
TimedPITRStorage-ByteHrs	GB	Mês	O armazenamento usado por backups para recuperação para um ponto no tempo (PITR). O DynamoDB monitora continuamente o tamanho das tabelas habilitadas para PITR durante todo o mês a fim de determinar as cobranças de backup e as contas para armazenamento durante o período em que a PITR permanece habilitada.
REG-RestoreDataSize-Bytes	GB	Tamanho	O tamanho total dos dados restaurados (incluindo dados de tabela, índices secundários locais e índices secundários globais) medido em GB por meio

UsageType	Unidades	Granularity	Descrição
			de backups do DynamoDB.

AWS Backup

O AWS Backup é um serviço de backup totalmente gerenciado que facilita a centralização e a automação do backup de dados entre todos os serviços da AWS na nuvem e on-premises. O AWS Backup é cobrado pelo armazenamento (armazenamento de alta e baixa atividade), atividades de restauração e transferência de dados entre regiões. As cobranças `UsageType` a seguir aparecem no ProductCode “AWSBackup” e não no “AmazonDynamoDB”.

UsageType	Unidades	Granularity	Descrição
REG-WarmStorage-ByteHrs-DynamoDB	GB	Mês	O armazenamento usado pelos backups do DynamoDB é gerenciado pelo AWS Backup durante todo o mês, medido em GB/mês.
REG-CrossRegion-WarmBytes-DynamoDB	GB	Tamanho	Os dados transferidos para uma região da AWS diferente dentro da mesma conta ou para outra conta da AWS. As cobranças por transferências entre regiões ocorrem ao copiar backups de uma região em outra. A cobrança é sempre realizada na conta

UsageType	Unidades	Granularity	Descrição
			da qual os dados são transferidos.
REG-Restore-WarmBytes-DynamoDB	GB	Tamanho	O tamanho total dos dados restaurados do armazenamento de alta atividade, medido em GB.
REG-ColdStorage-ByteHrs-DynamoDB	GB	Mês	O armazenamento de baixa atividade usado pelos backups do DynamoDB é gerenciado pelo AWS Backup durante todo o mês, medido em GB/mês.
REG-Restore-ColdBytes-DynamoDB	GB	Mês	O tamanho total dos dados restaurados do armazenamento de baixa atividade, medido em GB.

Exportação e importação

É possível exportar dados do DynamoDB para o Amazon S3 ou importar dados do Amazon S3 para uma nova tabela do DynamoDB.

Embora `UsageType` use `Bytes` como sufixo, o uso de exportação e importação no CUR é medido e precificado em GB.

UsageType	Unidades	Granularity	Descrição
REG-ExportDataSize-Bytes	GB	Tamanho	A cobrança pela exportação dos

UsageType	Unidades	Granularity	Descrição
			dados para o S3. O DynamoDB cobra por dados exportados com base no tamanho da tabela do DynamoDB (dados da tabela e índices secundários locais) no momento em que a exportação foi criada.
REG-ImportDataSize-Bytes	GB	Tamanho	A cobrança pela importação dos dados do S3. O tamanho é calculado com base no tamanho do objeto não compactado dos dados no Amazon S3. Não há custos adicionais pela importação para tabelas com GSIs.
REG-IncrementalExportDataSize-Bytes	GB	Tamanho	A cobrança pelo tamanho dos dados processados a por meio do backup contínuo para produzir exportações incrementais.

Transferência de dados

A atividade de transferência de dados pode aparecer associada ao serviço do DynamoDB. O DynamoDB não cobra pela transferência de dados de entrada e não cobra pelos dados transferidos

entre o DynamoDB e outros serviços da AWS na mesma região da AWS (em outras palavras, USD 0,00 por GB). Os dados transferidos entre regiões da AWS [como entre o DynamoDB na região Leste dos EUA (Norte da Virgínia) e o Amazon EC2 na região da UE (Irlanda)] são cobrados nos dois lados da transferência.

UsageType	Unidades	Granularity	Descrição
REG-DataTransfer-In-Bytes	GB	Unidades	Dados transferidos da internet para o DynamoDB.
REG-DataTransfer-Out-Bytes	GB	Unidades	Dados transferidos do DynamoDB para a internet.

CloudWatch Contributor Insights

O CloudWatch Contributor Insights para DynamoDB é uma ferramenta de diagnóstico para identificar as chaves acessadas com maior frequência e com controle de utilização na tabela do DynamoDB. As cobranças `UsageType` a seguir aparecem no ProductCode “AmazonCloudWatch” e não no “AmazonDynamoDB”.

UsageType	Unidades	Granularity	Descrição
REG-CW:ContributorEventsManaged	Eventos processados	Unidades	A quantidade de eventos do DynamoDB processados. Por exemplo, para uma tabela com o CloudWatch Contributor Insights habilitado, sempre que um item é lido ou gravado, ele é contabilizado como um evento. Se

UsageType	Unidades	Granularity	Descrição
			a tabela tiver uma chave de classific ação, isso vai gerar cobranças por dois eventos.
REG-CW:Contributor RulesManaged	Contagem de regras	Mês	O DynamoDB cria regras para identificar os itens mais acessados e as chaves com maior controle de utilização ao habilitar o Cloud Watch Contributor Insights. Essa cobrança é realizada pelas regras adicionadas para cada entidade (tabelas e GSIs) configurada para registrar em log o CloudWatch Contribut or Insights.

DynamoDB Accelerator (DAX)

O DynamoDB Accelerator (DAX) é cobrado por hora com base no tipo de instância selecionado para o serviço. As cobranças abaixo se referem às instâncias provisionadas do DynamoDB Accelerator. As cobranças `UsageType` a seguir aparecem no ProductCode “AmazonDAX” e não no “AmazonDynamoDB”.

UsageType	Unidades	Granularity	Descrição
REG-NodeUsage:dax-<INSTANCETYPE>	Node-hour	Hora	O uso horário de um tipo específico de instância. O preço é por hora de nó consumida, desde o momento em que um nó é lançado até seu encerramento. Cada hora de nó parcial consumida será cobrada como uma hora completa. O DAX cobra por nó em um cluster do DAX. Se você tiver um cluster com vários nós, verá vários itens de linha no relatório de faturamento.

O tipo de instância será um dos valores da lista a seguir. Para obter detalhes sobre os tipos de nó, consulte Nós.

- r3.2xlarge, r4.8xlarge ou r5.8xlarge
- r3.4xlarge, r4.large ou r5.large
- r3.8xlarge, r4.xlarge ou r5.xlarge
- r3.2xlarge, r5.12xlarge ou t2.medium
- r3.4xlarge, r4.large ou r5.large
- r3.xlarge, r5.16xlarge ou t2.small
- r4.16xlarge, r5.24xlarge ou t3.medium
- r4.2xlarge, r5.2xlarge ou t3.small
- r4.4xlarge ou r5.4xlarge

Migrar uma tabela do DynamoDB de uma conta para outra

É possível migrar uma tabela do Amazon DynamoDB de uma conta para outra a fim de implementar uma estratégia multiconta ou uma estratégia de backup. Você também pode fazer isso para fins de teste, depuração ou conformidade. Um caso de uso comum é copiar tabelas do DynamoDB em ambientes de produção, preparação, teste e desenvolvimento, onde cada ambiente utiliza uma conta da AWS diferente.

O DynamoDB oferece duas opções para migrar tabelas de uma conta da AWS para outra:

- AWS Backup para backup e restauração entre contas: o AWS Backup é um serviço de backup totalmente gerenciado que permite gerenciar os backups em diversos serviços da AWS de maneira central. Com sua funcionalidade de backup e restauração entre contas, você pode fazer backup de uma tabela do DynamoDB em uma conta e restaurar o backup em outra conta na mesma organização da AWS.
- Exportação e importação do DynamoDB para o Amazon S3: o uso dos recursos de exportação e importação do DynamoDB para o Amazon S3 permite que você faça uma exportação completa para um bucket do Amazon S3, depois importe esses dados para uma nova tabela em outra conta da AWS. Essa abordagem é adequada quando você precisa fazer uma migração entre contas que não fazem parte da mesma organização da AWS ou quando não quiser usar o AWS Backup.

> **Note**
>
> A importação do Amazon S3 não oferece suporte a tabelas com índices secundários locais (LSIs), mas oferece suporte a índices secundários globais (GSIs). Para obter mais informações sobre LSIs e GSIs, consulte Melhorar o acesso aos dados com índices secundários no DynamoDB.

Tópicos

- Migrar uma tabela usando o AWS Backup para backup e restauração entre contas
- Migrar uma tabela usando a exportação para o S3 e a importação do S3

Migrar uma tabela usando o AWS Backup para backup e restauração entre contas

Pré-requisitos

- As contas da AWS de origem e de destino devem pertencer à mesma organização no serviço AWS Organizations.
- Permissões do AWS Identity and Access Management (IAM) para criar e usar cofres do AWS Backup.

Para obter mais informações sobre como configurar backups entre contas, consulte Creating backup copies across AWS accounts.

Informações sobre preços

A AWS cobra pelo backup (com base no tamanho da tabela), por qualquer cópia de dados entre regiões da AWS (com base na quantidade de dados), pela restauração (com base na quantidade de dados) e por quaisquer cobranças contínuas de armazenamento. Para evitar cobranças contínuas, você pode excluir o backup se não precisar dele depois da restauração.

Para obter mais informações sobre preços, consulte Preços do AWS Backup.

Etapa 1: habilitar recursos avançados para o DynamoDB e o backup entre contas

1. Nas contas da AWS de origem e de destino, acesse o Console de Gerenciamento da AWS e abra o console do AWS Backup.
2. Selecione a opção Configurações.
3. Em Recursos avançados para backups do Amazon DynamoDB, confirme se a opção Recursos avançados está habilitada. Se não estiver, escolha Habilitar.
4. Em Gerenciamento entre contas, escolha Ativar para a opção Backup entre contas.

Etapa 2: criar um cofre de backup na conta de origem e na conta de destino

1. Nas contas de origem da AWS, abra o console do AWS Backup.
2. Escolha Cofres de backup.
3. Escolha Criar cofre de backup.

4. Copie e salve o nome do recurso da Amazon (ARN) dos cofres de backup criados e da conta de destino da AWS.
5. Você precisará dos ARNs dos cofres de backup de origem e de destino ao copiar o backup da tabela do DynamoDB entre contas.

Etapa 3: criar um backup de tabela do DynamoDB na conta de origem

1. Na página do painel do AWS Backup, escolha Criar um backup sob demanda.
2. Na seção Configurações, selecione DynamoDB como Tipo de recurso e escolha o nome da tabela.
3. Na lista suspensa Cofre de backup, escolha o cofre de backup que você criou na conta de origem.
4. Selecione o Período de retenção desejado.
5. Escolha Criar backup sob demanda.
6. Monitore o status do trabalho de backup na guia Tarefas de backup da página Trabalhos do AWS Backup.

Etapa 4: copiar o backup da tabela do DynamoDB da conta de origem para a conta de destino

1. Depois que o trabalho de backup for concluído, abra o console do AWS Backup na conta de origem e escolha Cofres de backup.
2. Em Backups, escolha o backup da tabela do DynamoDB. Escolha Ações e selecione Copiar.
3. Insira a região da AWS da conta de destino.
4. Em ARN do cofre externo, insira o ARN do cofre de backup que você criou na conta de destino.
5. No cofre de backup da conta de destino, habilite o acesso de uma conta de origem para permitir a cópia de backups.

Etapa 5: restaurar o backup da tabela do DynamoDB na conta de destino

1. Na conta da AWS de destino, abra o console do AWS Backup e escolha Cofres de backup.
2. Em Backups, selecione o backup que você copiou da conta de origem. Em Ações, escolha Restaurar.

3. Insira o nome da nova tabela do DynamoDB, a criptografia que essa nova tabela terá, a chave com a qual você deseja que a restauração seja criptografada e outras opções necessárias.
4. Quando a restauração for concluída, o status da tabela será exibido como Ativo.

Migrar uma tabela usando a exportação para o S3 e a importação do S3

Pré-requisitos

- É necessário habilitar a recuperação para um ponto no tempo (PITR) para sua tabela a fim de realizar a exportação para o S3. Para obter mais informações, consulte Habilitar a recuperação para um ponto no tempo no DynamoDB.
- Permissões válidas do IAM para realizar a exportação. Para obter mais informações, consulte Solicitação de uma exportação de tabela no DynamoDB.
- Permissões válidas do IAM suficientes para realizar a importação. Para obter mais informações, consulte Solicitação de importação de tabela no DynamoDB.

Informações sobre preços

A AWS cobra pela PITR (com base no tamanho da tabela e no tempo em que a PITR permanece habilitada). Se você só precisar da PITR para a exportação, poderá desativá-la após a conclusão da exportação. A AWS também cobra pelas solicitações feitas ao S3, pelo armazenamento dos dados exportados no S3 e pela importação (com base no tamanho não compactado dos dados importados).

Para obter informações sobre os preços do DynamoDB, consulte Preço do Amazon DynamoDB.

> **Note**
>
> Há limites de tamanho e quantidade de objetos ao importar do S3 para o DynamoDB. Para obter mais informações, consulte Importar cotas.

Solicitar uma exportação de tabela para o Amazon S3

1. Faça login no Console de Gerenciamento da AWS e abra o console do DynamoDB.
2. No painel de navegação, no lado esquerdo do console, escolha Export to S3 (Exportar para o S3).

3. Escolha uma tabela de origem e um bucket do S3 como destino. Insira o URL do bucket da conta de destino usando o formato `s3://bucketname/prefix`. O `/prefix` é uma pasta opcional para ajudar a manter o bucket de destino organizado.
4. Escolha Exportação completa. A exportação completa gera o snapshot completo da tabela, tal como ela estava no momento especificado.
   a. Selecione Hora atual para exportar o snapshot completo mais recente da tabela.
   b. Em Formato de arquivo exportado, escolha entre JSON do DynamoDB e Amazon Ion. A opção padrão é JSON do DynamoDB.
5. Clique no botão Export (Exportar) para iniciar a exportação.
6. As exportações de tabelas pequenas costumam ser concluídas em questão de minutos, mas tabelas em escala de terabytes podem levar mais de uma hora.

Solicitar uma importação de tabela do Amazon S3

1. Faça login no Console de Gerenciamento da AWS e abra o console do DynamoDB.
2. No painel de navegação, no lado esquerdo do console, escolha Import from S3 (Importar do S3).
3. Na página exibida, selecione Import from S3 (Importar do S#).
4. Insira o URL de origem do Amazon S3. Você também pode encontrá-lo usando o botão Procurar no S3. O formato do caminho esperado é `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`.
5. Especifique se você é o S3 bucket owner (Proprietário do bucket do S3).
6. Em Compactação de arquivos de importação, selecione GZIP para corresponder à exportação.
7. Em Formato de arquivo de importação, escolha JSON do DynamoDB para corresponder à exportação.
8. Escolha Próximo. Em Especificar detalhes da tabela, escolha as opções referentes à nova tabela que será criada para armazenar seus dados.
9. Escolha Próximo. Em Definir as configurações da tabela, personalize quaisquer configurações adicionais da tabela, se aplicável.
10. Selecione novamente Next (Próximo) para revisar suas opções de importação e, em seguida, clique em Import (Importar) para iniciar a tarefa de importação. Você verá sua nova tabela listada em Importações do S3 com o status Importando. Não é possível acessar a tabela durante esse período. As importações pequenas costumam ser concluídas em questão de minutos, mas tabelas em escala de terabytes podem levar mais de uma hora.

11. Após a conclusão da importação, quando o status é exibido como Ativo, você pode começar a usar a tabela.

Manter as tabelas sincronizadas durante a migração

Se você puder pausar as operações de gravação na tabela de origem durante a migração, a origem e a saída deverão ser idênticas após a migração. Se você não conseguir pausar as operações de gravação, a tabela de destino provavelmente ficará um pouco atrasada em relação à origem depois da migração. Para alcançar a tabela de origem, você pode usar o streaming (DynamoDB Streams ou Kinesis Data Streams para DynamoDB) para reproduzir as gravações que ocorreram na tabela de origem desde o backup ou a exportação.

Comece a ler os registros de fluxo antes do carimbo de data e hora da exportação da tabela de origem para o S3. Por exemplo, se a exportação para o S3 ocorreu às 14h e a importação para a tabela de destino foi concluída às 23h, você deverá iniciar a leitura de fluxos do DynamoDB a partir das 13h58. A tabela de opções de streaming para captura de dados de alteração resume os recursos de cada modelo de streaming.

O uso do DynamoDB Streams com o Lambda oferece uma abordagem simplificada para a sincronização de dados entre as tabelas de origem e de destino do DynamoDB. Você pode usar uma função do Lambda para reproduzir cada gravação na tabela de destino.

> **Note**
>
> Os itens são mantidos no DynamoDB Streams por 24 horas, portanto planeje a conclusão do backup e da restauração ou da exportação e importação dentro dessa janela.

Recomendações para integrar o DAX às aplicações do DynamoDB

O DynamoDB Accelerator (DAX) é um serviço de armazenamento em cache compatível com o DynamoDB que oferece rápido desempenho em memória para aplicações exigentes, como aplicações com alto volume de leituras. Com o DAX, você pode obter tempos de resposta na casa dos microssegundos para acessar os dados solicitados com frequência. Estas recomendações do DynamoDB Accelerator fornecem insights abrangentes e práticas recomendadas para integrar o DAX às suas aplicações do DynamoDB.

Este guia fornece conhecimento básico para quem está começando a usar o DAX ou quer otimizar as configurações existentes. Este guia aborda vários tópicos, incluindo quando usar o DAX e como criar um cluster do DAX. Também inclui exemplos práticos e explicações detalhadas para ajudar você a implementar o DAX de forma eficaz em seus projetos. Por fim, este guia oferece estratégias avançadas que você precisa implementar para maximizar os recursos de armazenamento em cache do DAX e garantir aplicações rápidas e escaláveis.

Tópicos

- Avaliar a adequação do DAX aos seus casos de uso
- Configurar clientes do DAX
- Configurar um cluster do DAX
- Dimensionar um cluster do DAX
- Implantar um cluster
- Gerenciar as operações do cluster
- Monitoramento do DAX

Avaliar a adequação do DAX aos seus casos de uso

Esta seção explica quando e por que usar o DAX. Use essas orientações para determinar se a integração do DAX com o DynamoDB é vantajosa para os padrões de workload, os requisitos de desempenho e as necessidades de consistência de dados da sua aplicação. Também abrange cenários em que o DAX pode não ser adequado, como workloads com alto volume de gravações e dados acessados com pouca frequência.

Nesta seção

- Quando e por que escolher o DAX
- Quando não usar o DAX

Quando e por que escolher o DAX

Você pode considerar a adição do DAX à sua pilha de aplicações em vários cenários. Por exemplo, use o DAX para reduzir a latência geral das solicitações de leitura no DynamoDB ou para minimizar as leituras repetidas dos mesmos dados de uma tabela. A seguinte lista apresenta exemplos de cenários nos quais você pode aproveitar a integração do DAX com o DynamoDB:

- Requisito de alto desempenho
  - Leituras de baixa latência: considere o uso do DAX se sua aplicação exigir tempos de resposta na casa dos microssegundos para leituras finais consistentes. O DAX também pode reduzir drasticamente o tempo de resposta de acesso a dados lidos com frequência.
- Workloads com alto volume de leituras
  - Aplicações com alto volume de leituras: para aplicações com alta relação entre leituras e gravações, por exemplo, 10:1 ou mais, o DAX resulta em mais acessos ao cache e menos dados obsoletos. Isso reduz a quantidade de leituras em uma tabela. Para evitar a leitura de dados obsoletos do cache se sua aplicação tiver alto volume de gravações, defina uma Usar a vida útil (TTL) no DynamoDB mais baixa para o cache.
  - Armazenamento de consultas comuns em cache: se sua aplicação lê frequentemente os mesmos dados, por exemplo, produtos populares em uma plataforma de comércio eletrônico, o DAX pode atender a essas solicitações diretamente do cache.
- Padrões de tráfego intermitente
  - Ajuste mais suave da escala da tabela: o DAX ajuda a suavizar os impactos de picos repentinos de tráfego. O DAX fornece um buffer para aumentar a escala verticalmente da capacidade da tabela do DynamoDB de forma adequada, o que reduz o risco de controle de utilização de leituras.
  - Maior throughput de leitura para cada item: o DynamoDB aloca partições individuais para cada item. No entanto, uma partição começa a aplicar o controle de utilização nas leituras de um item quando atinge 3.000 unidades de capacidade de leitura (RCUs). O DAX permite escalar as leituras de um único item para além de 3.000 RCUs.
- Otimização de custo
  - Redução dos custos do DynamoDB: a realização de leituras do DAX pode reduzir a quantidade de leituras enviadas para uma tabela do DynamoDB, o que pode afetar diretamente o custo. Com uma alta taxa de acertos de cache, o custo reduzido de leitura da tabela pode exceder o custo do cluster do DAX, resultando em uma redução do custo líquido.
- Requisitos de consistência de dados
  - Consistência eventual: o DAX oferece suporte a leituras finais consistentes. Isso torna o DAX adequado para casos de uso em que a consistência imediata não é essencial.
  - Armazenamento em cache com write-through: as gravações feitas no DAX são do tipo write-through. Depois que o DAX confirma a gravação de um item no DynamoDB, ele mantém essa versão do item no cache de itens. Esse mecanismo de write-through ajuda a manter uma

consistência de dados mais rigorosa entre o cache e o banco de dados, mas usa recursos adicionais do cluster do DAX.

Quando não usar o DAX

Embora o DAX seja poderoso, ele não é adequado para todos os cenários. A seguinte lista apresenta exemplos de cenários em que a integração do DAX com o DynamoDB não é adequada:

- Workloads com alto volume de gravações: a principal vantagem do DAX é acelerar as leituras, mas as gravações usam mais recursos do DAX do que as leituras. Se sua aplicação apresentar alto volume de gravações, os benefícios do DAX poderão ser limitados.
- Dados lidos com baixa frequência: se sua aplicação acessa dados com baixa frequência ou uma grande variedade de dados raramente reutilizados (dados frios), é provável que você tenha uma baixa cache hit ratio. Nesse caso, a despesa de manutenção do cache pode não justificar os ganhos de desempenho.
- Leituras ou gravações em massa: se sua aplicação realiza mais gravações em massa do que gravações individuais, é melhor gravar sem usar o DAX. Além disso, para leituras em massa, é necessário executar varreduras completas da tabela diretamente em uma tabela do DynamoDB.
- Forte consistência ou requisitos de transação: o DAX transmite leituras altamente consistentes e chamadas TransactGetItems para uma tabela do DynamoDB. Você deve realizar essas leituras sem usar o cluster do DAX para evitar o uso de recursos do cluster. Os itens lidos dessa forma não serão armazenados em cache, portanto rotear esses itens pelo DAX não oferece nenhuma vantagem.
- Aplicações simples com requisitos de desempenho moderado: para aplicações com requisitos de desempenho moderado e tolerância à latência direta do DynamoDB, a complexidade e o custo de adicionar o DAX podem não ser necessários. Por si só, o DynamoDB lida com alto throughput e fornece desempenho de latência inferior a 10 milissegundos.
- Necessidades de consultas complexas além do acesso a valores e chaves: o DAX é otimizado para padrões de acesso a valores e chaves. Se sua aplicação exigir recursos complexos de consulta com filtragem complexa, como operações Query e Scan, os benefícios do armazenamento em cache do DAX poderão ser limitados.

  Nessas situações, use o Amazon ElastiCache (Redis OSS) como alternativa. O ElastiCache (Redis OSS) comporta estruturas de dados avançadas, como listas, conjuntos e hashes. Ele também oferece outros recursos, como pub/sub, índices geoespaciais e scripts.

- Requisitos de conformidade: o DAX ainda não oferece os mesmos credenciamentos de conformidade do DynamoDB. Por exemplo, o DAX ainda não obteve o credenciamento SOC.

Configurar clientes do DAX

O cluster do DAX é um cluster baseado em instância que pode ser acessado usando vários SDKs do DAX. Cada SDK fornece aos desenvolvedores opções configuráveis, como requestTimeout e conexões, para atender aos requisitos específicos da aplicação.

Quando um cliente do DAX é configurado, uma consideração essencial é a escala da aplicação cliente, especificamente a proporção entre instâncias de cliente e instâncias de servidor do DAX (que tem no máximo 11). Grandes frotas de instâncias de cliente podem gerar numerosas conexões com instâncias de servidor do DAX e sobrecarregá-las. Este guia descreve as práticas recomendadas para a configuração de clientes do DAX.

Práticas recomendadas

1. Instâncias de cliente: implemente instâncias de cliente singleton para garantir a reutilização de instâncias em todas as solicitações. Para obter mais detalhes sobre a implementação, visite the section called “Etapa 4: executar uma aplicação de amostra”.
2. Tempos limite de solicitação: embora as aplicações geralmente exijam tempos limite de solicitação baixos com o objetivo de garantir latência mínima para sistemas precedentes, definir tempos limite muito baixos pode causar problemas. Tempos limite baixos podem desencadear reconexões frequentes com as instâncias de servidor quando os servidores do DAX experimentam picos temporários de latência. Quando ocorre um tempo limite, o cliente do DAX encerra a conexão existente do nó do servidor e estabelece uma nova. Como o estabelecimento da conexão consome muitos recursos, várias conexões consecutivas podem sobrecarregar os servidores do DAX. Recomendamos o seguinte:
   - Manter as configurações padrão de tempo limite da solicitação.
   - Se forem necessários tempos limite menores, implemente threads de aplicação separados com valores de tempo limite mais baixos e inclua mecanismos de repetição com recuo exponencial.
3. Tempo limite de conexão: com relação à maioria das aplicações, recomendamos manter as configurações de tempo limite de conexão padrão.
4. Conexões simultâneas: alguns SDKs, como o JavaV2, permitem o ajuste de conexões simultâneas com o servidor do DAX. Considerações importantes:
   - As instâncias de servidor do DAX comportam até 40 mil conexões simultâneas.

- As configurações padrão são adequadas para a maioria dos casos de uso.
- Grandes instâncias de cliente combinadas com altas conexões simultâneas podem sobrecarregar os servidores.
- Valores mais baixos de conexões simultâneas reduzem o risco de sobrecarga do servidor.
- Exemplo de cálculo de performance:
  - Supondo-se uma latência de solicitação de 1 ms, cada conexão pode, teoricamente, lidar com mil solicitações/segundo.
  - Para um cluster de três nós, uma única instância de cliente conectada a todos os nós pode processar 3 mil solicitações/segundo.
  - Com 10 conexões, o cliente pode lidar com aproximadamente 30 mil solicitações/segundo.

    Recomendação: comece com configurações de conexão simultânea mais baixas e valide por meio de testes de performance com padrões de workload de produção esperados.

Configurar um cluster do DAX

O cluster do DAX é um cluster gerenciado, mas você pode ajustar as configurações para atender aos requisitos da sua aplicação. Devido à estreita integração com as operações de API do DynamoDB, você deve considerar os seguintes aspectos ao integrar sua aplicação ao DAX.

Nesta seção

- Preços do DAX
- Cache de itens e cache de consultas
- Selecionar a configuração de TTL para os caches
- Armazenar várias tabelas em cache com um cluster do DAX
- Replicação de dados em tabelas globais do DAX e do DynamoDB
- Disponibilidade de regiões do DAX
- Comportamento de armazenamento em cache do DAX

Preços do DAX

O custo de um cluster depende da quantidade e do tamanho dos nós provisionados. Cada nó é cobrado por hora de execução no cluster. Para obter mais informações, consulte a Definição de preço do Amazon DynamoDB.

Os acertos de cache não geram custos do DynamoDB, mas afetam os recursos do cluster do DAX. As perdas no cache geram custos de leitura do DynamoDB e exigem recursos do DAX. As gravações geram custos de gravação do DynamoDB e afetam os recursos do cluster do DAX para proxy da gravação.

Cache de itens e cache de consultas

O DAX mantém um cache de itens e um cache de consultas. Compreender as diferenças entre esses caches pode ajudar você a determinar as características de desempenho e consistência que eles oferecem à sua aplicação.

Características do cache	Cache de itens	Cache de consultas
Finalidade	Armazena os resultados das operações de API GetItem e BatchGetItem.	Armazena os resultados das operações de API Query e Scan. Essas operações podem retornar vários itens com base nas condições da consulta, em vez de chaves de item específicas.
Tipo de acesso	Utiliza acesso baseado em chave. Quando uma aplicação solicita dados usando `GetItem` ou `BatchGetItem` , primeiro o DAX verifica o cache de itens usando a chave primária dos itens solicitados. Se o item estiver armazenado em cache e vigente, o DAX o retornará imediatamente sem acessar a tabela do DynamoDB.	Utiliza acesso baseado em parâmetro. O DAX armazena em cache o conjunto de resultados das operações de API `Query` e `Scan`. O DAX atende às solicitações subsequentes com os mesmos parâmetro s que incluem as mesmas condições de consulta, tabela e índice do cache. Isso reduz significativamente os tempos de resposta e o consumo de throughput de leitura do DynamoDB.

Características do cache	Cache de itens	Cache de consultas
Invalidação de cache	O DAX replica automatic amente os itens atualizados no cache de itens dos nós no cluster do DAX nos seguintes cenários: • Gravação de uma atualizaç ão de um item por meio do cache. • Leitura de uma versão atualizada do item na tabela.	É mais difícil invalidar o cache de consultas do que o cache de itens. As atualizaç ões de itens podem não ser mapeadas diretamente para consultas ou verificações armazenadas em cache. É necessário ajustar cuidadosa mente a TTL do cache de consultas para manter a consistência de dados. As gravações por meio do DAX ou da tabela base não são refletidas no cache de consultas até que a TTL expire a resposta anteriorm ente armazenada em cache e o DAX realize uma nova consulta no DynamoDB.
Índice secundário global	Como a operação da API `GetItem` não comporta índices secundários locais ou globais, o cache de itens só armazena em cache as leituras da tabela base.	O cache de consultas armazena em cache as consultas em tabelas e índices.

Selecionar a configuração de TTL para os caches

A TTL determina o período em que os dados são armazenados no cache antes de ficarem obsoletos. Após esse período, os dados são atualizados automaticamente na próxima solicitação. Escolher a configuração de TTL correta para seus caches do DAX envolve o equilíbrio entre a otimização do desempenho da aplicação e a consistência dos dados. Como não existe uma configuração de TTL universal que funcione para todas as aplicações, a configuração ideal de TTL varia de acordo com as

características e os requisitos específicos da aplicação. Recomendamos que você comece com uma configuração de TTL conservadora, usando estas recomendações. Depois, ajuste iterativamente a configuração de TTL com base nos dados e insights de desempenho da aplicação.

O DAX mantém uma lista de menos usados recentemente (LRU) para o cache de itens. A lista LRU rastreia quando os itens são gravados pela primeira vez ou lidos pela última vez no cache. Quando a memória do nó do DAX está cheia, o DAX remove itens antigos, mesmo que ainda não tenham expirado, para abrir espaço para novos itens. O algoritmo de LRU está sempre habilitado e não pode ser configurado pelo usuário.

Para definir uma duração de TTL que funcione para suas aplicações, considere os seguintes pontos:

Entenda os padrões de acesso aos dados

- Workloads com alto volume de leituras: para aplicação com workloads com alto volume leituras e atualizações de dados pouco frequentes, defina uma duração de TTL mais longa para reduzir o número de perdas no cache. Uma duração de TTL mais longa também reduz a necessidade de acessar a tabela subjacente do DynamoDB.
- Workloads com alto volume de gravações: para aplicações com atualizações frequentes que não são gravadas por meio do DAX, defina uma duração de TTL mais curta para garantir que o cache permaneça consistente com o banco de dados. Uma duração de TTL mais curta também reduz o risco de fornecer dados obsoletos.

Avalie os requisitos de desempenho da aplicação

- Sensibilidade à latência: se a aplicação exigir baixa latência em relação à atualização dos dados, use uma duração de TTL mais longa. Uma duração de TTL mais longa maximiza os acertos de cache, o que reduz a latência média de leitura.
- Throughput e escalabilidade: uma duração de TTL mais longa reduz a carga nas tabelas do DynamoDB e melhora o throughput e a escalabilidade. No entanto, você deve equilibrar isso com a necessidade de dados atualizados.

Analise a remoção de cache e o uso de memória

- Limites de memória de cache: monitore o uso de memória do cluster do DAX. Uma duração de TTL mais longa pode armazenar mais dados no cache, o que pode atingir os limites de memória e levar a remoções baseadas em LRU.

Use métricas e monitoramento para ajustar a TTL

Analise regularmente as métricas, por exemplo, acertos e perdas no cache e utilização de CPU e memória. Ajuste a configuração de TTL com base nessas métricas para encontrar um equilíbrio ideal entre desempenho e atualização dos dados. Se as perdas no cache forem altas e a utilização de memória for baixa, aumente a duração da TTL para aumentar a taxa de acertos de cache.

Considere os requisitos de negócios e a conformidade

As políticas de retenção de dados podem ditar a duração máxima da TTL que você pode definir para informações confidenciais ou pessoais.

Comportamento do cache ao definir a TTL como zero

Se você definir a TTL como 0, o cache de itens e o cache de consultas apresentarão os seguintes comportamentos:

- Cache de itens: os itens no cache são atualizados somente quando ocorre uma remoção por LRU ou uma operação do tipo write-through.
- Cache de consultas: as respostas das consultas não são armazenadas em cache.

Armazenar várias tabelas em cache com um cluster do DAX

Para workloads com várias tabelas pequenas do DynamoDB que não precisam de caches individuais, um único cluster do DAX armazena em cache as solicitações dessas tabelas. Isso proporciona um uso mais flexível e eficiente do DAX, especialmente para aplicações que acessam várias tabelas e exigem leituras de alto desempenho.

Assim como as APIs do plano de dados do DynamoDB, as solicitações do DAX exigem um nome de tabela. Se você usa várias tabelas no mesmo cluster do DAX, não precisa de nenhuma configuração específica. No entanto, você precisa garantir que as permissões de segurança do cluster permitam acesso a todas as tabelas armazenadas em cache.

Considerações sobre o uso do DAX com várias tabelas

Ao usar o DAX com várias tabelas do DynamoDB, você deve avaliar os seguintes pontos:

- Gerenciamento de memória: ao usar o DAX com várias tabelas, você deve considerar o tamanho total do seu conjunto de dados de trabalho. Todas as tabelas em seu conjunto de dados compartilharão o mesmo espaço de memória do tipo de nó selecionado.

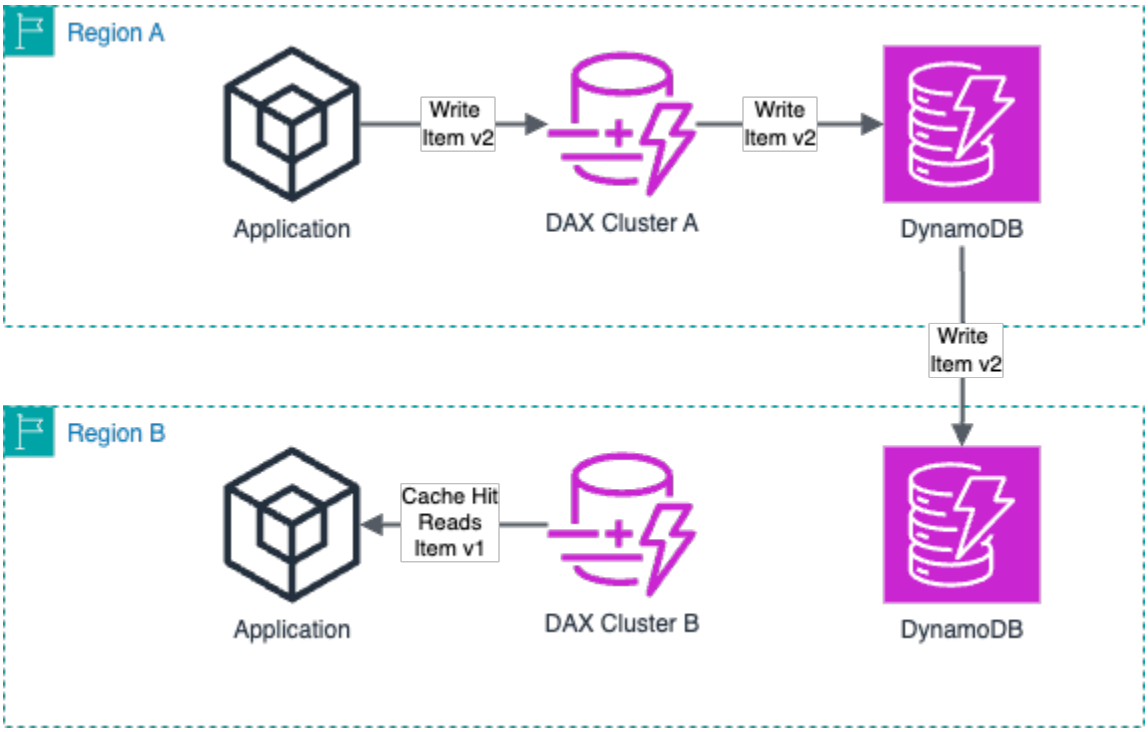
- Alocação de recursos: os recursos do cluster do DAX são compartilhados entre todas as tabelas armazenadas em cache. No entanto, uma tabela de alto tráfego pode causar a remoção de dados das tabelas menores vizinhas.
- Economias de escala: agrupe recursos menores em um cluster do DAX maior para calcular a média do tráfego em um padrão mais estável. Para o número total de recursos de leitura que o cluster do DAX exige, também é econômico ter três ou mais nós. Isso também aumenta a disponibilidade de todas as tabelas armazenadas em cache no cluster.

Replicação de dados em tabelas globais do DAX e do DynamoDB

O DAX é um serviço baseado em região, portanto um cluster só conhece o tráfego dentro de sua Região da AWS. As tabelas globais gravam sem usar o cache ao replicar dados de outra região.

Uma duração de TTL mais longa pode fazer com que dados obsoletos permaneçam na sua região secundária por mais tempo do que na região primária. Isso pode resultar em perdas no cache local da região secundária.

O diagrama a seguir mostra a replicação de dados ocorrendo no nível da tabela global na Região A de origem. O cluster do DAX na Região B não reconhece imediatamente os dados recém-replicados da Região A de origem.



Disponibilidade de regiões do DAX

Nem todas as regiões que oferecem suporte às tabelas do DynamoDB oferecem suporte à implantação de clusters do DAX. Se sua aplicação exigir baixa latência de leitura por meio do DAX, primeiro consulte a lista de regiões compatíveis com o DAX. Depois, selecione uma região para sua tabela do DynamoDB.

Comportamento de armazenamento em cache do DAX

O DAX executa armazenamento em cache de metadados e negativo. A compreensão desses comportamentos de armazenamento em cache ajudará você a gerenciar com eficiência os metadados de atributos dos itens armazenados em cache e das entradas de cache negativo.

- Armazenamento de metadados em cache: os clusters do DAX mantêm indefinidamente metadados sobre os nomes de atributos dos itens armazenados em cache. Esses metadados persistem mesmo depois que o item expira ou é removido do cache.

  Com o tempo, as aplicações que usam um número não vinculado de nomes de atributos podem provocar exaustão de memória no cluster do DAX. Essa limitação aplica-se somente aos nomes de atributo de nível superior, não a nomes de atributo aninhados. Exemplos de nomes de atributo não vinculados incluem carimbos de data e hora, UUIDs e IDs de sessão. Embora você possa usar carimbos de data e hora e IDs de sessão como valores de atributo, recomendamos usar nomes de atributo mais curtos e previsíveis.
- Armazenamento em cache negativo: se ocorrer uma perda no cache e a leitura de uma tabela do DynamoDB não gerar nenhum item correspondente, o DAX adicionará uma entrada de cache negativo no respectivo cache de itens ou de consultas. Essa entrada permanecerá até a duração de TTL do cache expirar ou até ocorrer um write-through. O DAX continuará retornando essa entrada de cache negativo para solicitações futuras.

  Se o comportamento de armazenamento em cache negativo não se adequar ao padrão da aplicação, leia a tabela do DynamoDB diretamente quando o DAX retornar um resultado vazio. Também recomendamos que você defina uma duração de TTL mais curta para o cache a fim de evitar resultados vazios duradouros no cache e melhorar a consistência com a tabela.

Dimensionar um cluster do DAX

A capacidade e a disponibilidade totais de um cluster do DAX dependem do tipo e da quantidade de nós. Mais nós no cluster aumentam sua capacidade de leitura, mas não a capacidade de

gravação. Tipos de nó maiores (até r5.8xlarge) conseguem lidar com mais gravações, mas uma quantidade pequena de nós pode afetar a disponibilidade em caso de falha em um nó. Para obter mais informações sobre como dimensionar um cluster do DAX, consulte Guia de dimensionamento de clusters do DAX.

As seções a seguir discutem os diferentes aspectos de dimensionamento que você deve considerar para equilibrar o tipo e a contagem de nós a fim de criar um cluster escalável e econômico.

Nesta seção

- Disponibilidade de planejamento
- Planejar o throughput de gravação
- Planejar o throughput de leitura
- Planejar o tamanho do conjunto de dados
- Calcular os requisitos aproximados de capacidade do cluster
- Aproximar a capacidade de throughput do cluster por tipo de nó
- Ajustar a escala da capacidade de gravação em clusters do DAX

Disponibilidade de planejamento

Ao dimensionar um cluster do DAX, primeiro é necessário se concentrar na disponibilidade desejada. A disponibilidade de um serviço em clusters, como o DAX, é uma dimensão do número total de nós no cluster. Como um cluster de nó único não tolera falhas, sua disponibilidade é igual a um nó. Em um cluster com dez nós, a perda de um único nó tem um impacto mínimo na capacidade geral do cluster. Essa perda não tem um impacto direto na disponibilidade porque os nós restantes conseguirão atender às solicitações de leitura. Para retomar as gravações, o DAX aponta rapidamente um novo nó primário.

O DAX é baseado em VPC. Ele usa um grupo de sub-redes para determinar em quais zonas de disponibilidade ele pode executar nós e quais endereços IP usar das sub-redes. Para workloads de produção, é altamente recomendável usar o DAX com pelo menos três nós em zonas de disponibilidade diferentes. Isso garante que o cluster tenha mais de um nó para lidar com as solicitações, mesmo em caso de falha de um único nó ou zona de disponibilidade. Um cluster pode ter até 11 nós, sendo um nó primário e dez réplicas de leitura.

Planejar o throughput de gravação

Todos os clusters do DAX têm um nó primário para solicitações do tipo write-through. O tamanho do tipo de nó do cluster determina sua capacidade de gravação. A adição de réplicas de leitura não aumenta a capacidade de gravação do cluster. Portanto, você deve considerar a capacidade de gravação durante a criação do cluster, pois não poderá alterar o tipo de nó posteriormente.

Se a aplicação precisar gravar por meio do DAX para atualizar o cache de itens, considere aumentar o uso de recursos do cluster para facilitar a gravação. As gravações no DAX consomem cerca de 25 vezes mais recursos do que as leituras de acertos de cache. Isso pode exigir um tipo de nó maior em relação a clusters somente leitura.

Para obter orientação adicional sobre como determinar se a gravação do tipo write-through ou write-around funcionará melhor para sua aplicação, consulte Estratégias para gravações.

Planejar o throughput de leitura

A capacidade de leitura de um cluster do DAX depende da taxa de acertos de cache da workload. Como o DAX lê dados do DynamoDB quando ocorre uma perda no cache, ele consome aproximadamente dez vezes mais recursos do cluster do que um acerto de cache. Para aumentar os acertos de cache, aumente a configuração de TTL do cache para definir o período durante o qual um item é armazenado no cache. No entanto, uma duração de TTL mais longa aumenta a chance de ler versões mais antigas do item, a menos que as atualizações sejam gravadas no DAX por write-through.

Para garantir que o cluster tenha capacidade de leitura suficiente, ajuste a escala do cluster horizontalmente conforme mencionado em Escalar um cluster horizontalmente. A adição de mais nós adiciona réplicas de leitura ao pool de recursos, enquanto a remoção de nós reduz a capacidade de leitura. Ao selecionar a quantidade de nós e seus tamanhos para um cluster, considere as quantidades mínima e máxima de capacidade de leitura necessária. Se você não conseguir ajustar a escala de um cluster horizontalmente com tipos de nó menores para atender aos seus requisitos de leitura, use um tipo de nó maior.

Planejar o tamanho do conjunto de dados

Cada tipo de nó disponível tem um tamanho de memória definido para que o DAX armazene os dados em cache. Se um tipo de nó for muito pequeno, o conjunto de dados de trabalho solicitado por uma aplicação não caberá na memória e resultará em perdas no cache. Como nós maiores oferecem suporte a caches maiores, use um tipo de nó maior do que o conjunto de dados estimado que você precisa armazenar em cache. Um cache maior também melhora a taxa de acertos de cache.

Você pode obter retornos decrescentes ao armazenar itens em cache com poucas leituras repetidas. Calcule o tamanho da memória para itens acessados com frequência e verifique se o cache é grande o suficiente para armazenar esse conjunto de dados.

Calcular os requisitos aproximados de capacidade do cluster

Você pode estimar as necessidades de capacidade total da sua workload para ajudar a selecionar o tamanho e a quantidade adequados de nós do cluster. Para fazer essa estimativa, calcule a variável de solicitação por segundo normalizada (RPS normalizada). Essa variável representa o total de unidades de trabalho ao qual a aplicação exige que o cluster do DAX ofereça suporte, incluindo acertos de cache, perdas no cache e gravações. Para calcular a RPS normalizada, use as seguintes entradas:

- `ReadRPS_CacheHit`: especifica o número de leituras por segundo que resultam em um acerto de cache.
- `ReadRPS_CacheMiss`: especifica o número de leituras por segundo que resultam em uma perda no cache.
- `WriteRPS`: especifica o número de gravações por segundo que passarão pelo DAX.
- `DaxNodeCount`: especifica o número de nós no cluster do DAX.
- `Size`: especifica o tamanho do item que está sendo gravado ou lido em KB, arredondado para o número inteiro de KB mais próximo.
- `10x_ReadMissFactor`: representa um valor de 10. Quando ocorre uma perda no cache, o DAX usa aproximadamente dez vezes mais recursos do que em acertos de cache.
- `25x_WriteFactor`: representa um valor de 25 porque uma gravação por write-through no DAX usa aproximadamente 25 vezes mais recursos do que em acertos de cache.

Com a fórmula a seguir, você pode calcular a RPS normalizada.

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

Por exemplo, considere os seguintes dados de entrada:

- `ReadRPS_CacheHit` = 50.000
- `ReadRPS_CacheMiss` = 1.000
- `ReadMissFactor` = 1

- `Size` = 2 KB
- `WriteRPS` = 100
- `WriteFactor` = 1
- `DaxNodeCount` = 3

Ao substituir esses valores na fórmula, você pode calcular a RPS normalizada conforme mostrado a seguir.

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

Neste exemplo, o valor calculado da RPS normalizada é 135.000. No entanto, esse valor de RPS normalizada não considera a manutenção da utilização do cluster abaixo de 100% nem a perda de nós. Recomendamos que você leve em consideração uma capacidade adicional. Para fazer isso, determine o maior entre estes dois fatores de multiplicação: utilização pretendida ou tolerância à perda de nós. Depois, multiplique a RPS normalizada pelo fator maior para obter uma solicitação por segundo pretendida (RPS pretendida).

- Utilização pretendida

  Como os impactos no desempenho aumentam as perdas no cache, não recomendamos executar o cluster do DAX com 100% de utilização. Idealmente, você deve manter a utilização do cluster igual ou inferior a 70%. Para alcançar isso, multiplique a RPS normalizada por 1,43.

- Tolerância à perda de nós

  Se um nó falhar, a aplicação deverá ser capaz de distribuir as solicitações entre os nós restantes. Para garantir que os nós permaneçam abaixo de 100% de utilização, escolha um tipo de nó grande o suficiente para absorver tráfego extra até que o nó com falha volte a ficar on-line. Para um cluster com menos nós, cada nó deve tolerar aumentos maiores de tráfego em caso de falha em um nó.

  Se um nó primário falhar, o DAX recuperará automaticamente uma réplica de leitura e a designará como novo nó primário. Se um nó de réplica falhar, outros nós do cluster do DAX ainda poderão atender às solicitações até que o nó com falha seja recuperado.

  Por exemplo, um cluster do DAX de três nós com uma falha de nó requer uma capacidade adicional de 50% nos dois nós restantes. Isso exige um fator multiplicador de 1,5. Por outro lado,

um cluster de 11 nós com falha em um nó requer uma capacidade adicional de 10% nos nós restantes, ou um fator de multiplicação de 1,1.

Com a fórmula a seguir, você pode calcular a RPS pretendida.

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

Por exemplo, para calcular a RPS pretendida, considere os seguintes valores:

- `Normalized RPS` = 135.000
- `TargetUtilization` = 1,43

  Como nosso objetivo é uma utilização máxima do cluster de 70%, vamos definir `TargetUtilization` como 1,43.
- `NodeLossTolerance` = 1,5

  Digamos que estamos usando um cluster de três nós, portanto vamos definir `NodeLossTolerance` como 50% da capacidade.

Ao substituir esses valores na fórmula, você pode calcular a RPS pretendida conforme mostrado a seguir.

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

Neste exemplo, como o valor de `NodeLossTolerance` é maior que `TargetUtilization`, calculamos o valor da RPS pretendida com `NodeLossTolerance`. Isso nos dá uma RPS pretendida de 202.500, que é a quantidade total de capacidade à qual o cluster do DAX deve oferecer suporte. Para determinar o número de nós necessários em um cluster, mapeie a RPS pretendida para uma coluna apropriada na tabela a seguir. Para este exemplo de RPS pretendida de 202.500, é necessário o tipo de nó dax.r5.large com três nós.

Aproximar a capacidade de throughput do cluster por tipo de nó

Usando a Target RPS formula, você pode estimar a capacidade do cluster para diferentes tipos de nó. A tabela a seguir mostra as capacidades aproximadas para clusters com 1, 3, 5 e 11 tipos de nó. Essas capacidades não substituem a necessidade de realizar testes de carga do DAX com padrões de solicitação e dados próprios. Além disso, essas capacidades não incluem instâncias do tipo t

devido à falta de capacidade fixa de CPU. A unidade para todos os valores na tabela a seguir é a RPS normalizada.

Tipo de nó (memória)	1 nó	3 nós	5 nós	11 nós
dax.r5.24xlarge (768 GB)	1 milhão	3 milhões	5 milhões	11 milhões
dax.r5.16xlarge (512 GB)	1 milhão	3 milhões	5 milhões	11 milhões
dax.r5.12xlarge (384 GB)	1 milhão	3 milhões	5 milhões	11 milhões
dax.r5.8xlarge (256 GB)	1 milhão	3 milhões	5 milhões	11 milhões
dax.r5.4xlarge (128 GB)	600 mil	1,8 milhão	3 milhões	6,6 milhões
dax.r5.2xlarge (64 GB)	300 mil	900 mil	1,5 milhão	3,3 milhões
dax.r5.xlarge (32 GB)	150 mil	450 mil	750 mil	1,65 milhão
dax.r5.large (16 GB)	75 mil	225 mil	375 mil	825 mil

Devido ao limite máximo de 1 milhão de NPS (operações de rede por segundo) para cada nó, os nós do tipo dax.r5.8xlarge ou maiores não contribuem com capacidade adicional para o cluster. Tipos de nó maiores que 8xlarge podem não contribuir para a capacidade de throughput total do cluster. No entanto, esses tipos de nó podem ser úteis para armazenar um conjunto maior de dados de trabalho na memória.

Ajustar a escala da capacidade de gravação em clusters do DAX

Cada gravação no DAX consome 25 solicitações normalizadas em cada nó. Como há um limite de 1 milhão de RPS para cada nó, um cluster do DAX é limitado a 40.000 gravações por segundo, sem considerar o uso de leitura.

Se seu caso de uso exigir mais de 40.000 gravações por segundo no cache, você deverá usar clusters do DAX separados e fragmentar as gravações entre eles. Semelhante ao DynamoDB, você pode fazer o hash da chave de partição dos dados que está gravando no cache. Depois, utilize o cálculo de módulo para determinar em qual fragmento gravar os dados.

O exemplo a seguir calcula o hash de uma string de entrada. Depois, é calculado o módulo do valor de hash com 10.

```
def hash_modulo(input_string):
    # Compute the hash of the input string
    hash_value = hash(input_string)

    # Compute the modulus of the hash value with 10
    bucket_number = hash_value % 10

    return bucket_number

#Example usage
if _name_ == "_main_":
    input_string = input("Enter a string: ")
    result = hash_modulo(input_string)
    print(f"The hash modulo 10 of '{input_string}' is: {result}.")
```

Implantar um cluster

A criação de um novo cluster do DAX exige configurações além das necessárias para o DynamoDB. Essas configurações são especialmente para redes porque o DAX é baseado na Amazon VPC. Isso oferece controle total sobre seu ambiente de rede virtual, incluindo posicionamento de recursos, conectividade e segurança. Esta seção apresenta as práticas recomendadas para as configurações necessárias durante a criação de um cluster.

Para obter mais informações sobre como escolher nós de cluster, consulte Dimensionar um cluster do DAX.

Nesta seção

- Configurar redes
- Configurar a segurança
- Grupo de parâmetros
- Janela de manutenção

Configurar redes

O DAX usa um grupo de sub-redes para determinar em quais zonas de disponibilidade ele pode executar nós e quais endereços IP usar das sub-redes. Para minimizar a latência entre a aplicação e o DAX, as sub-redes e zonas de disponibilidade dos servidores da sua aplicação e do cluster do DAX devem ser as mesmas.

Recomendamos que você distribua os nós do DAX entre várias zonas de disponibilidade. A opção padrão de alocação automática faz isso por você.

Para conferir as práticas recomendadas sobre como configurar uma VPC, consulte Conceitos básicos da Amazon VPC no Manual do usuário da Amazon VPC.

Configurar a segurança

Esta seção discute as medidas de segurança que você deve implementar para aplicações que usam o DAX. Esta seção também discute brevemente o suporte do DAX à criptografia de dados.

IAM

O DAX e o DynamoDB têm mecanismos de controle de acesso separados. O DAX exige um perfil do IAM para acessar suas tabelas do DynamoDB. Esse perfil deve seguir o princípio do privilégio mínimo e conceder acesso somente a tabelas e operações específicas do DynamoDB, como GetItem e PutItem. Para obter mais informações sobre os mecanismos de controle de acesso fornecidos pelo DAX, consulte Controle de acesso do DAX.

Criptografia

Configure a criptografia em repouso e a criptografia em trânsito ao criar um cluster do DAX. Essas opções são habilitadas por padrão. Recomendamos que você mantenha as configurações de criptografia padrão, a menos que os requisitos comerciais não permitam. Para obter mais informações, consulte Criptografia em repouso do DAX e Criptografia em trânsito do DAX.

Grupo de parâmetros

O DAX aplica um conjunto de configurações em cada nó em um cluster chamado de grupo de parâmetros. Você pode alterar essa configuração depois de criar o cluster.

O grupo de parâmetros do DAX contém configurações de TTL para cache de itens e cache de consultas. A duração de TTL padrão é de 5 minutos. Você pode substituir a duração de TTL por qualquer valor inteiro maior ou igual a 1 milissegundo.

Não é possível modificar grupos de parâmetros quando há uma instância do DAX em execução usando eles. Você pode alterar os valores do grupo de parâmetros durante o tempo de inatividade de um cluster do DAX.

Janela de manutenção

Para permitir atualizações e patches de software ocasionais em seus nós, uma janela de manutenção semanal é configurada para o cluster do DAX. Durante essa janela, o DAX executa atualizações contínuas nos nós. Clusters com mais de um nó não perdem a disponibilidade durante essas atualizações, mas têm capacidade reduzida até que o nó retorne. Se sua organização tiver um tempo previsível de baixo uso, considere definir a janela de manutenção manualmente para esse horário.

Gerenciar as operações do cluster

O DAX cuida da manutenção e da integridade do cluster para você. No entanto, você precisa fornecer informações operacionais para escalar o cluster horizontal ou verticalmente de acordo com seus padrões de uso. Esta seção descreve o processo recomendado para escalar clusters do DAX.

Nesta seção

- Escalar um cluster horizontalmente
- Escalar um cluster verticalmente

Escalar um cluster horizontalmente

O ajuste da escala de um cluster do DAX envolve o ajuste de sua capacidade para atender às demandas de throughput. Esse ajuste é feito aumentando ou diminuindo o número de nós (réplicas) no cluster enquanto ele está em execução. Esse processo, conhecido como escalabilidade horizontal, ajuda a distribuir a workload em mais nós ou a consolidá-la em menos nós quando a demanda é baixa.

Você pode aumentar e diminuir a escala de um cluster do DAX horizontalmente usando os comandos `decrease-replication-factor` ou `increase-replication-factor` na AWS CLI.

Aumentar o fator de replicação (aumentar a escala horizontalmente)

Aumentar o fator de replicação de um cluster do DAX adiciona mais nós ao cluster. O exemplo a seguir mostra o uso do comando `increase-replication-factor`.

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- Nesse comando, o argumento `cluster-name` especifica o nome do cluster. Por exemplo, *`yourClusterName`*.
- O argumento `new-replication-factor` especifica o número total de nós que devem ser adicionados ao cluster depois do ajuste de escala. Isso inclui o nó primário e os nós de réplica. Por exemplo, se o cluster tiver três nós e você quiser adicionar mais dois nós, defina o valor de `new-replication-factor` como 5.

Diminuir o fator de replicação (reduzir a escala horizontalmente)

Diminuir o fator de replicação de um cluster do DAX remove nós do cluster. A remoção de nós pode ajudar a reduzir os custos durante períodos de baixa demanda. O exemplo a seguir mostra o uso do comando `decrease-replication-factor`.

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- Nesse comando, o argumento `cluster-name` especifica o nome do cluster. Por exemplo, *`yourClusterName`*.
- O argumento `new-replication-factor` especifica o número reduzido de nós no cluster depois do ajuste de escala. Esse número precisa ser menor que o fator de replicação atual e deve incluir o nó primário. Por exemplo, se o cluster tiver cinco nós e você quiser remover dois nós, defina o valor de `new-replication-factor` como 3.

Considerações sobre a escalabilidade horizontal

Considere o seguinte ao planejar a escalabilidade horizontal:

- Nó primário: o cluster do DAX inclui um nó primário. O fator de replicação inclui esse nó primário. Por exemplo, um fator de replicação de 3 significa um nó primário e dois nós de réplica.
- Disponibilidade: a adição ou remoção de nós do DAX altera a disponibilidade e a tolerância a falhas do cluster. Mais nós podem melhorar a disponibilidade, mas também aumentam os custos.
- Migração de dados: quando você aumenta o fator de replicação, o DAX gerencia automaticamente a distribuição de dados entre o novo conjunto de nós. Quando um novo nó começa a fornecer tráfego, seu cache já está aquecido. No entanto, durante esse processo, pode haver um impacto temporário no desempenho durante a migração de dados.

Monitore os clusters do DAX com atenção durante e após o processo de escalabilidade para garantir que estejam funcionando conforme o esperado e faça ajustes adicionais se houver necessidade.

Escalar um cluster verticalmente

Para escalar verticalmente o tamanho de nó de um cluster existente, é necessário criar um cluster e migrar o tráfego da aplicação para o novo cluster. A migração para um novo cluster com nós diferentes envolve várias etapas para garantir uma transição suave com impacto mínimo no desempenho e na disponibilidade da aplicação.

Para criar um cluster com o intuito de escalar verticalmente o tamanho de um nó, considere os seguintes pontos:

- Acesse sua configuração atual: revise as métricas do cluster do DAX atual para determinar o novo tamanho e a quantidade de nós necessários. Use essas informações para definir o tamanho do cluster. Para mais informações, consulte Dimensionar um cluster do DAX.
- Configure um novo cluster do DAX: crie um cluster do DAX com o tipo de nó e a quantidade que você determinou. Você pode usar as configurações existentes do seu grupo de parâmetros, a menos que precise fazer ajustes.
- Sincronize os dados: como o DAX é uma camada de armazenamento em cache para o DynamoDB, não é necessário migrar os dados diretamente. No entanto, o novo cluster do DAX não terá nenhum conjunto de dados funcional na memória até que você envie tráfego para ele.
- Atualize a configuração da aplicação: atualize a configuração da sua aplicação para apontar para o endpoint do novo cluster do DAX. Talvez seja necessário alterar o código ou atualizar as variáveis de ambiente, dependendo da configuração da aplicação.

Para reduzir o impacto ao mudar para um novo cluster, envie tráfego canário para o novo cluster de uma pequena parte da sua frota de aplicações. Para fazer isso, implemente lentamente as atualizações da aplicação ou use uma entrada DNS de roteamento baseado em peso na frente do endpoint do DAX.

- Monitore e otimize: depois de mudar para o novo cluster do DAX, monitore atentamente suas métricas e logs de desempenho em busca de quaisquer problemas. Prepare-se para ajustar o número de nós com base nos padrões atualizados da workload.

  Até que o novo cluster armazene seu conjunto de dados de trabalho em cache adequadamente, você verá latências e taxas de perda no cache mais altas.
- Desative o cluster antigo: quando tiver certeza de que o novo cluster está funcionando conforme o esperado, desative com segurança o antigo cluster do DAX para evitar custos desnecessários.

Monitoramento do DAX

Você pode monitorar as principais métricas, como a taxa de acertos de cache, para garantir o desempenho ideal do cluster do DAX, diagnosticar problemas e determinar o momento de escalar o cluster. A verificação regular das principais métricas ajuda você a manter o desempenho, a estabilidade e a economia, ajustando a escala do cluster para atender às necessidades da workload. Para obter mais informações sobre o monitoramento do DAX, consulte Monitoramento da produção.

A seguinte lista apresenta algumas das principais métricas que você deve monitorar:

- Taxa de acertos de cache: mostra a eficiência com que o DAX fornece dados que estão armazenados em cache, reduzindo a necessidade de acessar as tabelas subjacentes do DynamoDB. Poucas perdas no cache para o cluster indicam boa eficiência de armazenamento em cache. Por outro lado, poucos acertos de cache sugerem que talvez seja necessário rever a configuração de TTL do armazenamento em cache ou que a workload não é adequada para armazenamento em cache.

  Use o Amazon CloudWatch para calcular a taxa de acertos de cache do cluster do DAX. Compare as métricas `ItemCacheHits`, `ItemCacheMisses`, `QueryCacheHits` e `QueryCacheMisses` para obter essa taxa. A fórmula a seguir mostra como a taxa de acertos de cache é calculada. Para calcular a taxa usando essa fórmula, divida a quantidade de acertos de cache pela soma dos acertos e das perdas no cache.

```
Cache hit ratio = Cache hits / (Cache hits + Cache misses)
```

A taxa de acertos de cache é um número entre 0 e 1, que é representado como uma porcentagem. Quanto maior for essa porcentagem, melhor será a utilização geral do cache.

- ErrorRequestCount: contagem de solicitações que resultaram em erros de usuário relatados pelo nó ou cluster. `ErrorRequestCount` inclui solicitações que foram limitadas pelo nó ou cluster. O monitoramento de erros do usuário pode ajudar você a identificar configurações incorretas de escalabilidade ou padrões de itens/partições ativos na aplicação.
- Latências de operação: o monitoramento da latência das operações de leitura e gravação de e para o cluster do DAX pode ajudar você a identificar gargalos de desempenho. O aumento das latências pode indicar problemas na rede ou na configuração do cluster do DAX, ou a necessidade de escalar.
- Consumo da rede: fique de olho nas métricas `NetworkBytesIn` e `NetworkBytesOut` para monitorar o tráfego de rede do cluster do DAX. Um aumento inesperado no throughput da rede pode significar mais solicitações de clientes ou padrões de consulta ineficientes que estão fazendo com que mais dados sejam transferidos.

  O monitoramento do consumo da rede ajuda você a gerenciar os custos do cluster do DAX. Também garante que a rede não se torne um gargalo para o desempenho do cluster.
- Taxa de remoção: mostra com que frequência os itens são removidos do cache para abrir espaço para novos itens. Se a taxa de remoção aumentar com o tempo, isso poderá significar que o cache é muito pequeno ou que sua estratégia de armazenamento em cache não é eficaz.

  Monitore a métrica `EvictedSize` no CloudWatch para determinar se o tamanho do cache é adequado para a workload. Se o tamanho total removido continuar aumentando, talvez seja necessário aumentar a escala do cluster do DAX verticalmente para acomodar um cache maior.
- Utilização da CPU: refere-se ao percentual de utilização da CPU pelo nó ou cluster. Essa é uma métrica essencial para monitorar qualquer banco de dados ou sistema de armazenamento em cache. Uma alta utilização da CPU pode significar que o cluster do DAX pode estar sobrecarregado e precisar passar por ajuste da escala para lidar com o aumento da demanda.

  Monitore a métrica `CPUUtilization` do cluster do DAX. Se a utilização da CPU se aproximar ou exceder consistentemente o patamar de 70–80%, considere aumentar a escala do cluster do DAX verticalmente conforme descrito na seção a seguir.

Se o número de solicitações enviadas ao DAX exceder a capacidade de um nó, o DAX limitará a taxa em que aceita solicitações adicionais. Ele faz isso retornando uma ThrottlingException. O DAX avalia continuamente a utilização da CPU para determinar o volume de solicitações que consegue processar enquanto mantém um estado de cluster íntegro.

Você pode monitorar a métrica `ThrottledRequestCount` publicada pelo DAX no CloudWatch. Se você vir essas exceções regularmente, considere aumentar o cluster.

Escalar um cluster do DAX usando dados de monitoramento

Você pode determinar se precisa aumentar ou reduzir a escala do cluster do DAX verticalmente monitorando suas métricas de desempenho.

- Aumentar a escala vertical ou horizontalmente: se o cluster do DAX tiver alta utilização da CPU, poucos acertos de cache (após otimizar a estratégia de armazenamento em cache) ou altas latências de operação, você deverá aumentar a escala verticalmente do cluster. A adição de nós, também chamada de aumento horizontal da escala, pode ajudar a distribuir a carga de maneira mais uniforme. Para workloads com quantidades cada vez maiores de gravações por segundo, pode ser necessário escolher nós mais poderosos (aumentar a escala verticalmente).
- Reduzir a escala verticalmente: se você observar consistentemente baixa utilização da CPU e latências de operação abaixo dos limites, talvez tenha provisionado recursos em excesso. Nesses casos, reduza a escala de nós verticalmente para reduzir os custos. Você pode reduzir o número de nós para 1 durante períodos de baixa utilização, mas não pode desligar totalmente o cluster.

Usar o DynamoDB com outros serviços da AWS

O Amazon DynamoDB é integrado com outros serviços da AWS, permitindo que você automatize tarefas repetitivas ou crie aplicações que abrangem vários serviços.

Tópicos

- Configurar credenciais da AWS usando o Amazon Cognito para DynamoDB
- Integração com o Amazon Redshift
- Processar dados do DynamoDB com o Apache Hive no Amazon EMR
- Integração do DynamoDB ao Amazon S3
- Integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse
- Integração ETL zero do DynamoDB com o Amazon OpenSearch Service
- Integrar o DynamoDB com o Amazon EventBridge
- Integrar o DynamoDB com o Amazon Managed Streaming for Apache Kafka
- Práticas recomendadas para integração com o DynamoDB

Configurar credenciais da AWS usando o Amazon Cognito para DynamoDB

A maneira recomendada para obter credenciais da AWS para suas aplicações móveis e da Web é usar o Amazon Cognito. O Amazon Cognito ajuda a evitar a codificação das suas credenciais da AWS em seus arquivos. Ele usa funções do AWS Identity and Access Management (IAM) para gerar credenciais temporárias para os usuários autenticados e não autenticados da sua aplicação.

Por exemplo, para configurar seus arquivos JavaScript para usar uma função não autenticada do Amazon Cognito para acessar o serviço da Web Amazon DynamoDB, faça o seguinte:

Para configurar credenciais a serem integradas ao Amazon Cognito

1. Crie um pool de identidades do Amazon Cognito que permita identidades não autenticadas.

```
aws cognito-identity create-identity-pool \
    --identity-pool-name DynamoPool \
    --allow-unauthenticated-identities \
    --output json
```

```
{
    "IdentityPoolId": "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
    "AllowUnauthenticatedIdentities": true,
    "IdentityPoolName": "DynamoPool"
}
```

2. Copie a seguinte política para um arquivo denominado `myCognitoPolicy.json`. Substitua o ID do grupo de identidades (*`us-west-2:12345678-1ab2-123a-1234-a12345ab12`*) pelo seu próprio `IdentityPoolId` obtido na etapa anterior.

   JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-
west-2:12345678-1ab2-123a-1234-a12345ab12"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

3. Crie uma função do IAM que assuma a política anterior. Dessa forma, o Amazon Cognito se torna uma entidade confiável capaz de assumir a função `Cognito_DynamoPoolUnauth`.

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. Conceda à função `Cognito_DynamoPoolUnauth` acesso total ao DynamoDB anexando uma política gerenciada (`AmazonDynamoDBFullAccess`).

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> Como alternativa, você pode conceder acesso refinado ao DynamoDB. Para obter mais informações, consulte Usar condições de política do IAM para um controle de acesso refinado.

5. Obtenha e copie o nome do recurso da Amazon (ARN) da função do IAM.

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. Adicione a função `Cognito_DynamoPoolUnauth` ao pool de identidades `DynamoPool`. O formato a ser especificado é `KeyName=string`, onde `KeyName` é `unauthenticated` e a string é o ARN da função obtido na etapa anterior.

```
aws cognito-identity set-identity-pool-roles \
--identity-pool-id "us-west-2:12345678-1ab2-123a-1234-a12345ab12" \
--roles unauthenticated=arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth --
output json
```

7. Especifique as credenciais do Amazon Cognito em seus arquivos. Modifique `IdentityPoolId` e `RoleArn` de acordo.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

Você agora pode executar seus programas JavaScript no serviço da Web DynamoDB usado credenciais do Amazon Cognito. Para ter mais informações, consulte Setting credentials in a web browser no Guia de conceitos básicos do AWS SDK para JavaScript.

Integração com o Amazon Redshift

O Amazon Redshift é um serviço de data warehouse rápido, totalmente gerenciado e em escala de petabytes que torna simples e econômica a análise eficiente de todos os seus dados usando as ferramentas de business intelligence existentes.

O DynamoDB e o Amazon Redshift podem ser usados juntos para atender às diferentes necessidades de armazenamento e processamento de dados em uma aplicação ou um ecossistema de dados.

Consulte os tópicos a seguir para obter mais detalhes sobre como integrar o DynamoDB ao Amazon Redshift.

Tópicos

- Considerações sobre a integração entre contas com a CMK
- Integração ETL zero do DynamoDB com o Amazon Redshift
- Enviar dados do DynamoDB ao Amazon Redshift com o comando COPY

Considerações sobre a integração entre contas com a CMK

Quando você tenta fazer a integração com o Amazon Redshift pelo DynamoDB, a ação inicial é iniciada pelo Amazon Redshift. Sem as permissões adequadas, essa ação pode gerar uma falha silenciosa. As seções a seguir detalham as permissões necessárias para essa integração entre contas.

Políticas e permissões obrigatórias do AWS KMS

Substitua os seguintes espaços reservados nos exemplos.

- `REDSHIFT_ACCOUNT_ID`: o ID da Conta da AWS em que o Amazon Redshift está hospedado.
- `DYNAMODB_ACCOUNT_ID`: o ID da Conta da AWS em que o DynamoDB está hospedado.
- `REDSHIFT_ROLE_NAME`: o nome do perfil do IAM usado pelo Amazon Redshift.
- `REGION`: a Região da AWS em que seus recursos estão localizados.
- `TABLE_NAME`: o nome da sua tabela do DynamoDB.
- `KMS_KEY_ID`: o ID da sua chave do KMS.

Política de chave do KMS na conta do DynamoDB

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::DYNAMODB_ACCOUNT_ID:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow Redshift to use the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::REDSHIFT_ACCOUNT_ID:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "*"
        }
    ]
}
```

Política do IAM para o perfil do Amazon Redshift (na conta do Amazon Redshift)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:ParitalBatchGetItem",
                "dynamodb:Scan",
```

```
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME",
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME/stream/*"
            ]
        },
        {
            "Sid": "AllowKMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:REGION:DYNAMODB_ACCOUNT_ID:key/KMS_KEY_ID"
        }
    ]
}
```

Relação de confiança para o perfil do Amazon Redshift

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Política de tabela do DynamoDB (se estiver usando políticas baseadas em recursos)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRedshiftAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::REDSHIFT_ACCOUNT_ID:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:ParitalBatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME",
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME/stream/*"
            ]
        }
    ]
}
```

Considerações importantes

1. Verifique se a chave do KMS está na mesma região que sua tabela do DynamoDB.
2. A chave do KMS deve ser uma chave gerenciada pelo cliente (CMK), não uma Chave gerenciada pela AWS.
3. Se você estiver usando tabelas globais do DynamoDB, configure as permissões para todas as regiões relevantes.
4. Considere a possibilidade de adicionar declarações condicionais para restringir o acesso com base em endpoints da VPC ou intervalos de IP.

5. Para aumentar a segurança, considere a possibilidade de usar a condição `aws:PrincipalOrgID` para restringir o acesso à sua organização.
6. Monitore o uso da chave do KMS por meio das métricas do CloudTrail e do CloudWatch.

Integração ETL zero do DynamoDB com o Amazon Redshift

A integração ETL zero do Amazon DynamoDB com o Amazon Redshift permite usar analytics nos dados do DynamoDB, sem nenhuma codificação e sem complicação. Esse recurso totalmente gerenciado replica automaticamente as tabelas do DynamoDB em um banco de dados do Amazon Redshift para que os usuários possam executar consultas SQL e analytics em seus dados do DynamoDB sem precisar configurar processos complexos de ETL. A integração se dá com a replicação dos dados da tabela do DynamoDB para o banco de dados do Amazon Redshift.

Para configurar a integração, basta especificar uma tabela do DynamoDB como origem e um banco de dados do Amazon Redshift como destino. Na ativação, a integração exporta a tabela completa do DynamoDB para preencher o banco de dados do Amazon Redshift. O tempo necessário para que esse processo inicial seja concluído depende do tamanho da tabela do DynamoDB. A integração ETL zero então replica incrementalmente as atualizações do DynamoDB para o Amazon Redshift a cada 15-30 minutos usando exportações incrementais do DynamoDB. Isso significa que a atualização dos dados replicados do DynamoDB no Amazon Redshift é feita automaticamente.

Depois de configurados, os usuários podem analisar os dados do DynamoDB no Amazon Redshift usando clientes e ferramentas SQL padrão, sem afetar o desempenho da tabela do DynamoDB. Pelo fato de eliminar as complicações de ETL, essa integração ETL zero oferece uma maneira rápida e fácil de revelar insights do DynamoDB por meio dos recursos de analytics e machine learning do Amazon Redshift.

Tópicos

- Pré-requisitos antes de criar uma integração ETL zero do DynamoDB com o Amazon Redshift
- Limitações ao usar integrações ETL zero do DynamoDB com o Amazon Redshift
- Criar integrações ETL zero do DynamoDB com o Amazon Redshift
- Visualizar integrações ETL zero do DynamoDB com o Amazon Redshift
- Excluir integrações ETL zero do DynamoDB com o Amazon Redshift

Pré-requisitos antes de criar uma integração ETL zero do DynamoDB com o Amazon Redshift

1. Você deve criar a tabela de origem do DynamoDB e o cluster Amazon Redshift de destino para só então criar uma integração. Essas informações são abordadas em Etapa 1: configurar uma tabela de origem do DynamoDB e Etapa 2: criar um data warehouse do Amazon Redshift.
2. Uma integração ETL zero entre o Amazon DynamoDB e o Amazon Redshift exige que sua tabela de origem do DynamoDB tenha a recuperação para um ponto no tempo (PITR) habilitada.
3. Com relação a políticas baseadas em recursos, se você criar uma integração em que a tabela do DynamoDB e o data warehouse do Amazon Redshift estejam na mesma conta, poderá usar a opção Corrigir para mim durante a etapa de criação da integração para aplicar automaticamente as políticas de recursos necessárias ao DynamoDB e ao Amazon Redshift.

   Se você criar uma integração em que a tabela do DynamoDB e o data warehouse do Amazon Redshift estejam em contas AWS diferentes, precisará aplicar a política de recursos a seguir à tabela do DynamoDB.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": [
                "dynamodb:ExportTableToPointInTime",
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
```

```
            }
        },
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "dynamodb:DescribeExport",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/table-name/
export/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        }
    ]
}
```

Talvez você também precise configurar a política de recursos no data warehouse do Amazon Redshift. Para ter mais informações, consulte Configurar a autorização na API do Amazon Redshift.

4. Em relação a políticas baseadas em identidade:

   a. O usuário que cria a integração exige uma política baseada em identidade que autorize as seguintes ações: `GetResourcePolicy`, `PutResourcePolicy` e `UpdateContinuousBackups`.

      > **Note**
      >
      > Os exemplos de política a seguir mostrarão o recurso como `arn:aws:redshift{-serverless}`. Este é um exemplo para mostrar que o ARN pode ser `arn:aws:redshift` ou `arn:aws:redshift-serverless`, dependendo se seu

namespace é um cluster do Amazon Redshift ou um namespace do Amazon Redshift sem servidor.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetResourcePolicy",
                "dynamodb:PutResourcePolicy",
                "dynamodb:UpdateContinuousBackups"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/table-name"
            ]
        },
        {
            "Sid": "AllowRedshiftDescribeIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRedshiftCreateIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
        },
```

```
        {
            "Sid": "AllowRedshiftModifyDeleteIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:ModifyIntegration",
                "redshift:DeleteIntegration"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:uuid"
        },
        {
            "Sid": "AllowRedshiftCreateInboundIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateInboundIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:namespace:uuid"
        }
    ]
}
```

b. O usuário responsável por configurar o namespace de destino do Amazon Redshift exige uma política baseada em identidade que autorize as seguintes ações: `PutResourcePolicy`, `DeleteResourcePolicy` e `GetResourcePolicy`.

   JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "redshift:PutResourcePolicy",
                "redshift:DeleteResourcePolicy",
                "redshift:GetResourcePolicy"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "redshift:DescribeInboundIntegrations"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        }
    ]
}
```

5. Permissões de criptografia de chaves

   Se a tabela de origem do DynamoDB for criptografada usando uma chave do AWS KMS gerenciada pelo cliente, você precisará adicionar a política a seguir à chave do KMS. Essa política permite que o Amazon Redshift exporte dados da tabela criptografada usando a chave do KMS.

```
{
    "Sid": "Statement to allow Amazon Redshift service to perform Decrypt operation
 on the source DynamoDB Table",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "redshift.amazonaws.com"
        ]
    },
    "Action": "kms:Decrypt",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "<account>"
        },
        "ArnEquals": {
            "aws:SourceArn": "arn:aws:redshift:<region>:<account>:integration:*"
        }
    }
}
```

Você também pode seguir as etapas em Conceitos básicos das integrações ETL zero no “Guia de gerenciamento do Amazon Redshift” para configurar as permissões de namespace do Amazon Redshift.

Limitações ao usar integrações ETL zero do DynamoDB com o Amazon Redshift

As limitações gerais a seguir se aplicam à versão atual dessa integração. Essas limitações podem mudar nas versões subsequentes.

> **Note**
>
> Além das limitações abaixo, examine também as considerações gerais ao usar integrações ETL zero. Consulte Considerações ao usar integrações ETL zero com o Amazon Redshift no Guia de gerenciamento do Amazon Redshift.

- A tabela do DynamoDB e o cluster do Amazon Redshift precisam estar na mesma região.
- A tabela de origem do DynamoDB deve ser criptografada com uma chave do AWS KMS de propriedade da Amazon ou gerenciada pelo cliente. A criptografia gerenciada pela Amazon não se aplica à tabela de origem do DynamoDB.

Criar integrações ETL zero do DynamoDB com o Amazon Redshift

Antes de criar uma integração ETL zero, primeiro especifique a tabela de origem do DynamoDB e, em seguida, o data warehouse de destino do Amazon Redshift.

Etapa 1: configurar uma tabela de origem do DynamoDB

Para criar uma integração ETL zero com o Amazon Redshift, é necessário habilitar a recuperação para um ponto no tempo (PITR) na tabela. Se você não tiver o PITR ativado, o console poderá corrigir isso para você durante o processo de configuração da integração. Para obter detalhes sobre como habilitar a PITR, consulte Recuperação para um ponto no tempo.

Etapa 2: criar um data warehouse do Amazon Redshift

Se você ainda não tem um data warehouse do Amazon Redshift, é possível criar um. Para criar um grupo de trabalho do Amazon Redshift sem servidor com um namespace, consulte Criar um grupo de trabalho com um namespace. Para criar um cluster do Amazon Redshift, consulte Criar um cluster.

O grupo de trabalho ou cluster de destino do Amazon Redshift deve ter o parâmetro enable_case_sensitive_identifier ativado para que a integração seja bem-sucedida. Para obter mais informações sobre como habilitar a diferenciação de maiúsculas e minúsculas, consulte Turn on case sensitivity for your data warehouse no “Guia de gerenciamento do Amazon Redshift”.

Depois que a configuração do grupo de trabalho ou cluster do Amazon Redshift estiver concluída, você precisará configurar o data warehouse. Consulte Integrações ETL zero no “Guia de gerenciamento do Amazon Redshift” para obter mais informações.

Etapa 3: criar uma integração ETL zero do DynamoDB

Antes de criar uma integração ETL zero, conclua as tarefas na seção intitulada Pré-requisitos antes de criar uma integração ETL zero do DynamoDB com o Amazon Redshift. O processo de criação de uma integração entre o DynamoDB e Amazon Redshift envolve duas etapas. Primeiro, crie uma integração no DynamoDB e, em seguida, anexe um banco de dados do Amazon Redshift a essa integração recém-criada.

Criar uma Integração ETL zero

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon DynamoDB em https://console.aws.amazon.com/dynamodbv2.
2. No painel de navegação, selecione Integrações.
3. Selecione Criar integração ETL zero e escolha Amazon Redshift.
4. Isso levará você ao console do Amazon Redshift. Para continuar com o procedimento, consulte a seção DynamoDB em Create a zero-ETL integration for DynamoDB.

Visualizar integrações ETL zero do DynamoDB com o Amazon Redshift

Você pode visualizar os detalhes de uma integração ETL zero para ver as respectivas informações de configuração e status atual.

Como visualizar os detalhes de uma integração ETL zero no console do Amazon DynamoDB:

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon DynamoDB em https://console.aws.amazon.com/dynamodbv2.
2. No console do DynamoDB, escolha Integrações.
3. No painel Integração ETL zero, selecione a integração ETL zero que você deseja visualizar.

Para visualizar os detalhes de uma integração ETL zero no console do Amazon Redshift:

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon Redshift em https://console.aws.amazon.com/redshiftv2.

2. Siga as etapas em Visualizar integrações ETL zero.

> **Note**
>
> Os possíveis status de uma integração ETL zero com o Amazon Redshift estão listados em Visualizar Integrações ETL zero no Guia de gerenciamento do Amazon Redshift.

Excluir integrações ETL zero do DynamoDB com o Amazon Redshift

Quando você exclui uma integração ETL zero, os dados não são excluídos do DynamoDB ou do Amazon Redshift, mas o DynamoDB deixa de enviar dados da tabela de origem ao destino do Amazon Redshift.

Como excluir uma integração ETL zero

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon DynamoDB em https://console.aws.amazon.com/dynamodbv2.
2. No console do DynamoDB, escolha Integrações.
3. No painel Integração ETL zero, selecione a integração ETL zero que você deseja excluir.
4. Escolha Gerenciar. Isso levará você para a página de detalhes da integração.
5. Para confirmar a exclusão, selecione Excluir.

Enviar dados do DynamoDB ao Amazon Redshift com o comando COPY

O Amazon Redshift trabalha com o Amazon DynamoDB usando recursos avançados de business intelligence e uma poderosa interface baseada em SQL. Ao copiar os dados de uma tabela do DynamoDB para o Amazon Redshift, você pode realizar consultas de análise de dados complexas nesses dados, incluindo junções com outras tabelas em seu cluster do Amazon Redshift.

Em termos de throughput provisionado, uma operação de cópia de uma tabela do DynamoDB entra na contagem da capacidade de leitura dessa tabela. Depois que os dados são copiados, as consultas SQL no Amazon Redshift não afetam o DynamoDB de forma alguma. Isso ocorre porque as consultas agem em uma cópia dos dados do DynamoDB, em vez de no DynamoDB em si.

Para poder carregar dados de uma tabela do DynamoDB, você deve primeiro criar uma tabela do Amazon Redshift para servir como o destino dos dados. Lembre-se de que você está copiando

dados de um ambiente NoSQL para um ambiente SQL, e que há determinadas regras em um ambiente que não se aplicam ao outro. Veja algumas das diferenças a considerar:

- Os nomes de tabela do DynamoDB podem conter até 255 caracteres, incluindo os caracteres "." (ponto) e "-" (traço), e diferenciam maiúsculas e minúsculas. Os nomes de tabela do Amazon Redshift são limitados a 127 caracteres, não podem conter pontos ou traços e não diferenciam maiúsculas e minúsculas. Além disso, nomes de tabela não podem entrar em conflito com quaisquer palavras reservadas do Amazon Redshift.
- O DynamoDB não é compatível com o conceito SQL de NULL. Você precisa especificar como o Amazon Redshift interpreta valores de atributo vazios ou em branco no DynamoDB, tratando-os como NULLs ou como campos vazios.
- Os tipos de dados do DynamoDB não correspondem diretamente aos do Amazon Redshift. Você precisa garantir que cada coluna na tabela do Amazon Redshift seja do tipo e do tamanho corretos para acomodar os dados do DynamoDB.

Este é um exemplo do comando COPY do SQL do Amazon Redshift:

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
credentials 'aws_access_key_id=<Your-Access-Key-ID>;aws_secret_access_key=<Your-Secret-
Access-Key>'
readratio 50;
```

Neste exemplo, a tabela de origem no DynamoDB é `my-favorite-movies-table`. A tabela de destino no Amazon Redshift é `favoritemovies`. A cláusula `readratio 50` regula a percentagem do throughput provisionado que é consumida; neste caso, o comando COPY usará não mais que 50% das unidades de capacidade de leitura provisionadas para `my-favorite-movies-table`. É altamente recomendável definir esse índice para um valor menor do que a média de throughput provisionado não utilizado.

Para obter instruções detalhadas sobre como carregar dados do DynamoDB no Amazon Redshift, consulte as seções a seguir no Guia do desenvolvedor de banco de dados do Amazon Redshift:

- Carregar dados de uma tabela do DynamoDB
- O comando COPY
- Exemplos de COPY

Processar dados do DynamoDB com o Apache Hive no Amazon EMR

O Amazon DynamoDB é integrado com o Apache Hive, uma aplicação de data warehousing que pode ser executada no Amazon EMR. O Hive pode ler e gravar dados nas tabelas do DynamoDB, permitindo que você:

- Consulte dados dinâmicos do DynamoDB usando uma linguagem semelhante a SQL (HiveQL).
- Copie dados de uma tabela do DynamoDB para um bucket do Amazon S3 e vice-versa.
- Copie dados de uma tabela do DynamoDB para o Hadoop Distributed File System (HDFS) e vice-versa.
- Execute operações de junção em tabelas do DynamoDB.

Tópicos

- Visão geral
- Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive
- Criar uma tabela externa no Hive
- Processar instruções em HiveQL
- Consultar dados no DynamoDB
- Copiar dados de e para o Amazon DynamoDB
- Ajuste de performance

Visão geral

O Amazon EMR é um serviço que facilita o processamento de grandes quantidades de dados de maneira rápida e econômica. Para usar o Amazon EMR, você deve iniciar um cluster gerenciado de instâncias do Amazon EC2 executando o framework de código aberto do Hadoop. O Hadoop é uma aplicação distribuída que implementa o algoritmo MapReduce, onde uma tarefa é mapeada para vários nós no cluster. Cada nó processa seu trabalho designado em paralelo com outros nós. Finalmente, as saídas são reduzidas em um único nó, gerando o resultado final.

Você pode optar por iniciar seu cluster do Amazon EMR de modo que ele seja persistente ou transitório:

- Um cluster persistente é executado até ser desativado por você. Os clusters persistentes são ideais para análise de dados, armazenamento de dados ou qualquer outro uso interativo.
- Um cluster transitório é executado por tempo suficiente para processar um fluxo de trabalho e, em seguida, é desativado automaticamente. Os clusters transitórios são ideais para tarefas de processamento periódicas, como a execução de scripts.

Para obter informações sobre arquitetura e administração do Amazon EMR, consulte o Guia de gerenciamento do Amazon EMR.

Ao iniciar um cluster do Amazon EMR, você deve especificar o número inicial e o tipo de instâncias do Amazon EC2. Você também pode especificar outras aplicações distribuídas (além do Hadoop em si) que deseja executar no cluster. Esses aplicativos incluem Matiz, Mahout, Pig, Spark e muito mais.

Para obter informações sobre aplicações para Amazon EMR, consulte o Guia de versão do Amazon EMR.

Dependendo da configuração do cluster, é possível ter um ou mais dos seguintes tipos de nó:

- Nó líder: gerencia o cluster, coordenando a distribuição do MapReduce executável e os subconjuntos de dados brutos, o núcleo e os grupos de instâncias de tarefa. Ele também acompanha o status de cada tarefa executada e monitora a integridade dos grupos de instâncias. Há apenas um nó principal em um cluster.
- Nós core - executam tarefas do MapReduce e armazenam dados usando o Hadoop Distributed File System (HDFS).
- Nós de tarefas (opcional) – executam tarefas do MapReduce.

Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive

Neste tutorial, você iniciará um cluster do Amazon EMR e, em seguida, usará o Apache Hive para processar os dados armazenados em uma tabela do DynamoDB.

O Hive é uma aplicação de data warehouse para o Hadoop que permite processar e analisar dados de várias fontes diferentes. O Hive oferece uma linguagem semelhante a SQL, HiveQL, que permite trabalhar com dados armazenados localmente no cluster do Amazon EMR ou em uma fonte de dados externa (como o Amazon DynamoDB).

Para obter mais informações, consulte o Tutorial do Hive.

Tópicos

- Antes de começar
- Etapa 1: criar um par de chaves do Amazon EC2
- Etapa 2: iniciar um cluster do Amazon EMR
- Etapa 3: conectar ao nó líder
- Etapa 4: carregar dados no HDFS
- Etapa 5: copiar dados para o DynamoDB
- Etapa 6: consultar os dados na tabela do DynamoDB
- Etapa 7: (opcional) limpar

Antes de começar

Para este tutorial, você precisará do seguinte:

- Uma conta da AWS. Se você não tiver uma, consulte Como se cadastrar na AWS.
- Um cliente SSH (Secure Shell). É possível usar o cliente SSH para se conectar ao nó líder do cluster do Amazon EMR e executar comandos interativos. Os clientes SSH estão disponíveis por padrão na maioria das instalações de Linux, Unix e Mac OS X. Os usuários do Windows podem fazer download e instalar o cliente PuTTY, que oferece suporte para SSH.

Próxima etapa

Etapa 1: criar um par de chaves do Amazon EC2

Etapa 1: criar um par de chaves do Amazon EC2

Nesta etapa, você criará o par de chaves do Amazon EC2 de que precisa para se conectar a um nó líder do Amazon EMR e executar comandos do Hive.

1. Faça login no AWS Management Console e abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. Escolha uma região (por exemplo, `US West (Oregon)`). Essa deve ser a mesma região em que a tabela do DynamoDB está localizada.
3. No painel de navegação, selecione Key Pairs (Pares de chaves).
4. Escolha Criar par de chaves.

5. Em Key pair name, digite um nome para o seu par de chaves (por exemplo, `mykeypair`) e, em seguida, escolha Create.
6. Baixe o arquivo de chave privada. O nome do arquivo terminará com `.pem` (por exemplo, `mykeypair.pem`). Mantenha esse arquivo de chave privada em um lugar seguro. Você precisará dele para acessar qualquer cluster do Amazon EMR que iniciar com esse par de chaves.

   > **Important**
   >
   > Se você perder o par de chaves, não poderá se conectar ao nó líder do seu cluster do Amazon EMR.

   Para obter informações sobre pares de chaves, consulte Pares de chaves do Amazon EC2 no Guia do usuário do Amazon EC2.

Próxima etapa

Etapa 2: iniciar um cluster do Amazon EMR

Etapa 2: iniciar um cluster do Amazon EMR

Nesta etapa, você irá configurar e iniciar um cluster do Amazon EMR. O Hive e um handler de armazenamento para DynamoDB já estarão instalados no cluster.

1. Abra o console do Amazon EMR em https://console.aws.amazon.com/emr.
2. Selecione Create Cluster (Criar cluster).
3. Na página Create Cluster - Quick Options, faça o seguinte:

   a. Em Cluster name, digite um nome para o seu cluster (por exemplo: `My EMR cluster`).
   b. Em EC2 key pair, escolha o par de chaves que você criou mais cedo.

   Deixe as outras configurações nos valores padrão.
4. Selecione Criar cluster.

Vários minutos serão necessários para o cluster ser iniciado. Você pode usar a página Cluster Details (Detalhes do cluster) no console do Amazon EMR para monitorar seu progresso.

Quando o status mudar para `Waiting`, o cluster estará pronto para uso.

Arquivos de log do cluster e Amazon S3

Um cluster do Amazon EMR gera arquivos de log que contêm informações sobre o status do cluster e as informações de depuração. As configurações padrão de Create Cluster – Quick Options (Criar cluster – Opções rápidas) incluem a configuração do registro em log do Amazon EMR.

O AWS Management Console criará um bucket do Amazon S3, caso ainda não exista um. O nome do bucket é `aws-logs-`*`account-id`*`-`*`region`*, onde *`account-id`* é o número da sua conta da AWS e *`region`* é a região em que você iniciou o cluster (por exemplo, `aws-logs-123456789012-us-west-2`).

> **Note**
>
> Você pode usar o console do Amazon S3 para visualizar os arquivos de log. Para obter mais informações, consulte Visualizar arquivos de log no Guia de gerenciamento do Amazon EMR.

Você pode usar esse bucket para outras finalidades, além do log. Por exemplo, você pode usar o bucket como um local para armazenar um script do Hive ou como um destino ao exportar dados do Amazon DynamoDB para o Amazon S3.

Próxima etapa

Etapa 3: conectar ao nó líder

Etapa 3: conectar ao nó líder

Quando o status do seu cluster do Amazon EMR mudar para `Waiting`, você poderá se conectar ao nó líder usando SSH e executar operações de linha de comando.

1. No console do Amazon EMR, escolha o nome do cluster para visualizar seu status.
2. Na página Cluster Details (Detalhes do cluster), encontre o campo Master public DNS (DNS público mestre). Este é o nome DNS público do nó líder do seu cluster do Amazon EMR.
3. À direita do nome DNS, escolha o link SSH.
4. Siga as instruções em Conectar ao nó líder via SSH.

Dependendo do seu sistema operacional, escolha a guia Windows ou Mac/Linux e siga as instruções para se conectar ao nó líder.

Depois de se conectar ao nó líder usando o SSH ou o PuTTY, você verá um prompt de comando semelhante ao seguinte:

```
[hadoop@ip-192-0-2-0 ~]$
```

Próxima etapa

Etapa 4: carregar dados no HDFS

Etapa 4: carregar dados no HDFS

Nesta etapa, você poderá copiar um arquivo de dados para o Hadoop Distributed File System (HDFS) e, em seguida, criar uma tabela do Hive externa que mapeia para o arquivo de dados.

Baixar os dados de amostra

1. Faça download do arquivo de dados de exemplo (`features.zip`):

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. Extraia o arquivo `features.txt` do arquivo:

   ```
   unzip features.zip
   ```

3. Visualize as primeiras linhas do arquivo `features.txt`:

   ```
   head features.txt
   ```

   O resultado deve ter a seguinte aparência:

   ```
   1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
   875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
   1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
   26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
   1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
   1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
   ```

```
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

O arquivo `features.txt` contém um subconjunto de dados do United States Board on Geographic Names (http://geonames.usgs.gov/domestic/download_data.htm). Os campos em cada linha representam o seguinte:

- ID do recurso (identificador exclusivo)
- Nome
- Classe (lago; floresta; riacho; e assim por diante)
- State
- Latitude (graus)
- Longitude (graus)
- Altura (em pés)

4. No prompt de comando, digite o seguinte comando:

```
hive
```

O prompt de comando muda para: `hive>`

5. Insira a seguinte instrução HiveQL para criar uma tabela nativa do Hive:

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

6. Insira a seguinte instrução HiveQL para carregar a tabela com dados:

```
LOAD DATA
```

```
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. Você agora tem uma tabela nativa do Hive preenchida com os dados do arquivo `features.txt`. Para verificar, insira a seguinte instrução HiveQL:

   ```
   SELECT state_alpha, COUNT(*)
   FROM hive_features
   GROUP BY state_alpha;
   ```

   A saída deve mostrar uma lista de estados e o número de recursos geográficos em cada uma delas.

Próxima etapa

Etapa 5: copiar dados para o DynamoDB

Etapa 5: copiar dados para o DynamoDB

Nesta etapa, você poderá copiar dados na tabela do Hive (`hive_features`) para uma nova tabela no DynamoDB.

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. Selecione Create Table (Criar tabela).
3. Na página Create DynamoDB table, faça o seguinte:

   a. Na tabela, digite **`Features`**.

   b. Para Primary key (Chave primária), no campo Partition key (Chave da partição), digite **`Id`**. Defina o tipo de dados como Number (Número).

      Desmarque Use Default Settings (Usar configurações padrão). Para Provisioned Capacity, digite o seguinte:

      - Unidades de capacidade de leitura—`10`
      - Unidades de capacidade de gravação—`10`

   Escolha Criar.

4. No prompt do Hive, insira a seguinte instrução HiveQL:

```
CREATE EXTERNAL TABLE ddb_features
    (feature_id   BIGINT,
    feature_name  STRING,
    feature_class STRING,
    state_alpha   STRING,
    prim_lat_dec  DOUBLE,
    prim_long_dec DOUBLE,
    elev_in_ft    BIGINT)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
    "dynamodb.table.name" = "Features",

 "dynamodb.column.mapping"="feature_id:Id,feature_name:Name,feature_class:Class,state_alpha
);
```

   Você agora estabeleceu um mapeamento entre o Hive e a tabela Resources no DynamoDB.

5. Insira a seguinte instrução HiveQL para importar dados para o DynamoDB:

```
INSERT OVERWRITE TABLE ddb_features
SELECT
    feature_id,
    feature_name,
    feature_class,
    state_alpha,
    prim_lat_dec,
    prim_long_dec,
    elev_in_ft
FROM hive_features;
```

   O Hive enviará um trabalho do MapReduce que será processado por seu cluster do Amazon EMR. Levará vários minutos para o trabalho ser concluído.

6. Verifique se os dados foram carregados para o DynamoDB:

   a. No painel de navegação do console do DynamoDB, escolha Tables (Tabelas).

   b. Escolha a tabela Recursos e, em seguida, escolha a guia Items para visualizar os dados.

Próxima etapa

Etapa 6: consultar os dados na tabela do DynamoDB

Etapa 6: consultar os dados na tabela do DynamoDB

Nesta etapa, você usará HiveQL para consultar a tabela Resources no DynamoDB. Experimente as seguintes consultas Hive:

1. Todos os tipos de recursos (`feature_class`) em ordem alfabética:

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. Todos os lagos que começam com a letra "M":

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. Estados com pelo menos três recursos com mais de uma milha (1,61 km) (5.280 pés, 1.609 m):

```
SELECT state_alpha, feature_class, COUNT(*)
FROM ddb_features
WHERE elev_in_ft > 5280
GROUP by state_alpha, feature_class
HAVING COUNT(*) >= 3
ORDER BY state_alpha, feature_class;
```

Próxima etapa

Etapa 7: (opcional) limpar

Etapa 7: (opcional) limpar

Agora que concluiu o tutorial, você pode continuar a ler esta seção para saber mais sobre como trabalhar com dados do DynamoDB no Amazon EMR. Você pode optar por manter seu cluster do Amazon EMR em execução ao mesmo tempo em que faz isso.

Se não precisar mais do cluster, termine-o e remova quaisquer recursos associados. Isso ajudará você a não ser cobrado por recursos de que você não precisa.

1. Terminar o cluster do Amazon EMR:
   a. Abra o console do Amazon EMR em https://console.aws.amazon.com/emr.
   b. Escolha o cluster do Amazon EMR, escolha Terminate (Terminar) e, em seguida, confirme.
2. Exclua a tabela Resources no DynamoDB:
   a. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
   b. No painel de navegação, selecione Tables (Tabelas).
   c. Escolha a tabela Resources. No menu Actions, escolha Delete Table.
3. Excluir o bucket do Amazon S3 que contém os arquivos de log do Amazon EMR:
   a. Abra o console do Amazon S3 em https://console.aws.amazon.com/s3/.
   b. Na lista de buckets, escolha `aws-logs-` *`accountID`*`-`*`region`*, onde *`accountID`* é o número da sua conta da AWS e *`region`* é a região em que você iniciou o cluster.
   c. No menu Action, escolha Delete.

Criar uma tabela externa no Hive

No Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive, você criou uma tabela externa do Hive mapeada para uma tabela do DynamoDB. Quando você emitiu instruções HiveQL na tabela externa, as operações de leitura e gravação foram repassadas para a tabela do DynamoDB.

Você pode pensar em uma tabela externa como um ponteiro para uma fonte de dados que é gerenciada e armazenada em outro lugar. Neste caso, a fonte de dados subjacente é uma tabela do DynamoDB. (A tabela já deve existir. Não é possível criar, atualizar ou excluir uma tabela do DynamoDB via Hive.) Você usa a instrução `CREATE EXTERNAL TABLE` para criar a tabela externa. Depois disso, você pode usar HiveQL para trabalhar com dados no DynamoDB, como se esses dados estivessem armazenados localmente no Hive.

> **Note**
>
> Você pode usar as instruções `INSERT` para inserir dados em uma tabela externa e as instruções `SELECT` para selecionar dados nela. No entanto, você não pode usar as instruções `UPDATE` ou `DELETE` para manipular dados na tabela.

Se não precisar mais da tabela externa, remova-a usando a instrução `DROP TABLE`. Neste caso, `DROP TABLE` remove apenas a tabela externa no Hive. Ele não afeta a tabela subjacente do DynamoDB ou qualquer um dos seus dados.

Tópicos

- Sintaxe CREATE EXTERNAL TABLE
- Mapeamentos de tipo de dados

Sintaxe CREATE EXTERNAL TABLE

As considerações a seguir mostram a sintaxe HiveQL para criação de uma tabela externa do Hive mapeada em uma tabela do DynamoDB:

```
CREATE EXTERNAL TABLE hive_table

 (hive_column1_name hive_column1_datatype, hive_column2_name hive_column2_datatype...)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES (
    "dynamodb.table.name" = "dynamodb_table",
    "dynamodb.column.mapping" =
 "hive_column1_name:dynamodb_attribute1_name,hive_column2_name:dynamodb_attribute2_name..."
);
```

A linha 1 é o início da instrução `CREATE EXTERNAL TABLE`, na qual você fornece o nome da tabela do Hive (hive_table) que você deseja criar.

A linha 2 especifica as colunas e os tipos de dados de hive_table. É necessário definir colunas e tipos de dados que correspondem aos atributos da tabela do DynamoDB.

A linha 3 é a cláusula `STORED BY`, onde você especifica uma classe que lida com o gerenciamento de dados entre o Hive e a tabela do DynamoDB. Para o DynamoDB, `STORED BY` deve ser definido como `'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'`.

A linha 4 é o início da cláusula `TBLPROPERTIES`, na qual você define os parâmetros a seguir para `DynamoDBStorageHandler`:

- `dynamodb.table.name`: o nome da tabela do DynamoDB.
- `dynamodb.column.mapping`: pares de nomes de coluna na tabela do Hive e seus atributos correspondentes na tabela do DynamoDB. Cada par tem o formato hive_column_name: dynamodb_attribute_name e os pares são separados por vírgulas.

Observe o seguinte:

- O nome da tabela do Hive não precisa ser o mesmo nome da tabela do DynamoDB.
- Os nomes das colunas da tabela do Hive não precisam ser os mesmos que os da tabela do DynamoDB.
- A tabela especificada por `dynamodb.table.name` deve existir no DynamoDB.
- Para `dynamodb.column.mapping`:
  - Você deve mapear os atributos de esquema de chaves da tabela do DynamoDB. Isso inclui a chave de partição e a chave de classificação (se houver).
  - Não é necessário mapear os atributos que não são chaves da tabela do DynamoDB. No entanto, você não verá os dados desses atributos ao consultar a tabela do Hive.
  - Se os tipos de dados de uma coluna de tabela do Hive e um atributo do DynamoDB forem incompatíveis, você verá `NULL` nessas colunas ao consultar a tabela do Hive.

> **Note**
>
> A instrução `CREATE EXTERNAL TABLE` não executa qualquer validação na cláusula `TBLPROPERTIES`. Os valores fornecidos para `dynamodb.table.name` e `dynamodb.column.mapping` são avaliados apenas pela classe `DynamoDBStorageHandler` quando você tenta acessar a tabela.

Mapeamentos de tipo de dados

A tabela a seguir mostra os tipos de dados do DynamoDB e os tipos de dados compatíveis do Hive:

Tipo de dados do DynamoDB	Tipo de dados do Hive
String	`STRING`
Número	`BIGINT` ou `DOUBLE`
Binário	`BINARY`
String Set	`ARRAY<STRING>`
Number Set	`ARRAY<BIGINT>` ou `ARRAY<DOUBLE>`
Binary Set	`ARRAY<BINARY>`

> **Note**
>
> Os tipos de dados do DynamoDB a seguir não são compatíveis com a classe `DynamoDBStorageHandler`, portanto, eles não podem ser usados com `dynamodb.column.mapping`:
>
> - Mapa
> - Listar
> - Booleano
> - Null
>
> No entanto, se precisar trabalhar com esses tipos de dados, você poderá criar uma única entidade chamada `item` que represente todo o item do DynamoDB como um mapa de strings para chaves e valores no mapa. Para ter mais informações, consulte Copiar dados sem um mapeamento de colunas.

Caso deseje mapear um atributo do DynamoDB do tipo Number, será necessário escolher um tipo apropriado do Hive:

- O tipo do Hive `BIGINT` é para inteiros assinados de 8 bytes. Ele é o mesmo que o tipo de dados `long` em Java.

- O tipo do Hive `DOUBLE` é para números de ponto flutuante de dupla precisão com 8 bits. Ele é o mesmo que o tipo `double` em Java.

Se você tiver dados numéricos armazenados no DynamoDB que tenham uma maior precisão do que o tipo de dados do Hive que você escolheu, então, acessar os dados do DynamoDB poderia causar uma perda de precisão.

Se você exportar dados do tipo Binary do DynamoDB para (Amazon S3) ou HDFS, os dados serão armazenados como uma string codificada por Base64. Caso importe dados do Amazon S3 ou HDFS para o tipo Binary do DynamoDB, você deverá garantir que os dados sejam codificados como uma string Base64.

Processar instruções em HiveQL

O Hive é uma aplicação executada no Hadoop que consiste em um framework orientado a lote para a execução de trabalhos do MapReduce. Quando você emite uma instrução HiveQL, o Hive determina se ele pode retornar os resultados imediatamente ou se ele deve enviar um trabalho do MapReduce.

Por exemplo, considere a tabela ddb_features (de Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive). A consulta do Hive a seguir imprime as abreviações de estado e o número de conferências em cada:

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

O Hive não retorna os resultados imediatamente. Em vez disso, ele submete um trabalho do MapReduce, que é processado pelo framework do Hadoop. O Hive aguardará até que o trabalho seja concluído antes de mostrar os resultados da consulta:

```
AK  2
AL  2
AR  2
AZ  3
CA  7
CO  2
CT  2
ID  1
```

```
KS  1
ME  2
MI  1
MT  3
NC  1
NE  1
NM  1
NY  2
OR  5
PA  1
TN  1
TX  1
UT  4
VA  1
VT  2
WA  2
WY  3
Time taken: 8.753 seconds, Fetched: 25 row(s)
```

Monitorar e cancelar trabalhos

Quando o Hive executa um trabalho do Hadoop, ele imprime a saída desse trabalho. O status da conclusão do trabalho é atualizado à medida que o trabalho avança. Em alguns casos, o status pode não ser atualizado por um longo período. (Isso pode acontecer quando você está consultando uma tabela grande do DynamoDB que tem uma baixa capacidade de leitura provisionada configurada.)

Caso precise cancelar o trabalho antes que ele seja concluído, digite **`Ctrl+C`** a qualquer momento.

Consultar dados no DynamoDB

Os exemplos a seguir mostram algumas formas como você pode usar HiveQL para consultar dados armazenados no DynamoDB.

Esses exemplos referenciam a tabela ddb_features no tutorial (Etapa 5: copiar dados para o DynamoDB).

Tópicos

- Usar funções agregadas
- Usar cláusulas GROUP BY e HAVING
- União de duas tabelas do DynamoDB
- União de tabelas de origens diferentes

Usar funções agregadas

O HiveQL fornece funções internas para resumir valores de dados. Por exemplo, você pode usar a função `MAX` para localizar o maior valor para uma coluna selecionada. O exemplo a seguir retorna a elevação do recurso mais elevado no estado do Colorado.

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

Usar cláusulas GROUP BY e HAVING

Você pode usar a cláusula `GROUP BY` para coletar dados em vários registros. Ela é usada frequentemente com uma função agregada, como `SUM`, `COUNT`, `MIN` ou `MAX`. Você também pode usar a cláusula `HAVING` para descartar os resultados que não atenderem a determinados critérios.

O exemplo a seguir retorna uma lista das elevações mais altas entre os estados que têm mais de cinco recursos na tabela ddb_features.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

União de duas tabelas do DynamoDB

O exemplo a seguir mapeia outra tabela do Hive (east_coast_states) em uma tabela no DynamoDB. A instrução `SELECT` é uma junção entre essas duas tabelas. A junção é calculada no cluster e retornada. A junção não ocorre no DynamoDB.

Considere uma tabela do DynamoDB chamada EastCoastStates que contém os seguintes dados:

```
StateName       StateAbbrev

Maine           ME
New Hampshire   NH
Massachusetts   MA
Rhode Island    RI
Connecticut     CT
New York        NY
```

```
New Jersey      NJ
Delaware        DE
Maryland        MD
Virginia        VA
North Carolina  NC
South Carolina  SC
Georgia         GA
Florida         FL
```

Vamos supor que a tabela esteja disponível como uma tabela externa do Hive chamada east_coast_states:

```
CREATE EXTERNAL TABLE ddb_east_coast_states (state_name STRING, state_alpha STRING)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "EastCoastStates",
"dynamodb.column.mapping" = "state_name:StateName,state_alpha:StateAbbrev");
```

A junção a seguir retorna os estados na Costa Leste dos Estados Unidos que têm pelo menos três recursos:

```
SELECT ecs.state_name, f.feature_class, COUNT(*)
FROM ddb_east_coast_states ecs
JOIN ddb_features f on ecs.state_alpha = f.state_alpha
GROUP BY ecs.state_name, f.feature_class
HAVING COUNT(*) >= 3;
```

União de tabelas de origens diferentes

No exemplo a seguir, s3_east_coast_states é uma tabela do Hive associada a um arquivo CSV armazenado no Amazon S3. A tabela ddb_features é associada aos dados no DynamoDB. O exemplo a seguir junta essas duas tabelas, retornando os recursos geográficas de estados cujos nomes começam com "Nova".

```
create external table s3_east_coast_states (state_name STRING, state_alpha STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';
```

```
SELECT ecs.state_name, f.feature_name, f.feature_class
FROM s3_east_coast_states ecs
JOIN ddb_features f
```

```
ON ecs.state_alpha = f.state_alpha
WHERE ecs.state_name LIKE 'New%';
```

Copiar dados de e para o Amazon DynamoDB

No Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive , você copiou os dados de uma tabela nativa do Hive para uma tabela externa do DynamoDB e, em seguida, consultou a tabela externa do DynamoDB. A tabela é externa porque existe fora do Hive. Mesmo se você descartar a tabela do Hive mapeada a ela, a tabela do DynamoDB não será afetada.

O Hive é uma excelente solução para copiar dados entre tabelas do DynamoDB, buckets do Amazon S3, tabelas nativas do Hive e o Hadoop Distributed File System (HDFS). Esta seção fornece exemplos dessas operações.

Tópicos

- Copiar dados entre o DynamoDB e uma tabela nativa do Hive
- Copiar dados entre o DynamoDB e o Amazon S3
- Copiar dados entre o DynamoDB e o HDFS
- Usando a compactação de dados
- Ler dados de caracteres UTF-8 não imprimíveis

Copiar dados entre o DynamoDB e uma tabela nativa do Hive

Caso tenha dados em uma tabela do DynamoDB, você poderá copiá-los para uma tabela nativa do Hive. Isso gerará um snapshot dos dados, a partir do momento em que você os copiou.

Você pode optar por fazer isso caso precise executar muitas consultas HiveQL, mas não deseja consumir o throughput provisionado do DynamoDB. Como os dados da tabela nativa do Hive são uma cópia dos dados do DynamoDB, e não dados "ao vivo", suas consultas não devem esperar que os dados sejam atualizados.

> **Note**
>
> Os exemplos nesta seção são escritos sob a premissa de que você seguiu as etapas de Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive e tem uma tabela externa chamada ddb_features no DynamoDB.

Example Do DynamoDB para a tabela nativa do Hive

Você pode criar uma tabela nativa do Hive e preenchê-la com dados de ddb_features, desta forma:

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

Em seguida, você pode atualizar os dados a qualquer momento:

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

Nesses exemplos, a subconsulta `SELECT * FROM ddb_features` recuperará todos os dados de ddb_features. Se quiser copiar apenas um subconjunto dos dados, você pode usar uma cláusula `WHERE` na subconsulta.

O exemplo a seguir cria uma tabela nativa do Hive, contendo apenas alguns dos atributos para lagos e picos:

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example Da tabela nativa do Hive para o DynamoDB

Use a instrução HiveQL a seguir para copiar os dados da tabela nativa do Hive para ddb_features:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

Copiar dados entre o DynamoDB e o Amazon S3

Caso tenha dados em uma tabela do DynamoDB, você poderá usar o Hive para copiá-los para um bucket do Amazon S3.

Você poderá fazer, por exemplo, se quiser criar um arquivo de dados na sua tabela do DynamoDB. Por exemplo, suponha que você tem um ambiente de teste no qual precisa trabalhar com um conjunto de dados de teste de linha de base no DynamoDB. Você pode copiar os dados de

referência para um bucket do Amazon S3 e, em seguida, executar os testes. Depois disso, você poderá redefinir o ambiente de teste restaurando os dados de referência do bucket do Amazon S3 para o DynamoDB.

Se você trabalhou no Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive, então já tem um bucket do Amazon S3 que contém seus logs do Amazon EMR. Você pode usar esse bucket para os exemplos desta seção, se souber o caminho raiz do bucket:

1. Abra o console do Amazon EMR em https://console.aws.amazon.com/emr.
2. Para Name, escolha o seu cluster.
3. O URI é listado no Log URI em Configuration Details.
4. Anote o caminho raiz do bucket. A convenção de nomenclatura é:

   `s3://aws-logs-`*`accountID`*`-`*`region`*

   onde *`accountID`* é o ID da sua conta da AWS e região é a região da AWS do bucket.

> **Note**
>
> Para esses exemplos, usaremos um subcaminho no bucket, como neste exemplo: `s3://aws-logs-123456789012-us-west-2/hive-test`

Os procedimentos a seguir são escritos sob a premissa de que você seguiu as etapas do tutorial e tem uma tabela externa chamada ddb_features no DynamoDB.

Tópicos

- Copiar dados usando o formato padrão do Hive
- Copiar dados com um formato especificado pelo usuário
- Copiar dados sem um mapeamento de colunas
- Visualizar os dados no Amazon S3

Copiar dados usando o formato padrão do Hive

Example Do DynamoDB para o Amazon S3

Use uma instrução `INSERT OVERWRITE` para escrever diretamente para no Amazon S3.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

O arquivo de dados no Amazon S3 é semelhante a:

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Cada campo é separado por um caractere SOH (início de cabeçalho, 0x01). No arquivo, SOH aparece como **^A**.

Example Do Amazon S3 para o DynamoDB

1. Crie uma tabela externa apontando para os dados não formatados no Amazon S3.

   ```
   CREATE EXTERNAL TABLE s3_features_unformatted
       (feature_id       BIGINT,
       feature_name      STRING ,
       feature_class     STRING ,
       state_alpha       STRING,
       prim_lat_dec      DOUBLE ,
       prim_long_dec     DOUBLE ,
       elev_in_ft        BIGINT)
   LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
   ```

2. Copie os dados para o DynamoDB.

   ```
   INSERT OVERWRITE TABLE ddb_features
   SELECT * FROM s3_features_unformatted;
   ```

Copiar dados com um formato especificado pelo usuário

Se você deseja especificar seu próprio caractere separador de campos, crie uma tabela externa mapeada no bucket do Amazon S3. Você pode usar essa técnica para a criação de arquivos de dados com valores separados por vírgulas (CSV).

Example Do DynamoDB para o Amazon S3

1. Crie uma tabela externa do Hive mapeada no Amazon S3. Ao fazer isso, certifique-se de que os tipos de dados sejam consistentes com os da tabela externa do DynamoDB.

```
CREATE EXTERNAL TABLE s3_features_csv
    (feature_id       BIGINT,
    feature_name      STRING,
    feature_class     STRING,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE,
    prim_long_dec     DOUBLE,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. Copie os dados do DynamoDB.

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

O arquivo de dados no Amazon S3 é semelhante a:

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example Do Amazon S3 para o DynamoDB

Com uma única instrução HiveQL, você pode preencher a tabela do DynamoDB usando os dados do Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

Copiar dados sem um mapeamento de colunas

Você pode copiar dados do DynamoDB em um formato bruto e gravá-los no Amazon S3 sem especificar quaisquer tipos de dados ou mapeamento da coluna. Você pode usar esse método para criar um arquivamento de dados do DynamoDB e armazená-lo no Amazon S3.

Example Do DynamoDB para o Amazon S3

1. Crie uma tabela externa associada à sua tabela do DynamoDB. (Não há `dynamodb.column.mapping` nesta instrução HiveQL.)

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. Crie outra tabela externa associada ao seu bucket do Amazon S3.

   ```
   CREATE EXTERNAL TABLE s3_features_no_mapping
       (item MAP<STRING, STRING>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   LINES TERMINATED BY '\n'
   LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
   ```

3. Copie os dados do DynamoDB para o Amazon S3.

   ```
   INSERT OVERWRITE TABLE s3_features_no_mapping
   SELECT * FROM ddb_features_no_mapping;
   ```

O arquivo de dados no Amazon S3 é semelhante a:

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
```

```
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Cada campo começa com um caractere STX (início de texto, 0x02) e termina com um caractere ETX (fim de texto, 0x03). No arquivo, o caractere STX é mostrado como **^B** e o caractere ETX é mostrado como **^C**.

Example Do Amazon S3 para o DynamoDB

Com uma única instrução HiveQL, você pode preencher a tabela do DynamoDB usando os dados do Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

Visualizar os dados no Amazon S3

Se você usa SSH para se conectar ao nó líder, poderá usar a AWS Command Line Interface (AWS CLI) para acessar os dados escritos pelo Hive no Amazon S3.

As etapas a seguir são escritas sob a premissa de que você copiou os dados do DynamoDB para o Amazon S3 usando um dos procedimentos desta seção.

1. Se você estiver no momento no prompt de comando do Hive, saia do prompt de comando do Linux.

   ```
   hive> exit;
   ```

2. Liste o conteúdo do diretório hive-test em seu bucket do Amazon S3. (Esse é o local para o qual o Hive copiou os dados do DynamoDB.)

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   A resposta deve ter a seguinte aparência:

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

   O nome do arquivo (000000_0) é gerado pelo sistema.

3. (Opcional) Você pode copiar o arquivo de dados do Amazon S3 para o sistema de arquivos local no nó líder. Depois de fazer isso, você pode usar os utilitários de linha de comando padrão do Linux para trabalhar com os dados no arquivo.

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

   A resposta deve ter a seguinte aparência:

   ```
   download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
   to ./000000_0
   ```

   > **Note**
   >
   > O sistema de arquivos local no nó líder tem capacidade limitada. Não use este comando com arquivos maiores do que o espaço disponível no sistema de arquivos local.

Copiar dados entre o DynamoDB e o HDFS

Caso tenha dados em uma tabela do DynamoDB, você poderá usar o Hive para copiar os dados para o Hadoop Distributed File System (HDFS).

Você poderá fazer isso se estiver executando um trabalho do MapReduce que exige dados do DynamoDB. Se você copiar os dados do DynamoDB para o HDFS, o Hadoop poderá processá-los, usando todos os nós disponíveis no cluster do Amazon EMR em paralelo. Quando o trabalho do MapReduce for concluído, você poderá gravar os resultados do HDFS no DDB.

Nos exemplos a seguir, o Hive irá ler e gravar no diretório HDFS a seguir: `/user/hadoop/hive-test`

> **Note**
>
> Os exemplos nesta seção são escritos sob a premissa de que você seguiu as etapas de Tutorial: Como trabalhar com o Amazon DynamoDB e o Apache Hive e tem uma tabela externa chamada ddb_features no DynamoDB.

Tópicos

- Copiar dados usando o formato padrão do Hive

- Copiar dados com um formato especificado pelo usuário
- Copiar dados sem um mapeamento de colunas
- Acessar dados no HDFS

Copiar dados usando o formato padrão do Hive

Example Do DynamoDB para o HDFS

Use uma instrução `INSERT OVERWRITE` para gravar diretamente para o HDFS.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

O arquivo de dados do HDFS é semelhante a:

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Cada campo é separado por um caractere SOH (início de cabeçalho, 0x01). No arquivo, SOH aparece como **^A**.

Example Do HDFS para o DynamoDB

1. Crie uma tabela externa mapeada para os dados não formatados no HDFS.

   ```
   CREATE EXTERNAL TABLE hdfs_features_unformatted
       (feature_id       BIGINT,
       feature_name      STRING ,
       feature_class     STRING ,
       state_alpha       STRING,
       prim_lat_dec      DOUBLE ,
       prim_long_dec     DOUBLE ,
       elev_in_ft        BIGINT)
   LOCATION 'hdfs:///user/hadoop/hive-test';
   ```

2. Copie os dados para o DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

Copiar dados com um formato especificado pelo usuário

Caso deseje usar um caractere separador de campos diferente, você poderá criar uma tabela externa mapeada no diretório do HDFS. Você pode usar essa técnica para a criação de arquivos de dados com valores separados por vírgulas (CSV).

Example Do DynamoDB para o HDFS

1. Crie uma tabela externa do Hive mapeada no HDFS. Ao fazer isso, certifique-se de que os tipos de dados sejam consistentes com os da tabela externa do DynamoDB.

```
CREATE EXTERNAL TABLE hdfs_features_csv
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. Copie os dados do DynamoDB.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

O arquivo de dados do HDFS é semelhante a:

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example Do HDFS para o DynamoDB

Com uma única instrução HiveQL, você pode preencher a tabela do DynamoDB usando os dados do HDFS:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

Copiar dados sem um mapeamento de colunas

Você pode copiar dados do DynamoDB em um formato bruto e gravá-los no HDFS sem especificar quaisquer tipos de dados ou mapeamento de coluna. Você pode usar esse método para criar um arquivo com dados do DynamoDB e armazená-lo no HDFS.

> **Note**
>
> Se a sua tabela do DynamoDB contém atributos do tipo Map, List, Boolean ou Null, essa é a única forma possível de usar o Hive para copiar dados do DynamoDB para o HDFS.

Example Do DynamoDB para o HDFS

1. Crie uma tabela externa associada à sua tabela do DynamoDB. (Não há `dynamodb.column.mapping` nesta instrução HiveQL.)

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. Crie outra tabela externa associada ao seu diretório HDFS.

   ```
   CREATE EXTERNAL TABLE hdfs_features_no_mapping
       (item MAP<STRING, STRING>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   LINES TERMINATED BY '\n'
   LOCATION 'hdfs:///user/hadoop/hive-test';
   ```

3. Copie os dados do DynamoDB para o HDFS.

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

O arquivo de dados do HDFS é semelhante a:

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Cada campo começa com um caractere STX (início de texto, 0x02) e termina com um caractere ETX (fim de texto, 0x03). No arquivo, o caractere STX é mostrado como **^B** e o caractere ETX é mostrado como **^C**.

Example Do HDFS para o DynamoDB

Com uma única instrução HiveQL, você pode preencher a tabela do DynamoDB usando os dados do HDFS:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

Acessar dados no HDFS

O HDFS é um sistema de arquivos distribuído, acessível para todos os nós do cluster do Amazon EMR. Caso use o SSH para se conectar ao nó líder, você poderá usar as ferramentas de linha de comando para acessar os dados gravados pelo Hive no HDFS.

O HDFS não é a mesma coisa que o sistema de arquivos local no nó líder. Você não pode trabalhar com arquivos e diretórios no HDFS usando os comandos padrão do Linux (como `cat`, `cp`, `mv` ou `rm`). Em vez disso, você executa essas tarefas usando o comando `hadoop fs`.

As etapas a seguir são escritas sob a premissa de que você copiou os dados do DynamoDB para o HDFS usando um dos procedimentos desta seção.

1. Se você estiver no momento no prompt de comando do Hive, saia do prompt de comando do Linux.

   ```
   hive> exit;
   ```

2. Liste o conteúdo do diretório /user/hadoop/hive-test no HDFS. (Esse é o local para o qual o Hive copiou os dados do DynamoDB.)

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   A resposta deve ter a seguinte aparência:

   ```
   Found 1 items
   -rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
   ```

   O nome do arquivo (000000_0) é gerado pelo sistema.

3. Visualize o conteúdo do arquivo :

   ```
   hadoop fs -cat /user/hadoop/hive-test/000000_0
   ```

   > **Note**
   >
   > Neste exemplo, o arquivo é relativamente pequeno (aproximadamente 29 KB). Tenha cuidado ao usar este comando com arquivos que são muito grandes ou contêm caracteres não imprimíveis.

4. (Opcional) Você pode copiar o arquivo de dados do HDFS para o sistema de arquivos local no nó líder. Depois de fazer isso, você pode usar os utilitários de linha de comando padrão do Linux para trabalhar com os dados no arquivo.

   ```
   hadoop fs -get /user/hadoop/hive-test/000000_0
   ```

   Este comando não substituirá o arquivo.

> **Note**
>
> O sistema de arquivos local no nó líder tem capacidade limitada. Não use este comando com arquivos maiores do que o espaço disponível no sistema de arquivos local.

Usando a compactação de dados

Ao usar o Hive para copiar dados entre diferentes fontes de dados, você pode solicitar a compactação de dados durante o processo. O Hive fornece vários codecs de compactação. É possível escolher um durante a sessão do Hive. Quando você faz isso, os dados são compactados no formato especificado.

O exemplo a seguir compacta os dados usando o algoritmo Lempel-Ziv-Oberhumer (LZO).

```
SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
FROM hiveTableName;
```

O arquivo resultante no Amazon S3 terá um nome gerado pelo sistema com `.lzo` no final (por exemplo, `8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`).

Os codecs de compactação disponíveis são:

- `org.apache.hadoop.io.compress.GzipCodec`
- `org.apache.hadoop.io.compress.DefaultCodec`
- `com.hadoop.compression.lzo.LzoCodec`
- `com.hadoop.compression.lzo.LzopCodec`
- `org.apache.hadoop.io.compress.BZip2Codec`
- `org.apache.hadoop.io.compress.SnappyCodec`

Ler dados de caracteres UTF-8 não imprimíveis

Para ler e gravar dados de caracteres UTF-8 não imprimíveis, é possível usar a cláusula `STORED AS SEQUENCEFILE` ao criar uma tabela do Hive. Um SequenceFile é um formato de arquivo binário do Hadoop. É necessário usar o Hadoop para ler esse arquivo. O exemplo a seguir mostra como exportar dados do DynamoDB para o Amazon S3. Você pode usar essa funcionalidade para lidar com caracteres de codificação UTF-8 não imprimíveis.

```
CREATE EXTERNAL TABLE s3_export(a_col string, b_col bigint, c_col array<string>)
STORED AS SEQUENCEFILE
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3_export SELECT *
FROM hiveTableName;
```

Ajuste de performance

Ao criar uma tabela externa do Hive mapeada em uma tabela do DynamoDB, você não consome nenhuma capacidade de leitura ou gravação do DynamoDB. No entanto, a atividade de leitura e gravação na tabela do Hive (como `INSERT` ou `SELECT`) se traduz diretamente em operações de leitura e gravação na tabela subjacente do DynamoDB.

O Apache Hive no Amazon EMR implementa sua próxima lógica para balancear a carga de E/S na tabela do DynamoDB e procura minimizar a possibilidade de exceder o throughput provisionado da tabela. No final de cada consulta do Hive, o Amazon EMR retorna as métricas de tempo de execução, incluindo o número de vezes em que o throughput provisionado foi excedido. Você pode usar essas informações, com as métricas do CloudWatch em sua tabela do DynamoDB, para melhorar a performance em solicitações subsequentes.

O console do Amazon EMR fornece ferramentas de monitoramento básicas para seu cluster. Para obter mais informações, consulte Visualizar e monitorar um cluster no Guia de gerenciamento do Amazon EMR.

É possível monitorar o cluster e trabalhos do Hadoop usando ferramentas baseadas na Web, como Matiz, Ganglia e a interface da web do Hadoop. Para obter mais informações, consulte Visualizar interfaces da Web hospedadas em clusters do Amazon EMR no Guia de gerenciamento do Amazon EMR.

Esta seção descreve as etapas que você pode executar para ajustar a performance das operações do Hive em tabelas externas do DynamoDB.

Tópicos

- Throughput provisionado de DynamoDB
- Ajustar mapeadores
- Tópicos adicionais

Throughput provisionado de DynamoDB

Quando você emite instruções HiveQL na tabela externa do DynamoDB, a classe `DynamoDBStorageHandler` faz as solicitações de API de baixo nível do DynamoDB apropriadas, as quais consomem throughput provisionado. Se não houver capacidade de leitura ou gravação suficiente na tabela do DynamoDB, a solicitação será limitada, o que resultará em baixa performance do HiveQL. Por esse motivo, você deve garantir que a tabela tenha capacidade de throughput suficiente.

Por exemplo, suponha que você provisionou 100 unidades de capacidade de leitura para sua tabela do DynamoDB. Isso permitirá que você leia 409.600 bytes por segundo (100 × tamanho da unidade de capacidade de leitura de 4 KB). Agora, suponha que a tabela contenha 20 GB de dados (21.474.836.480 bytes) e você deseja usar a instrução `SELECT` para selecionar todos os dados usando HiveQL. Você pode estimar quanto tempo a consulta levará para ser executada desta forma:

21.474.836.480/409.600 = 52.429 segundos = 14,56 horas

Nesse cenário, a tabela do DynamoDB é um gargalo. Ela não ajudará a adicionar mais nós do Amazon EMR, pois o throughput do Hive é limitado a apenas 409.600 bytes por segundo. A única maneira de reduzir o tempo necessário para a instrução `SELECT` é aumentar a capacidade de leitura provisionada da tabela do DynamoDB.

Você pode executar um cálculo semelhante para estimar a quantidade de tempo necessária para a carga de grandes volumes de dados em uma tabela externa do Hive mapeada para uma tabela do DynamoDB. Determine o número total de unidades de capacidade de gravação necessárias por item (menos de 1 KB = 1, 1-2 KB = 2 etc.) e multiplique pelo número de itens a serem carregados. Isso fornecerá o número de unidades de capacidade de gravação exigidas. Divida esse número pelo número de unidades de capacidade de gravação que são alocadas por segundo. Isso resultará no número de segundos necessários para carregar a tabela.

Você deve monitorar regularmente as métricas do CloudWatch da sua tabela. Para obter uma visão geral rápida no console do DynamoDB, escolha sua tabela e, em seguida, selecione a guia Metrics

(Métricas). Aqui, você pode visualizar as unidades de capacidade de leitura e gravação consumidas e as solicitações que foram limitadas.

Capacidade de leitura

O Amazon EMR gerencia a carga de solicitação em sua tabela do DynamoDB de acordo com as configurações de throughput provisionado. No entanto, se você perceber um grande número de mensagens `ProvisionedThroughputExceeded` na saída do trabalho, poderá ajustar a taxa de leitura padrão. Para fazer isso, você pode modificar a variável de configuração `dynamodb.throughput.read.percent`. Você pode usar o comando `SET` para definir essa variável no prompt de comando do Hive.

```
SET dynamodb.throughput.read.percent=1.0;
```

Esta variável persiste apenas durante a sessão atual do Hive. Se você sair do Hive e retornar mais tarde, `dynamodb.throughput.read.percent` retornará para o seu valor padrão.

O valor de `dynamodb.throughput.read.percent` pode variar entre `0.1` e `1.5`. `0.5` representa a taxa de leitura padrão, o que significa que o Hive tentará consumir metade da capacidade de leitura da tabela. Se você aumentar o valor acima de `0.5`, o Hive aumentará a taxa de solicitação; diminuir o valor abaixo de `0.5` reduz a taxa de solicitação de leitura. (A taxa de leitura real poderá variar, dependendo de fatores como se há uma distribuição de chaves uniformes na tabela do DynamoDB.)

Se você notar que o Hive está esgotando frequentemente a capacidade de leitura provisionada da tabela ou se o controle de utilização de solicitações de leitura estiver sendo aplicado com muita frequência, tente reduzir `dynamodb.throughput.read.percent` para menos de `0.5`. Se você tiver capacidade de leitura suficiente e quiser operações HiveQL mais responsivas, defina o valor acima de `0.5`.

Capacidade de gravação

O Amazon EMR gerencia a carga de solicitação em sua tabela do DynamoDB de acordo com as configurações de throughput provisionado. No entanto, se você perceber um grande número de mensagens `ProvisionedThroughputExceeded` na saída do trabalho, poderá ajustar a taxa de gravação padrão. Para fazer isso, você pode modificar a variável de configuração `dynamodb.throughput.write.percent`. Você pode usar o comando `SET` para definir essa variável no prompt de comando do Hive.

```
SET dynamodb.throughput.write.percent=1.0;
```

Esta variável persiste apenas durante a sessão atual do Hive. Se você sair do Hive e retornar mais tarde, `dynamodb.throughput.write.percent` retornará para o seu valor padrão.

O valor de `dynamodb.throughput.write.percent` pode variar entre `0.1` e `1.5`. `0.5` representa a taxa de gravação padrão, o que significa que o Hive tentará consumir metade da capacidade de gravação da tabela. Se você aumentar o valor acima de `0.5`, o Hive aumentará a taxa de solicitação. Diminuir o valor abaixo de `0.5` reduz a taxa de solicitação de gravação. (A taxa de gravação real poderá variar dependendo de fatores como a existência de uma distribuição de chaves uniformes na tabela do DynamoDB.)

Se você notar que o Hive está esgotando frequentemente a capacidade de leitura provisionada da tabela ou se o controle de utilização de solicitações de gravação estiver sendo aplicado com muita frequência, tente reduzir `dynamodb.throughput.write.percent` para menos de `0.5`. Se você tiver capacidade suficiente na tabela e quiser operações HiveQL mais responsivas, defina o valor acima de `0.5`.

Ao gravar dados no DynamoDB usando o Hive, certifique-se de que o número de unidades de capacidade de gravação seja maior do que o número de mapeadores no cluster. Por exemplo, considere um cluster do Amazon EMR que consiste em 10 nós m1.xlarge. O tipo de nó m1.xlarge fornece 8 tarefas de mapeador. Assim, o cluster teria um total de 80 mapeadores (10 × 8). Se a sua tabela do DynamoDB tiver menos de 80 unidades de capacidade de gravação, uma operação de gravação do Hive poderia consumir todo o throughput de gravação dessa tabela.

Para determinar o número de mapeadores dos tipos de nó do Amazon EMR, consulte Configuração da tarefa no Guia do desenvolvedor do Amazon EMR.

Para obter mais informações sobre mapeadores, consulte Ajustar mapeadores.

Ajustar mapeadores

Quando o Hive inicia um trabalho do Hadoop, esse trabalho é processado por uma ou mais tarefas do mapeador. Supondo que a tabela do DynamoDB tenha capacidade de throughput suficiente, é possível modificar o número de mapeadores no cluster, o que melhora potencialmente a performance.

> **Note**
>
> O número de tarefas do mapeador usadas em um trabalho do Hadoop é influenciado pelas divisões de entrada, onde o Hadoop subdivide os dados em blocos lógicos. Se o Hadoop não

realizar divisões de entrada suficientes, as operações de gravação poderão não ser capazes de consumir todo o throughput de gravação disponível na tabela do DynamoDB.

Aumentar o número de mapeadores

Cada mapeador em um Amazon EMR tem uma taxa máxima de leitura de 1 MiB por segundo. O número de mapeadores em um cluster depende do tamanho dos nós no cluster. (Para obter informações sobre tamanhos de nó e o nome de mapeadores por nó, consulte Configuração da tarefa no Guia do desenvolvedor do Amazon EMR.)

Se a tabela do DynamoDB possuir ampla capacidade de throughput para leituras, você poderá tentar aumentar o número de mapeadores executando um dos procedimentos a seguir:

- Aumentar o tamanho dos nós no cluster. Por exemplo, se o cluster estiver usando nós m1.large (três mapeadores por nó), você pode tentar atualizar para os nós m1.xlarge (oito mapeadores por nó).
- Aumentar o número de nós no cluster. Por exemplo, caso possua um cluster de três nós m1.xlarge, você terá um total de 24 mapeadores disponíveis. Se você dobrar o tamanho do cluster, com o mesmo tipo de nó, terá 48 mapeadores.

É possível usar o AWS Management Console para gerenciar o tamanho ou o número de nós em seu cluster. (Você pode precisar reiniciar o cluster para que essas alterações entrem em vigor.)

Outra forma de aumentar o número de mapeadores é modificar o parâmetro de configuração do Hadoop `mapred.tasktracker.map.tasks.maximum`. (Este é um parâmetro do Hadoop, e não um parâmetro do Hive. Não é possível modificá-lo interativamente a partir do prompt de comando.). Caso aumente o valor de `mapred.tasktracker.map.tasks.maximum`, você poderá aumentar o número de mapeadores sem aumentar o tamanho e o número de nós. No entanto, é possível que os nós do cluster fiquem sem memória se você definir o valor muito alto.

Você define o valor para `mapred.tasktracker.map.tasks.maximum` como uma ação de bootstrap ao iniciar o primeiro cluster do Amazon EMR. Para obter mais informações, consulte (Opcional) Criar ações de bootstrap para instalar software adicional no Guia de gerenciamento do Amazon EMR.

Diminuir o número de mapeadores

Se você usar a instrução SELECT para selecionar dados de uma tabela externa do Hive mapeada no DynamoDB, o trabalho do Hadoop poderá usar quantas tarefas forem necessárias, até o número máximo de mapeadores no cluster. Nesse cenário, é possível que uma consulta do Hive de longa duração possa consumir toda a capacidade de leitura provisionada da tabela do DynamoDB, afetando negativamente outros usuários.

Você pode usar o parâmetro `dynamodb.max.map.tasks` para definir um limite superior para mapear tarefas:

```
SET dynamodb.max.map.tasks=1
```

Esse valor deve ser igual ou maior que 1. Quando o Hive processa sua consulta, o trabalho do Hadoop resultante não usa mais do que `dynamodb.max.map.tasks` ao ler a tabela do DynamoDB.

Tópicos adicionais

A seguir estão algumas maneiras de ajustar as aplicações que usam o Hive para acessar o DynamoDB.

Período de novas tentativas

Por padrão, o Hive executará novamente um trabalho do Hadoop se ele não tiver retornado quaisquer resultados do DynamoDB em até dois minutos. Você pode ajustar esse intervalo, modificando o parâmetro `dynamodb.retry.duration`:

```
SET dynamodb.retry.duration=2;
```

O valor deve ser um número inteiro diferente de zero, representando o número de minutos no intervalo de repetição. O padrão de `dynamodb.retry.duration` é 2 (minutos).

Solicitações de dados em paralelo

Várias solicitações de dados, seja de mais de um usuário ou de mais de um aplicativo, para uma única tabela pode esgotar o throughput provisionado de leitura e diminuir o desempenho.

Duração do processo

A consistência dos dados no DynamoDB depende da ordem de operações de leitura e gravação em cada nó. Embora uma consulta do Hive esteja em andamento, outra aplicação pode carregar

novos dados para a tabela do DynamoDB ou modificar ou excluir dados existentes. Nesse caso, os resultados da consulta do Hive podem não refletir as alterações feitas nos dados enquanto a consulta estava em execução.

Tempo de solicitação

Programa consultas do Hive que acessam uma tabela do Dynamo quando há menor demanda na tabela do DynamoDB melhora a performance. Por exemplo, se a maioria dos usuários da sua aplicação vive em São Francisco, talvez você opte por exportar os dados diariamente às 4h (PST), ou seja, quando a maioria dos usuários está dormindo e não está atualizando registros em seu banco de dados do DynamoDB.

Integração do DynamoDB ao Amazon S3

Os recursos de importação e exportação do Amazon DynamoDB oferecem uma maneira simples e eficiente de mover dados entre tabelas do Amazon S3 e do DynamoDB sem escrever nenhum código.

Os recursos de importação e exportação do DynamoDB ajudam você a mover, transformar e copiar contas de tabela do DynamoDB. É possível importar de suas origens do S3, exportar os dados da tabela do DynamoDB para o Amazon S3 e usar serviços da AWS como o Athena, o Amazon SageMaker AI e o AWS Lake Formation para analisar os dados e extrair insights acionáveis. Você também pode importar dados diretamente para novas tabelas do DynamoDB com o objetivo de criar novas aplicações em escala com performance de milissegundos de um dígito, facilitar o compartilhamento de dados entre tabelas e contas e simplificar os planos de recuperação de desastres e continuidade de negócios.

Tópicos

- Importação de dados do Amazon S3 para o DynamoDB: como funciona
- Como funciona a exportação de dados do DynamoDB para o Amazon S3

Importação de dados do Amazon S3 para o DynamoDB: como funciona

Para importar dados para o DynamoDB, eles devem estar em um bucket do Amazon S3 no formato CSV, DynamoDB JSON ou Amazon Ion. Os dados podem ser compactados no formato ZSTD ou GZIP, ou podem ser importados diretamente em formato não compactado. Os dados de origem podem ser um único ou vários objetos do Amazon S3 que usam o mesmo prefixo.

Os dados serão importados para uma nova tabela do DynamoDB, que será criada quando você iniciar a solicitação de importação. Você pode criar essa tabela com índices secundários e, em seguida, consultar e atualizar os dados em todos os índices primários e secundários assim que a importação for concluída. Você também pode adicionar uma réplica de tabela global após a conclusão da importação.

> **Note**
>
> Durante o processo de importação do Amazon S3, o DynamoDB cria uma tabela de destino para a qual será realizada a importação. No momento, a importação para tabelas existentes não é compatível com esse recurso.

Como a importação do Amazon S3 não consome capacidade de gravação na nova tabela, você não precisa provisionar nenhuma capacidade extra para importar dados para o DynamoDB. O preço da importação de dados baseia-se no tamanho não compactado dos dados de origem no Amazon S3, que são processados como resultado da importação. Os itens que são processados, mas não são carregados na tabela devido à formatação ou a outras inconsistências nos dados de origem, também são cobrados como parte do processo de importação. Para obter detalhes de preço, consulte Preço do Amazon DynamoDB.

Você poderá importar dados de um bucket do Amazon S3 pertencente a uma conta diferente caso tenha as permissões corretas para ler desse bucket específico. A nova tabela também pode estar em uma região diferente do bucket do Amazon S3 de origem. Para obter mais informações, consulte Amazon Simple Storage Service setup and permissions (Configuração e permissões do Amazon Simple Storage Service).

Os tempos de importação estão diretamente relacionados às características de seus dados no Amazon S3. Isso inclui tamanho dos dados, formato de dados, esquema de compactação, uniformidade de distribuição de dados, número de objetos do Amazon S3 e outras variáveis relacionadas. Mais especificamente, conjuntos de dados com chaves uniformemente distribuídas serão mais rápidos de importar do que conjuntos de dados distorcidos. Por exemplo, se a chave do índice secundário estiver usando o mês do ano para particionamento e todos os dados forem do mês de dezembro, a importação desses dados poderá demorar muito mais.

Espera-se que os atributos associados às chaves sejam exclusivos na tabela-base. Se alguma chave não for exclusiva, a importação substituirá os itens associados até o momento em que restar apenas a última substituição. Por exemplo, se a chave primária for o mês e vários itens forem

definidos para o mês de setembro, cada novo item substituirá os itens escritos anteriormente e apenas um item com o mês da (chave primária) definido como setembro permanecerá. Nesses casos, o número de itens processados na descrição da tabela de importação não corresponderá ao número de itens na tabela de destino.

O AWS CloudTrail registra todas as ações de console e API para importação de tabela. Para obter mais informações, consulte Registrar em log as operações do DynamoDB usando o AWS CloudTrail.

O vídeo a seguir é uma introdução à importação direta do Amazon S3 para o DynamoDB.

Importar do Amazon S3

Tópicos

- Solicitação de importação de tabela no DynamoDB
- Formatos de importação do Amazon S3 para o DynamoDB
- Importar cotas e validação de formato
- Práticas recomendadas de importação do Amazon S3 para o DynamoDB

Solicitação de importação de tabela no DynamoDB

A importação do DynamoDB permite que você importe dados de um bucket do Amazon S3 para uma nova tabela do DynamoDB. Você pode solicitar uma importação de tabela do console do DynamoDB, da CLI, do CloudFormation ou da API do DynamoDB.

Se desejar usar a AWS CLI, você precisará configurá-la primeiro. Para obter mais informações, consulte Acessar o DynamoDB.

> **Note**
>
> - O recurso de importação de tabela interage com vários serviços da AWS, como o Amazon S3 e o CloudWatch. Antes de iniciar uma importação, verifique se o usuário ou o perfil que invoca as APIs de importação tem permissões para todos os serviços e recursos dos quais o recurso depende.
> - Não modifique os objetos do Amazon S3 enquanto a importação estiver em andamento, pois isso poderá fazer com que a operação falhe ou seja cancelada.

Para obter mais informações sobre erros e solução de problemas, consulte Importar cotas e validação de formato.

Tópicos

- Configurar permissões do IAM
- Solicitar uma importação usando o AWS Management Console
- Obter detalhes sobre importações anteriores na AWS Management Console
- Solicitar uma importação usando a AWS CLI
- Obter detalhes sobre importações anteriores na AWS CLI

Configurar permissões do IAM

Você pode importar dados de qualquer bucket do Amazon S3 que você tenha permissão para ler. O bucket de destino não precisa estar na mesma região ou ter o mesmo proprietário que a tabela de origem. Seu AWS Identity and Access Management (IAM) deve incluir as ações relevantes no bucket do Amazon S3 de origem e as permissões necessárias do CloudWatch para fornecer informações de depuração. Um exemplo de política é mostrado abaixo.

JSON

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Sid": "AllowDynamoDBImportAction",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ImportTable",
        "dynamodb:DescribeImport"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table*"
    },
    {
      "Sid": "AllowS3Access",
      "Effect": "Allow",
```

```
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket/*",
        "arn:aws:s3:::your-bucket"
      ]
    },
    {
      "Sid": "AllowCloudwatchAccess",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group/aws-dynamodb/*"
    },
    {
      "Sid": "AllowDynamoDBListImports",
      "Effect": "Allow",
      "Action": "dynamodb:ListImports",
      "Resource": "*"
    }
  ]
}
```

Permissões do Amazon S3

Ao iniciar uma importação em uma origem de bucket do Amazon S3 que pertence a outra conta, verifique se o perfil ou o usuário tem acesso aos objetos do Amazon S3. Você pode verificar isso executando um comando `GetObject` do Amazon S3 e usando as credenciais. Ao usar a API, o parâmetro do proprietário do bucket do Amazon S3 é definido por padrão como o ID da conta do usuário atual. Para importações entre contas, verifique se esse parâmetro está preenchido corretamente com o ID da conta do proprietário do bucket. O código a seguir é um exemplo de política de bucket do Amazon S3 na conta de origem.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {"Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS Key Management Service

Ao criar a tabela para importação, se você selecionar uma chave de criptografia em repouso que não pertença ao DynamoDB, forneça as permissões do AWS KMS necessárias para operar uma tabela do DynamoDB criptografada com chaves gerenciadas pelo cliente. Para obter mais informações, consulte Autorizar o uso da chave do AWS KMS. Se os objetos do Amazon S3 estiverem criptografados com criptografia do lado do servidor do KMS (SSE-KMS), verifique se o perfil ou o usuário que está iniciando a importação tem acesso para descriptografar usando a chave do AWS KMS. Esse recurso não oferece suporte a objetos do Amazon S3 criptografados com chaves de criptografia fornecidas pelo cliente (SSE-C).

Permissões do CloudWatch

A função ou o usuário que está iniciando a importação precisará de permissões de criação e de gerenciamento para o grupo de logs e os fluxos de log associados à importação.

Solicitar uma importação usando o AWS Management Console

O exemplo a seguir demonstra como usar o console do DynamoDB para importar dados existentes para uma nova tabela chamada `MusicCollection`.

Como solicitar uma importação de tabela

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, escolha Import from S3 (Importar do S3).
3. Na página exibida, selecione Import from S3 (Importar do S#).
4. Escolha Import from S3 (Importar do S3).
5. Em URL do S3 de origem, insira o URL de origem do Amazon S3.

   Se você for proprietário do bucket de origem, escolha Procurar S3 para procurá-lo. Como alternativa, insira o URL do bucket no seguinte formato: `s3://bucket/prefix`. O `prefix` é um prefixo das chaves do Amazon S3. É o nome do objeto do Amazon S3 a ser importado ou o prefixo de chave compartilhado por todos os objetos do Amazon S3 que você deseja importar.

   > **Note**
   >
   > Não é possível usar o mesmo prefixo das solicitações de exportação do DynamoDB. O recurso de exportação cria uma estrutura de pastas e arquivos de manifesto para todas as exportações. Se você usar o mesmo caminho do Amazon S3, isso gerará um erro. Em vez disso, aponte a importação para a pasta, a qual contém dados dessa exportação específica. O formato do caminho correto nesse caso será `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/Data/`, em que `XXXXXXXX-XXXXXX` é o ID de exportação. É possível encontrar o ID no ARN de exportação, o qual tem o seguinte formato: `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>`. Por exemplo, .`arn:aws:dynamodb:`*`us-east-1`*`:`*`123456789012`*`:table/`*`ProductCatalog`*`/export/`*`01234567890123-a1b2c3d4`*
6. Especifique se você é o S3 bucket owner (Proprietário do bucket do S3). Se o bucket de origem pertencer a outra conta, selecione Uma conta diferente da AWS. Depois, insira o ID da conta do proprietário do bucket.
7. Em Import file compression (Importar compactação de arquivo), selecione No compression (Sem compactação), GZIP ou ZSTD conforme apropriado.
8. Selecione o formato de arquivo de importação pertinente. As opções são DynamoDB JSON, Amazon Ion ou CSV. Se você selecionar CSV, terá duas opções adicionais: CSV header (Cabeçalho CSV) e CSV delimiter character (Caractere delimitador CSV).

Em CSV header (Cabeçalho CSV), escolha se o cabeçalho será retirado da primeira linha do arquivo ou será personalizado. Se você selecionar Customize your headers (Personalize seus cabeçalhos), poderá especificar os valores de cabeçalho com os quais deseja importar. Os cabeçalhos CSV especificados por esse método diferenciam maiúsculas de minúsculas e devem conter as chaves da tabela de destino.

Em CSV delimiter character (Caractere delimitador CSV), você define o caracteres que separará os itens. A vírgula é selecionada por padrão. Se você selecionar Custom delimiter character (Caractere delimitador personalizado), o delimitador deverá corresponder ao padrão regex: `[,;:|\t ]`.

9. Selecione a opção Next (Próximo) e as opções para a nova tabela que será criada para armazenar os dados.

> **Note**
>
> A chave primária e a chave de classificação devem corresponder aos atributos no arquivo. Do contrário, a importação falhará. Os atributos diferenciam maiúsculas de minúsculas.

10. Selecione novamente Next (Próximo) para revisar suas opções de importação e, em seguida, clique em Import (Importar) para iniciar a tarefa de importação. Você verá primeiro sua nova tabela, listada em “Tables” (Tabelas) com o status “Creating” (Criando). No momento, a tabela não está acessível.
11. Quando a importação for concluída, o status será exibido como “Active” (Ativo) e você poderá começar a usar a tabela.

Obter detalhes sobre importações anteriores na AWS Management Console

Para encontrar informações sobre tarefas de importação executadas anteriormente, clique em Import from S3 (Importar do S3) na barra lateral de navegação e selecione a guia Imports (Importações). O painel de importação contém uma lista de todas as importações que você criou nos últimos 90 dias. Se você selecionar o ARN de uma tarefa listada na guia Imports (Importações), recuperará informações sobre essa importação, incluindo as configurações avançadas escolhidas.

Solicitar uma importação usando a AWS CLI

O exemplo a seguir importa dados formatados em CSV de um bucket do S3 chamado bucket com um prefixo de prefixo para uma nova tabela chamada target-table.

```
aws dynamodb import-table --s3-bucket-source S3Bucket=bucket,S3KeyPrefix=prefix \
            --input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"hk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"hk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}' \
            --input-format-options '{"Csv": {"HeaderList": ["hk", "title", "artist",
 "year_of_release"], "Delimiter": ";"}}'
```

> **Note**
>
> Se você optar por criptografar sua importação usando uma chave protegida pelo AWS Key Management Service (AWS KMS), a chave deverá estar na mesma região que o bucket do Amazon S3 de destino.

Obter detalhes sobre importações anteriores na AWS CLI

Você pode encontrar informações sobre tarefas de importação executadas no passado usando o comando `list-imports`. Esse comando retorna uma lista de todas as importações que você criou nos últimos 90 dias. Observe que, embora os metadados da tarefa de importação expirem após 90 dias e os trabalhos mais antigos não sejam mais encontrados nessa lista, o DynamoDB não exclui nenhum dos objetos no bucket do Amazon S3.

```
aws dynamodb list-imports
```

Para recuperar informações detalhadas sobre uma tarefa de importação específica, incluindo quaisquer configurações avançadas, use o comando `describe-import`.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

Formatos de importação do Amazon S3 para o DynamoDB

O DynamoDB pode importar dados em três formatos: CSV, DynamoDB JSON e Amazon Ion.

Tópicos

- CSV
- DynamoDB JSON
- Amazon Ion

CSV

Um arquivo no formato CSV consiste em vários itens delimitados por novas linhas. Por padrão, o DynamoDB interpreta a primeira linha de um arquivo de importação como o cabeçalho e espera que as colunas sejam delimitadas por vírgulas. Você também pode definir cabeçalhos que serão aplicados, desde que correspondam ao número de colunas no arquivo. Se você definir cabeçalhos explicitamente, a primeira linha do arquivo será importada como valores.

> **Note**
>
> Ao importar de arquivos CSV, todas as colunas, exceto o intervalo de hash e as chaves da tabela base e dos índices secundários, são importadas como strings do DynamoDB.

Escape em aspas duplas

Todos os caracteres de aspas duplas presentes no arquivo CSV devem receber escape. Se não receberem escape, como no exemplo a seguir, a importação falhará:

```
id,value
"123",Women's Full "Length" Dress
```

Essa mesma importação será bem-sucedida se as aspas receberem escape com dois conjuntos de aspas duplas:

```
id,value
"""123""","Women's Full ""Length"" Dress"
```

Depois que o texto receber escape e for importado corretamente, ele aparecerá como no arquivo CSV original:

```
id,value
```

```
"123",Women's Full "Length" Dress
```

DynamoDB JSON

Um arquivo no formato JSON do DynamoDB pode consistir em vários objetos de item. Cada objeto está no formato JSON organizado padrão do DynamoDB, e as novas linhas são usadas como delimitadores de itens. Um recurso adicional é usar exportações de um ponto anterior no tempo como uma fonte de importação como padrão.

> **Note**
>
> Novas linhas são usadas como delimitadores de itens para um arquivo no formato JSON do DynamoDB e não devem ser usadas em um objeto de item.

```
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "333-3333333333"}, "Id": {"N": "103"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 103 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "444-444444444"}, "Id": {"N": "104"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 104 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "555-5555555555"}, "Id": {"N": "105"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 105 Title"}}}
```

Amazon Ion

O Amazon Ion é um formato de serialização de dados hierárquico, autodescritivo e altamente compatível com vários tipos de declaração e criado para lidar com os desafios de desacoplamento, desenvolvimento rápido e eficiência enfrentados todos os dias na criação de arquiteturas orientadas a serviços em grande escala.

Quando você importa dados no formato Ion, os tipos de dados do Ion são mapeados para os tipos de dados do DynamoDB na nova tabela do DynamoDB.

S. Não.	Conversão de tipos de dados Ion em DynamoDB	B
1	`Ion Data Type`	`DynamoDB Representation`
2	`string`	`String (s)`
3	`bool`	`Boolean (BOOL)`
4	`decimal`	`Number (N)`
5	`blob`	`Binary (B)`
6	`list (with type annotation $dynamodb_SS, $dynamodb_NS, or $dynamodb_BS)`	`Set (SS, NS, BS)`
7	`list`	`List`
8	`struct`	`Map`

Em um arquivo do Ion, os itens são delimitados por novas linhas. Cada linha começa com um marcador de versão do Ion, seguido por um item no formato Ion.

> **Note**
>
> No exemplo a seguir, formatamos os itens de um arquivo em formato Ion em várias linhas para melhorar a legibilidade.

```
$ion_1_0
[
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"333-3333333333",
```

```
      Id:103.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 103 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"444-4444444444",
      Id:104.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 104 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"555-5555555555",
      Id:105.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 105 Title"
    }
  }
]
```

Importar cotas e validação de formato

Importar cotas

A importação do Amazon S3 para o DynamoDB pode comportar até 50 trabalhos de importação simultâneos com um tamanho total de objeto de origem de importação de 15 TB por vez nas regiões us-east-1, us-west-2 e eu-west-1. Em todas as outras regiões, podem ser feitas até 50 tarefas de

importação simultâneas com um tamanho total de 1 TB. Cada trabalho de importação pode ter até 50 mil objetos do Amazon S3 em todas as regiões. Essas cotas padrão são aplicadas a todas as contas. Se você achar que precisa revisar essas cotas, entre em contato com a equipe da sua conta. Isso será considerado caso a caso. Para obter mais detalhes sobre os limites do DynamoDB, consulte Service Quotas.

Erros de validação

Durante o processo de importação, o DynamoDB pode encontrar erros ao analisar os dados. Para cada erro, o DynamoDB emite um log do CloudWatch e mantém uma contagem do número total de erros encontrados. Se o próprio objeto do Amazon S3 for malformado ou se o respectivo conteúdo não puder formar um item do DynamoDB, poderemos ignorar o processamento da parte restante do objeto.

> **Note**
>
> Se a fonte de dados do Amazon S3 tiver vários itens que compartilham a mesma chave, os itens serão substituídos até restar um. Por isso, pode parecer que um item foi importado e os outros foram ignorados. Os itens duplicados serão substituídos em ordem aleatória, não serão contados como erros e não serão emitidos para os logs do CloudWatch.
> Quando a importação estiver concluída, será possível ver a contagem total de itens importados, de erros e de itens processados. Para solucionar problemas adicionais, também é possível verificar o tamanho total dos itens importados e dos dados processados.

Há três categorias de erros de importação: erros de validação de API, erros de validação de dados e erros de configuração.

Erros de validação de API

Erros de validação de API são erros em nível de item da API de sincronização. As causas comuns são problemas de permissões, falta de parâmetros obrigatórios e falhas de validação de parâmetro. Os detalhes sobre o motivo de falha da chamada de API encontram-se nas exceções lançadas pela solicitação `ImportTable`.

Erros de validação de dados

Erros de validação de dados podem ocorrer em nível de item ou de arquivo. Durante a importação, os itens são validados com base nas regras do DynamoDB antes de serem importados para a

tabela de destino. Quando há falha na validação de um item e ele não é importado, o trabalho de importação o ignora e prossegue para o próximo item. No final do trabalho, o status de importação é definido como FAILED (Com falha) com um FailureCode, ItemValidationError e o FailureMessage “Some of the items failed validation checks and were not imported. Please check CloudWatch error logs for more details” (Alguns dos itens apresentaram erro nas verificações de validação e não foram importados. Verifique os logs de erro do CloudWatch para obter mais detalhes).

As causas comuns de erros de validação de dados incluem objetos não analisáveis, objetos no formato incorreto (a entrada especifica DYNAMODB_JSON, mas o objeto não está nesse formato) e incompatibilidade de esquema com as chaves de tabela de origem especificadas.

Erros de configuração

Erros de configuração geralmente são erros de fluxo de trabalho decorrentes de validação de permissão. O fluxo de trabalho de importação verifica algumas permissões depois de aceitar a solicitação. Se houver problemas ao chamar qualquer uma das dependências necessárias, como o Amazon S3 ou o CloudWatch, o processo marcará o status de importação como FAILED (Com falha). O `failureCode` e a `failureMessage` indicam o motivo da falha. Quando aplicável, a mensagem de falha também contém o ID da solicitação, que você pode usar para investigar o motivo da falha no CloudTrail.

Erros comuns de configuração incluem ter o URL errado para o bucket do Amazon S3 e não ter permissão para acessar o bucket do Amazon S3, o CloudWatch Logs e as chaves do AWS KMS usadas para descriptografar o objeto do Amazon S3. Para obter mais informações, consulte Usar e chaves de dados.

Validar objetos de origem do Amazon S3

Para validar objetos de origem do S3, execute as etapas a seguir.

1. Validar o formato dos dados e o tipo de compactação

   - Verifique se todos os objetos correspondentes do Amazon S3 sob o prefixo especificado têm o mesmo formato (DYNAMODB_JSON, DYNAMODB_ION, CSV)
   - Verifique se todos os objetos correspondentes do Amazon S3 sob o prefixo especificado estão compactados da mesma maneira (GZIP, ZSTD, NONE)

> **Note**
>
> Os objetos do Amazon S3 não precisam ter a extensão correspondente (.csv/.json/.ion/.gz/.zstd etc.), pois o formato de entrada especificado na chamada ImportTable tem precedência.

2. Valide se os dados de importação estão em conformidade com o esquema de tabela desejado

   - Verifique se cada item nos dados de origem tem a chave primária. A chave de classificação é opcional para importações.
   - Verifique se o tipo de atributo associado à chave primária e a qualquer chave de classificação corresponde ao tipo de atributo na tabela e no esquema GSI, conforme especificado nos parâmetros de criação de tabela.

Solução de problemas

Logs do CloudWatch

Para trabalhos de importação que apresentam falha, mensagens de erro detalhadas são publicadas nos logs do CloudWatch. Para acessar esses logs, primeiro recupere o ImportArn da saída e describe-import usando este comando:

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

Resultado do exemplo:

```
aws dynamodb describe-import --import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311"
{
    "ImportTableDescription": {
        "ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658528578619-c4d4e311",
        "ImportStatus": "FAILED",
        "TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
        "TableId": "7b7ecc22-302f-4039-8ea9-8e7c3eb2bcb8",
        "ClientToken": "30f8891c-e478-47f4-af4a-67a5c3b595e3",
        "S3BucketSource": {
```

```
            "S3BucketOwner": "ACCOUNT",
            "S3Bucket": "my-import-source",
            "S3KeyPrefix": "import-test"
        },
        "ErrorCount": 1,
        "CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-
dynamodb/imports:*",
        "InputFormat": "CSV",
        "InputCompressionType": "NONE",
        "TableCreationParameters": {
            "TableName": "target-table",
            "AttributeDefinitions": [
                {
                    "AttributeName": "pk",
                    "AttributeType": "S"
                }
            ],
            "KeySchema": [
                {
                    "AttributeName": "pk",
                    "KeyType": "HASH"
                }
            ],
            "BillingMode": "PAY_PER_REQUEST"
        },
        "StartTime": 1658528578.619,
        "EndTime": 1658528750.628,
        "ProcessedSizeBytes": 70,
        "ProcessedItemCount": 1,
        "ImportedItemCount": 0,
        "FailureCode": "ItemValidationError",
        "FailureMessage": "Some of the items failed validation checks and were not
 imported. Please check CloudWatch error logs for more details."
    }
}
```

Recupere o grupo de logs e o ID de importação da resposta acima e use-os para recuperar os logs de erro. O ID de importação é o último elemento do caminho do campo `ImportArn`. O nome do grupo de logs é `/aws-dynamodb/imports`. O nome do fluxo de logs de erro é `import-id/error`. Para este exemplo, seria `01658528578619-c4d4e311/error`.

Ausência da chave pk no item

Se o objeto S3 de origem não contiver a chave primária fornecida como parâmetro, a importação apresentará falha. Por exemplo, quando você define a chave primária para a importação como o nome da coluna “pk”.

```
aws dynamodb import-table –s3-bucket-source S3Bucket=my-import-
source,S3KeyPrefix=import-test.csv \
            –input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"pk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"pk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}'
```

A coluna “pk” está existe no objeto de origem `import-test.csv` que tem o seguinte conteúdo:

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

Essa importação apresentará falha devido a um erro de validação do item porque a chave primária não está presente na fonte de dados.

Exemplo de log de erros do CloudWatch:

```
aws logs get-log-events –log-group-name /aws-dynamodb/imports –log-stream-name
 01658528578619-c4d4e311/error
{
"events": [
{
"timestamp": 1658528745319,
"message": "{\"itemS3Pointer\":{\"bucket\":\"my-import-source\",\"key\":
\"import-test.csv\",\"itemIndex\":0},\"importArn\":\"arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311\",\"errorMessages
\":[\"One or more parameter values were invalid: Missing the key pk in the item\"]}",
"ingestionTime": 1658528745414
}
],
"nextForwardToken": "f/36986426953797707963335499204463414460239026137054642176/s",
"nextBackwardToken": "b/36986426953797707963335499204463414460239026137054642176/s"
}
```

Esse log de erro indica “One or more parameter values were invalid: Missing the key pk in the item” (Um ou mais valores de parâmetro estavam inválidos: o item não contém a chave pk). Como

esse trabalho de importação apresentou falha, a tabela “target-table” agora existe e está vazia porque nenhum item foi importado. O primeiro item foi processado e o objeto apresentou falha na validação do item.

Para corrigir o problema, primeiro exclua a “target-table” se ela não for mais necessária. Em seguida, use um nome de coluna de chave primária existente no objeto de origem ou atualize os dados de origem para:

```
pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

A tabela de destino existe

Quando você inicia um trabalho de importação e recebe a seguinte resposta:

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

Para corrigir esse erro, você precisará escolher um nome de tabela que ainda não exista e tentar importar novamente.

O bucket especificado não existe

Se o bucket de origem não existir, a importação apresentará falha e registrará os detalhes da mensagem de erro no CloudWatch.

Exemplo de descrição de importação:

```
aws dynamodb –endpoint-url $ENDPOINT describe-import –import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658530687105-e6035287"
{
"ImportTableDescription": {
"ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658530687105-e6035287",
"ImportStatus": "FAILED",
"TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
"TableId": "e1215a82-b8d1-45a8-b2e2-14b9dd8eb99c",
"ClientToken": "3048e16a-069b-47a6-9dfb-9c259fd2fb6f",
"S3BucketSource": {
"S3BucketOwner": "531234567890",
"S3Bucket": "BUCKET_DOES_NOT_EXIST",
"S3KeyPrefix": "import-test"
```

```
},
"ErrorCount": 0,
"CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-dynamodb/
imports:*",
"InputFormat": "CSV",
"InputCompressionType": "NONE",
"TableCreationParameters": {
"TableName": "target-table",
"AttributeDefinitions": [
{
"AttributeName": "pk",
"AttributeType": "S"
}
],
"KeySchema": [
{
"AttributeName": "pk",
"KeyType": "HASH"
}
],
"BillingMode": "PAY_PER_REQUEST"
},
"StartTime": 1658530687.105,
"EndTime": 1658530701.873,
"ProcessedSizeBytes": 0,
"ProcessedItemCount": 0,
"ImportedItemCount": 0,
"FailureCode": "S3NoSuchBucket",
"FailureMessage": "The specified bucket does not exist (Service: Amazon S3; Status
 Code: 404; Error Code: NoSuchBucket; Request ID: Q4W6QYYFDWY6WAKH; S3 Extended Request
 ID: ObqSlLeIMJpQqHLRX2C5Sy7n+8g6iGPwy7ixg7eEeTuEkg/+chU/JF+RbliWytMlkUlUcuCLTrI=;
 Proxy: null)"
}
}
```


O `FailureCode` é `S3NoSuchBucket`, e `FailureMessag` contém detalhes como o ID da solicitação e o serviço que gerou o erro. Como o erro foi detectado antes de os dados serem importados para a tabela, uma nova tabela do DynamoDB não é criada. Em alguns casos, quando esses erros são encontrados após o início da importação de dados, a tabela com dados parcialmente importados é mantida.

Para corrigir esse erro, verifique se o bucket do Amazon S3 de origem existe e reinicie o processo de importação.

Práticas recomendadas de importação do Amazon S3 para o DynamoDB

Veja a seguir as práticas recomendadas para importar dados do Amazon S3 para o DynamoDB.

Observar o limite de 50 mil objetos do S3

Cada trabalho de importação comporta no máximo 50 mil objetos do S3. Se o conjunto de dados tiver mais de 50 mil objetos, consolide-os em objetos maiores.

Evite objetos do S3 excessivamente grandes

Os objetos do S3 são importados em paralelo. Ter vários objetos do S3 de tamanho médio permite a execução em paralelo sem sobrecarga excessiva. Para itens com menos de 1 KB, coloque 4 milhões de itens em cada objeto do S3. Se você tiver um tamanho médio de item maior, coloque proporcionalmente menos itens em cada objeto do S3.

Randomize dados classificados

Se um objeto do S3 mantiver os dados em ordem de classificação, ele poderá criar uma partição em funcionamento contínuo. Essa é uma situação em que uma partição recebe toda a atividade, depois a próxima partição e assim por diante. Os dados em ordem de classificação são definidos como itens em sequência no objeto do S3 que serão gravados na mesma partição de destino durante a importação. Uma situação comum em que os dados estão em ordem de classificação é um arquivo CSV em que os itens são classificados por chave de partição para que itens repetidos compartilhem a mesma chave de partição.

Para evitar uma partição em funcionamento contínuo, recomendamos que você randomize a ordem nesses casos. Isso pode melhorar a performance ao espalhar as operações de gravação. Para ter mais informações, consulte Distribuir a atividade de gravação de maneira eficiente durante o upload de dados no DynamoDB.

Compacte os dados para manter o tamanho total do objeto do S3 abaixo do limite regional

No processo de importação do S3, há um limite na soma do tamanho total dos dados do objeto do S3 a serem importados. O limite é de 15 TB nas regiões us-east-1, us-west-2 e eu-west-1 e de 1 TB em todas as outras regiões. O limite é baseado nos tamanhos brutos dos objetos do S3.

A compactação permite que mais dados brutos se encaixem dentro do limite. Se a compactação por si só não for suficiente para ajustar a importação dentro do limite, você também poderá entrar em contato com o AWS Premium Support para aumentar a cota.

Esteja ciente de como o tamanho do item afeta a performance

Se o tamanho médio do item for muito pequeno (abaixo de 200 bytes), o processo de importação poderá demorar um pouco mais do que para itens maiores.

Considere a possibilidade de importar sem nenhum índice secundário global

A duração de uma tarefa de importação pode depender da presença de um ou vários índices secundários globais (GSIs). Se você planeja estabelecer índices com chaves de partição com baixa cardinalidade, poderá ver uma importação mais rápida se adiar a criação do índice até a conclusão da tarefa de importação (em vez de incluí-los no trabalho de importação).

> **Note**
>
> A criação de um GSI durante a importação não resulta em cobranças de gravação (a criação de um GSI após a importação resultaria).

Como funciona a exportação de dados do DynamoDB para o Amazon S3

A exportação do DynamoDB para o S3 é uma solução totalmente gerenciada para exportar dados do DynamoDB para um bucket do Amazon S3 em grande escala. Usando a exportação do DynamoDB para o S3, é possível exportar dados de uma tabela do Amazon DynamoDB para um bucket do Amazon S3 a qualquer momento dentro da janela de recuperação para um ponto no tempo (PITR). Você precisa habilitar a PITR na tabela para usar a funcionalidade de exportação. Esse recurso permite que você realize analytics e consultas complexas nos dados usando outros serviços da AWS, como Athena, AWS Glue, IA do Amazon SageMaker, Amazon EMR e AWS Lake Formation.

A exportação do DynamoDB para o S3 permite que você exporte dados completos e incrementais da tabela do DynamoDB. As exportações são assíncronas, não consomem unidade de capacidade de leitura (RCUs) e não têm impacto na performance e na disponibilidade da tabela. Os formatos de arquivo de exportação aceitos são os formatos DynamoDB JSON e Amazon Ion. Também é possível exportar dados de tabela para um bucket do S3 pertencente a outra conta da AWS e para uma região da AWS diferente. Seus dados são sempre criptografados de ponta a ponta.

As exportações completas do DynamoDB são cobradas com base no tamanho da tabela do DynamoDB (dados da tabela e índices secundários locais) no momento em que a exportação é feita. As exportações incrementais do DynamoDB são cobradas com base no tamanho dos dados processados nos backups contínuos durante o período que está sendo exportado. A exportação incremental tem um custo mínimo de 10 MB. Cobranças adicionais se aplicam ao armazenamento de

dados exportados no Amazon S3 e às solicitações PUT feitas no bucket do Amazon S3. Para obter mais informações sobre essas cobranças, consulte Preço do Amazon DynamoDB e Definição de preço do Amazon S3.

Para obter detalhes sobre cotas de serviço, consulte Exportação de tabela para o Amazon S3.

Tópicos

- Solicitação de uma exportação de tabela no DynamoDB
- Formato de saída de exportação da tabela do DynamoDB

Solicitação de uma exportação de tabela no DynamoDB

As exportações de tabelas do DynamoDB permitem exportar dados de tabela para um bucket do Amazon S3, o que possibilita que você execute analytics e consultas complexas nos dados usando outros serviços da AWS, como Athena, AWS Glue, IA do Amazon SageMaker, Amazon EMR e AWS Lake Formation. Você pode solicitar uma exportação de tabela usando o AWS Management Console, a AWS CLI ou a API do DynamoDB.

> **Note**
>
> Buckets do Amazon S3 pagos pelo solicitante não são aceitos.

O DynamoDB comporta exportação completa e exportação incremental:

- Nas exportações completas, é possível exportar um snapshot completo da tabela para o bucket do Amazon S3 em qualquer momento na janela de recuperação para um ponto no tempo (PITR).
- Nas exportações incrementais, você pode exportar para o bucket do Amazon S3 dados da tabela do DynamoDB que foram alterados, atualizados ou excluídos entre um espaço de tempo especificado, dentro da janela de PITR.

Tópicos

- Pré-requisitos
- Solicitação de uma exportação usando o AWS Management Console
- Obtenção de detalhes sobre exportações anteriores no AWS Management Console
- Solicitação de uma exportação usando o AWS CLI

- Obtenção de detalhes sobre exportações anteriores no AWS CLI
- Solicitar uma exportação usando o AWS SDK
- Obter detalhes sobre exportações anteriores no AWS SDK

Pré-requisitos

Habilitar a PITR

Para usar o recurso de exportação para o S3, você precisará habilitar a PITR na tabela. Para ter detalhes sobre como habilitar a PITR, consulte Habilitar a recuperação para um ponto no tempo. Se você solicitar uma exportação para uma tabela que não tenha a PITR habilitada, a solicitação falhará com uma mensagem de exceção: “Ocorreu um erro (PointInTimeRecoveryUnavailableException) ao chamar a operação `ExportTableToPointInTime`: a recuperação para um ponto no tempo não é ativada para ‘my-dynamodb-table’ da tabela”. Só é possível solicitar e exportar de um ponto no tempo que esteja dentro do `RecoveryPeriodInDays` da PITR configurada.

Configurar permissões do S3

É possível exportar os dados da tabela para qualquer bucket do Amazon S3 em que se tenha permissão para gravar. O bucket de destino não precisa estar na mesma região da AWS nem ter o mesmo proprietário que a tabela de origem. Sua política do AWS Identity and Access Management (IAM) precisa permitir que você realize ações do S3 (`s3:AbortMultipartUpload`, `s3:PutObject` `es3:PutObjectAcl`) e a ação de exportação do DynamoDB (`dynamodb:ExportTableToPointInTime`). Veja um exemplo de uma política que concederá permissões ao usuário para realizar exportações para um bucket do S3.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "AllowS3BucketWrites",
```

```
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Se você precisar gravar em um bucket do Amazon S3 que esteja em outra conta ou não tiver permissões para gravar, o proprietário do bucket do Amazon S3 precisará adicionar uma política de bucket para permitir que você exporte do DynamoDB para esse bucket. Veja um exemplo de política no bucket do Amazon S3 de destino.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Revogar essas permissões enquanto uma exportação estiver em andamento resultará em arquivos parciais.

> **Note**
>
> Se a tabela ou o bucket para o qual você está exportando estiver criptografado com chaves gerenciadas pelo cliente, as políticas de chave do KMS deverão conceder ao DynamoDB permissão para usá-lo. Essa permissão é concedida por meio do usuário/perfil do IAM que aciona o trabalho de exportação. Para obter mais informações sobre criptografia, inclusive práticas recomendadas, consulte "Como o DynamoDB usa o AWS KMS" e "Usar uma chave do KMS personalizada".

Solicitação de uma exportação usando o AWS Management Console

O exemplo a seguir demonstra como usar o console do DynamoDB para restaurar uma tabela existente chamada `MusicCollection`.

> **Note**
>
> Esse procedimento supõe que você habilitou a recuperação em um ponto anterior no tempo. Para habilitá-la para a tabela `MusicCollection`, na guia Overview (Visão geral) da tabela, na seção Table details (Detalhes da tabela), selecione Enable (Habilitar) para Point-in-time recovery (Recuperação em um ponto anterior no tempo).

Para solicitar uma exportação de tabela

1. Faça login no AWS Management Console e abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/.
2. No painel de navegação, no lado esquerdo do console, escolha Export to S3 (Exportar para o S3).
3. Selecione o botão Exportar para o S3.
4. Escolha uma tabela de origem e um bucket do S3 como destino. Se o bucket de destino pertencer à sua conta, você poderá usar o botão Browse S3 (Procurar no S3) para encontrá-lo. Caso contrário, insira o URL do bucket usando o `s3://bucketname/prefix format.`. O **`prefix`** é uma pasta opcional para ajudar a manter o bucket de destino organizado.
5. Escolha Exportação completa ou Exportação incremental. A Exportação completa gera o snapshot completo da tabela tal como ela estava no momento especificado. A Exportação incremental gera as alterações feitas na tabela durante o período de exportação especificado.

A saída é compactada de forma que contenha apenas o estado final do item do período de exportação. O item só aparecerá uma vez na exportação, mesmo que tenha várias atualizações no mesmo período de exportação.

Full export

1. Selecione o momento do qual você deseja exportar o snapshot completo da tabela. Isso pode ocorrer em qualquer momento dentro da janela de PITR. Você também pode selecionar Hora atual para exportar o snapshot mais recente.
2. Com relação ao Formato de arquivo exportado, escolha entre DynamoDB JSON e Amazon Ion. Por padrão, sua tabela será exportada no formato JSON do DynamoDB a partir da hora restaurável mais recente na janela de recuperação em um ponto anterior no tempo e criptografada usando uma chave do Amazon S3 (SSE-S3). Você pode alterar essas configurações de exportação, se necessário.

> **Note**
>
> Se você optar por criptografar a exportação usando uma chave protegida pelo AWS Key Management Service (AWS KMS), a chave deverá estar na mesma região que o bucket do S3 de destino.

Incremental export

1. Selecione o Período de exportação em que você deseja que os dados incrementais sejam exportados. Escolha uma hora de início na janela de PITR. A duração do período de exportação deve ser de pelo menos 15 minutos e não superior a 24 horas. A hora de início do período de exportação é inclusiva e a hora de término é excludente.
2. Escolha entre o Modo absoluto ou o Modo relativo.
   a. O Modo absoluto exportará dados incrementais de acordo com o período especificado.
   b. O modo relativo exportará dados incrementais de acordo com um período de exportação que é relativo ao tempo de envio do trabalho de exportação.
3. Com relação ao Formato de arquivo exportado, escolha entre DynamoDB JSON e Amazon Ion. Por padrão, sua tabela será exportada no formato JSON do DynamoDB a partir da hora restaurável mais recente na janela de recuperação em um ponto anterior

no tempo e criptografada usando uma chave do Amazon S3 (SSE-S3). Você pode alterar essas configurações de exportação, se necessário.

> **Note**
>
> Se você optar por criptografar a exportação usando uma chave protegida pelo AWS Key Management Service (AWS KMS), a chave deverá estar na mesma região que o bucket do S3 de destino.

4. Em Exportar tipo de visualização, selecione Imagens novas e antigas ou Somente novas imagens. Imagem nova fornece o estado mais recente do item Imagem antiga fornece o estado do item imediatamente antes da “data e hora de início” especificadas A configuração padrão é Imagens novas e antigas. Para obter mais informações sobre imagens novas e imagens antigas, consulte Saída de exportação Incremental.

6. Escolha Exportar para começar.

Os dados exportados não são consistentes do ponto de vista transacional. As operações de transação podem ser divididas entre duas saídas de exportação. Um subconjunto de itens pode ser modificado por uma operação de transação refletido na exportação, enquanto outro subconjunto de modificações na mesma transação não é refletido na mesma solicitação de exportação. No entanto, as exportações acabam sendo consistentes. Se uma transação for interrompida durante uma exportação, você terá a transação restante na próxima exportação contígua, sem duplicações. Os períodos usados para exportações são baseados em um relógio interno do sistema e podem variar em um minuto em relação ao relógio local da aplicação.

Obtenção de detalhes sobre exportações anteriores no AWS Management Console

É possível encontrar informações sobre tarefas de exportação realizadas no passado selecionando a seção Exportações para o S3 na barra de navegação lateral. Essa seção contém uma lista de todas as exportações que você criou nos últimos 90 dias. Selecione o ARN de uma tarefa listada na guia Exportações para recuperar informações sobre essa exportação, incluindo quaisquer configurações avançadas escolhidas. Observe que, embora os metadados da tarefa de exportação expirem após 90 dias e os trabalhos mais antigos não sejam mais encontrados nesta lista, os objetos no bucket do S3 permanecerão disponíveis enquanto suas políticas de bucket permitirem. O DynamoDB jamais exclui os objetos que ele cria no bucket do S3 durante uma exportação.

Solicitação de uma exportação usando o AWS CLI

O exemplo a seguir mostra como usar a AWS CLI para exportar uma tabela chamada `MusicCollection` para um bucket do S3 chamado `ddb-export-musiccollection`.

> **Note**
>
> Esse procedimento supõe que você habilitou a recuperação em um ponto anterior no tempo. Para habilitar esse recurso para a tabela `MusicCollection`, execute o seguinte comando.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Full export

O comando a seguir exporta o `MusicCollection` para um bucket do S3 chamado `ddb-export-musiccollection-9012345678` com um prefixo de `2020-Nov`. Os dados da tabela serão exportados no formato JSON do DynamoDB a partir da hora restaurável mais recente na janela de recuperação em um ponto anterior no tempo e serão criptografados usando uma chave do Amazon S3 (SSE-S3).

> **Note**
>
> Se solicitar uma exportação de tabela entre contas, inclua a opção `--s3-bucket-owner`.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
  --s3-bucket ddb-export-musiccollection-9012345678 \
  --s3-prefix 2020-Nov \
  --export-format DYNAMODB_JSON \
  --export-time 1604632434 \
  --s3-bucket-owner 9012345678 \
  --s3-sse-algorithm AES256
```

Incremental export

O comando a seguir executa uma exportação incremental fornecendo um novo `--export-type` e `--incremental-export-specification`. Substitua seus próprios valores por qualquer coisa em itálico. Os horários são especificados como segundos desde o epoch.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
  --s3-bucket BUCKET --s3-prefix PREFIX \
  --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
  --export-type INCREMENTAL_EXPORT
```

> **Note**
>
> Se você optar por criptografar a exportação usando uma chave protegida pelo AWS Key Management Service (AWS KMS), a chave deverá estar na mesma região que o bucket do S3 de destino.

Obtenção de detalhes sobre exportações anteriores no AWS CLI

Você pode encontrar informações sobre solicitações de exportação que executou no passado usando o comando `list-exports`. Esse comando retorna uma lista de todas as exportações que você criou nos últimos 90 dias. Observe que, embora os metadados da tarefa de exportação expirem após 90 dias e os trabalhos mais antigos não sejam mais retornados pelo `list-exports` comando, os objetos em seu bucket do S3 permanecerão disponíveis enquanto suas políticas de bucket permitirem. O DynamoDB jamais exclui os objetos que ele cria no bucket do S3 durante uma exportação.

As exportações permanecem no status `PENDING` até serem concluídas com sucesso ou falha. Se forem bem-sucedidas, o status mudará para `COMPLETED`. Se falharem, o status mudará para `FAILED` com uma `failure_message` e um `failure_reason`.

No exemplo a seguir, usamos o parâmetro opcional `table-arn` para listar apenas as exportações de uma tabela específica.

```
aws dynamodb list-exports \
```

```
    --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

Para recuperar informações detalhadas sobre uma tarefa de exportação específica, incluindo quaisquer configurações avançadas, use o comando `describe-export`.

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

Solicitar uma exportação usando o AWS SDK

Use esses trechos de código para solicitar uma exportação de tabela usando o AWS SDK de sua escolha.

Python

Exportação completa

```
import boto3
from datetime import datetime

# remove endpoint_url for real use
client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/export_table_to_point_in_time.html
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Exportação incremental

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')
```

```
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    IncrementalExportSpecification={
      'ExportFromTime': datetime(2023, 9, 20, 12, 0, 0),
      'ExportToTime': datetime(2023, 9, 20, 13, 0, 0),
      'ExportViewType': 'NEW_AND_OLD_IMAGES'
    },
    ExportType='INCREMENTAL_EXPORT',
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Obter detalhes sobre exportações anteriores no AWS SDK

Use esses trechos de código para obter detalhes sobre exportações de tabela anteriores usando o AWS SDK de sua escolha.

Python

Lista

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/list_exports.html

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
  )
)
```

Descrever

```
import boto3

client = boto3.client('dynamodb')
```

```
# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/describe_export.html

print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

Formato de saída de exportação da tabela do DynamoDB

Uma exportação de tabela do DynamoDB inclui arquivos de manifesto, além dos arquivos que contêm os dados da tabela. Esses arquivos são salvos no bucket do Amazon S3 especificado por você na solicitação de exportação. As seções a seguir descrevem o formato e o conteúdo de cada objeto de saída.

Saída de exportação completa

Arquivos de manifesto

O DynamoDB cria arquivos de manifesto, bem como os respectivos arquivos de soma de verificação, no bucket do S3 especificado para cada solicitação de exportação.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Você escolhe um **`export-prefix`** ao solicitar uma exportação de tabela. Isso ajuda a manter os arquivos no bucket do S3 de destino organizados. O **`ExportId`** é um token exclusivo gerado pelo serviço para garantir que várias exportações para o mesmo bucket do S3 e `export-prefix` não se sobreponham.

Essa exportação cria pelo menos um arquivo por partição. Para partições vazias, a solicitação de exportação criará um arquivo vazio. Todos os itens em cada arquivo são do espaço de chave com hash dessa partição específica.

> **Note**
>
> O DynamoDB também cria um arquivo vazio chamado `_started` no mesmo diretório dos arquivos de manifesto.' Esse arquivo verifica se o bucket de destino pode ser gravado e se a exportação foi iniciada. Ele pode ser excluído com segurança.

O manifesto resumido

O arquivo `manifest-summary.json` contém informações resumidas sobre o trabalho de exportação. Isso permite que você saiba quais arquivos de dados na pasta de dados compartilhada estão associados a essa exportação. Seu formato é o seguinte:

```
{
   "version": "2020-06-30",
   "exportArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4",
   "startTime": "2020-11-04T07:28:34.028Z",
   "endTime": "2020-11-04T07:33:43.897Z",
   "tableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog",
   "tableId": "12345a12-abcd-123a-ab12-1234abc12345",
   "exportTime": "2020-11-04T07:28:34.028Z",
   "s3Bucket": "ddb-productcatalog-export",
   "s3Prefix": "2020-Nov",
   "s3SseAlgorithm": "AES256",
   "s3SseKmsKeyId": null,
   "manifestFilesS3Key": "AWSDynamoDB/01693685827463-2d8752fd/manifest-files.json",
   "billedSizeBytes": 0,
   "itemCount": 8,
   "outputFormat": "DYNAMODB_JSON",
   "exportType": "FULL_EXPORT"
}
```

O manifesto de arquivos

O arquivo `manifest-files.json` contém informações sobre os arquivos que contêm os dados da tabela exportada. Como o arquivo está no formato linhas JSON, novas linhas são usadas como delimitadores de itens. No exemplo a seguir, os detalhes de um arquivo de dados de um manifesto de arquivos são formatados em várias linhas por razões de legibilidade.

```
{
```

```
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

Arquivos de dados

O DynamoDB pode exportar os dados da tabela em dois formatos: DynamoDB JSON e Amazon Ion. Independentemente do formato escolhido, os dados serão gravados em vários arquivos compactados nomeados pelas chaves. Esses arquivos também estão listados no arquivo `manifest-files.json`.

A estrutura de diretórios do bucket do Amazon S3 após uma exportação completa conterá todos os arquivos de manifesto e arquivos de dados na pasta de ID de exportação.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685827463-2d8752fd     // the single full export
    #   ### manifest-files.json     // manifest points to files under 'data' subfolder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### data                    // The data exported by full export
    #   #   ### asdl123dasas.json.gz
    #   #   ...
    #   ### _started                // empty file for permission check
```

DynamoDB JSON

Uma exportação de tabela no formato JSON do DynamoDB consiste em vários objetos `Item`. Cada objeto individual está no formato JSON organizado padrão do DynamoDB.

Ao criar analisadores personalizados para dados de exportação JSON do DynamoDB, o formato é linhas JSON. Isso significa que as novas linhas são usadas como delimitadores de item. Muitos serviços da AWS, como Athena e AWS Glue, analisarão esse formato automaticamente.

No exemplo a seguir, um único item de uma exportação DynamoDB JSON foi formatado em várias linhas para fins de legibilidade.

```
{
```

```
    "Item":{
        "Authors":{
            "SS":[
                "Author1",
                "Author2"
            ]
        },
        "Dimensions":{
            "S":"8.5 x 11.0 x 1.5"
        },
        "ISBN":{
            "S":"333-3333333333"
        },
        "Id":{
            "N":"103"
        },
        "InPublication":{
            "BOOL":false
        },
        "PageCount":{
            "N":"600"
        },
        "Price":{
            "N":"2000"
        },
        "ProductCategory":{
            "S":"Book"
        },
        "Title":{
            "S":"Book 103 Title"
        }
    }
}
```

Amazon Ion

O Amazon Ion é um formato de serialização de dados hierárquico, autodescritivo e altamente compatível com vários tipos de declaração e criado para lidar com os desafios de desacoplamento, desenvolvimento rápido e eficiência enfrentados todos os dias na criação de arquiteturas orientadas a serviços em grande escala. O DynamoDB oferece suporte à exportação de dados de tabelas no formato de texto do Ion, o qual é um superconjunto do JSON.

Quando você exporta uma tabela para o formato Ion, os tipos de dados do DynamoDB usados na tabela são mapeados em tipos de dados Ion. O DynamoDB define o uso de anotações do tipo Ion para eliminar ambiguidades do tipo de dados usado na tabela de origem.

A tabela a seguir lista o mapeamento entre os tipos de dados do DynamoDB e os tipos de dados do Ion:

Tipo de dados do DynamoDB	Representação no Ion
string (S)	string
Boolean (BOOL)	bool
número (N)	decimal
binário (B)	blob
Conjunto (SS, NS, BS)	list (com anotação de tipo $dynamodb_SS, $dynamodb_NS ou $dynamodb_BS)
Lista	list
Mapa	struct

Os itens em uma exportação do Ion são delimitados por novas linhas. Cada linha começa com um marcador de versão do Ion, seguido por um item no formato Ion. No exemplo a seguir, um item de uma exportação ION foi formatado em várias linhas para fins de legibilidade.

```
$ion_1_0 {
    Item:{
        Authors:$dynamodb_SS::["Author1","Author2"],
        Dimensions:"8.5 x 11.0 x 1.5",
        ISBN:"333-3333333333",
        Id:103.,
        InPublication:false,
        PageCount:6d2,
        Price:2d3,
        ProductCategory:"Book",
        Title:"Book 103 Title"
    }
```

```
}
```

Saída de exportação Incremental

Arquivos de manifesto

O DynamoDB cria arquivos de manifesto, bem como os respectivos arquivos de soma de verificação, no bucket do S3 especificado para cada solicitação de exportação.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Você escolhe um **`export-prefix`** ao solicitar uma exportação de tabela. Isso ajuda a manter os arquivos no bucket do S3 de destino organizados. O **`ExportId`** é um token exclusivo gerado pelo serviço para garantir que várias exportações para o mesmo bucket do S3 e `export-prefix` não se sobreponham.

Essa exportação cria pelo menos um arquivo por partição. Para partições vazias, a solicitação de exportação criará um arquivo vazio. Todos os itens em cada arquivo são do espaço de chave com hash dessa partição específica.

> **Note**
>
> O DynamoDB também cria um arquivo vazio chamado `_started` no mesmo diretório dos arquivos de manifesto.' Esse arquivo verifica se o bucket de destino pode ser gravado e se a exportação foi iniciada. Ele pode ser excluído com segurança.

O manifesto resumido

O arquivo `manifest-summary.json` contém informações resumidas sobre o trabalho de exportação. Isso permite que você saiba quais arquivos de dados na pasta de dados compartilhada estão associados a essa exportação. Seu formato é o seguinte:

```
{
 "version": "2023-08-01",
 "exportArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test/
export/01695097218000-d6299cbd",
```

```
 "startTime": "2023-09-19T04:20:18.000Z",
 "endTime": "2023-09-19T04:40:24.780Z",
 "tableArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test",
 "tableId": "b116b490-6460-4d4a-9a6b-5d360abf4fb3",
 "exportFromTime": "2023-09-18T17:00:00.000Z",
 "exportToTime": "2023-09-19T04:00:00.000Z",
 "s3Bucket": "jason-exports",
 "s3Prefix": "20230919-prefix",
 "s3SseAlgorithm": "AES256",
 "s3SseKmsKeyId": null,
 "manifestFilesS3Key": "20230919-prefix/AWSDynamoDB/01693685934212-ac809da5/manifest-
files.json",
 "billedSizeBytes": 20901239349,
 "itemCount": 169928274,
 "outputFormat": "DYNAMODB_JSON",
 "outputView": "NEW_AND_OLD_IMAGES",
 "exportType": "INCREMENTAL_EXPORT"
}
```


O manifesto de arquivos

O arquivo `manifest-files.json` contém informações sobre os arquivos que contêm os dados da tabela exportada. Como o arquivo está no formato linhas JSON, novas linhas são usadas como delimitadores de itens. No exemplo a seguir, os detalhes de um arquivo de dados de um manifesto de arquivos são formatados em várias linhas por razões de legibilidade.


```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```


Arquivos de dados

O DynamoDB pode exportar os dados da tabela em dois formatos: DynamoDB JSON e Amazon Ion. Independentemente do formato escolhido, os dados serão gravados em vários arquivos compactados nomeados pelas chaves. Esses arquivos também estão listados no arquivo `manifest-files.json`.

Os arquivos de dados para exportações incrementais estão todos contidos em uma pasta de dados comum no bucket do S3. Os arquivos de manifesto estão na pasta de ID de exportação.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685934212-ac809da5     // an incremental export ID
    #   ### manifest-files.json     // manifest points to files under 'data' folder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### _started                // empty file for permission check
    ### 01693686034521-ac809da5
    #   ### manifest-files.json
    #   ### manifest-files.checksum
    #   ### manifest-summary.json
    #   ### manifest-summary.md5
    #   ### _started
    ### data                        // stores all the data files for incremental
 exports
    #   ### sgad6417s6vss4p7owp0471bcq.json.gz
    #   ...
```

Nos arquivos de exportação, a saída de cada item inclui um carimbo de data/hora que representa quando o item foi atualizado na tabela e uma estrutura de dados que indica se foi uma operação `insert`, `update` ou `delete`. O carimbo de data/hora baseia-se em um relógio interno do sistema e pode divergir do relógio da aplicação. Para exportações incrementais, você pode escolher entre dois tipos de visualização de exportação para a estrutura de saída: Imagens novas e antigas ou Somente novas imagens.

- Imagem nova fornece o estado mais recente do item
- Imagem antiga fornece o estado do item imediatamente antes da data e hora de início especificadas

Os tipos de visualização podem ser úteis se você quiser ver como o item foi alterado no período de exportação. Também pode ser útil para atualizar com eficiência os sistemas subsequentes, especialmente se esses sistemas tiverem uma chave de partição diferente da chave de partição do DynamoDB.

Você pode inferir se um item na saída de exportação incremental era `insert`, `update` ou `delete` observando a estrutura da saída. A estrutura de exportação incremental e as operações correspondentes estão resumidas na tabela abaixo para os dois tipos de visualização de exportação.

Operation	Somente novas imagens	Imagens novas e antigas
Inserir	Chaves + imagem nova	Chaves + imagem nova
Atualizar	Chaves + imagem nova	Chaves + imagem nova + imagem antiga
Delete	Chaves	Chaves + imagem antiga
Inserir + excluir	Sem saída	Sem saída

DynamoDB JSON

Uma exportação de tabela no formato JSON do DynamoDB consiste em um carimbo de data/hora de metadados que indica a hora de gravação do item, seguido das chaves do item e dos valores. Veja a seguir um exemplo de saída JSON do DynamoDB usando o tipo de visualização de exportação como Imagens novas e antigas.

```
// Ex 1: Insert
//   An insert means the item did not exist before the incremental export window
//   and was added during the incremental export window

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#100"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#100"
     },
     "FirstName": {
       "S": "John"
     },
     "LastName": {
       "S": "Don"
     }
```

```
   }
}

// Ex 2: Update
//   An update means the item existed before the incremental export window
//   and was updated during the incremental export window.
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#200"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Grace"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Smith"
     }
   }
}

// Ex 3: Delete
//   A delete means the item existed before the incremental export window
//   and was deleted during the incremental export window
```

```
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#300"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#300"
     },
     "FirstName": {
       "S": "Jose"
     },
     "LastName": {
       "S": "Hernandez"
     }
   }
}

// Ex 4: Insert + Delete
//   Nothing is exported if an item is inserted and deleted within the
//   incremental export window.
```

Amazon Ion

O Amazon Ion é um formato de serialização de dados hierárquico, autodescritivo e altamente compatível com vários tipos de declaração e criado para lidar com os desafios de desacoplamento, desenvolvimento rápido e eficiência enfrentados todos os dias na criação de arquiteturas orientadas a serviços em grande escala. O DynamoDB oferece suporte à exportação de dados de tabelas no formato de texto do Ion, o qual é um superconjunto do JSON.

Quando você exporta uma tabela para o formato Ion, os tipos de dados do DynamoDB usados na tabela são mapeados em tipos de dados Ion. O DynamoDB define o uso de anotações do tipo Ion para eliminar ambiguidades do tipo de dados usado na tabela de origem.

A tabela a seguir lista o mapeamento entre os tipos de dados do DynamoDB e os tipos de dados do Ion:

Tipo de dados do DynamoDB	Representação no Ion
string (S)	string
Boolean (BOOL)	bool
número (N)	decimal
binário (B)	blob
Conjunto (SS, NS, BS)	list (com anotação de tipo $dynamodb_SS, $dynamodb_NS ou $dynamodb_BS)
Lista	list
Mapa	struct

Os itens em uma exportação do Ion são delimitados por novas linhas. Cada linha começa com um marcador de versão do Ion, seguido por um item no formato Ion. No exemplo a seguir, um item de uma exportação ION foi formatado em várias linhas para fins de legibilidade.

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:false,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        },
        NewImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            Dimensions:"8.5 x 11.0 x 1.5",
            ISBN:"333-3333333333",
```

```
            Id:103.,
            InPublication:true,
            PageCount:6d2,
            Price:2d3,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        }
    }
}
```

Integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse

A integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse elimina a necessidade de criar pipelines personalizados de movimentação de dados ao replicar automaticamente os dados do DynamoDB para o Amazon SageMaker Lakehouse. Essa integração sem código ajuda os clientes a executar workloads de analytics nos dados do DynamoDB usando o Amazon SageMaker Lakehouse sem consumir nenhuma capacidade de tabela do DynamoDB. A integração exporta automaticamente os dados da sua tabela e mantém o destino atualizado, normalmente dentro de 15 a 30 minutos.

Tópicos

- Integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse

Integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse

Configurar uma integração entre a tabela do DynamoDB e o Amazon SageMaker Lakehouse exige alguns pré-requisitos, como a configuração de perfis do IAM que o AWS Glue usa para acessar dados da origem e gravar no destino, e o uso de chaves do KMS para criptografar os dados no local intermediário ou de destino.

Tópicos

- Pré-requisitos antes de criar uma integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse
- Criar uma integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse
- Visualizar métricas do CloudWatch para integração

Pré-requisitos antes de criar uma integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse

Para configurar uma integração ETL zero com uma origem do DynamoDB, você precisa configurar uma política de acesso baseado em recursos (RBAC) que permita que o AWS Glue acesse e exporte dados da tabela do DynamoDB. A política deve incluir permissões específicas, como `ExportTableToPointInTime`, `DescribeTable` e `DescribeExport`, com condições que restrinjam o acesso a uma Conta da AWS e uma região específicas. Consulte Configurar uma origem do Amazon DynamoDB para receber mais informações.

A recuperação para um ponto no tempo (PITR) precisa estar habilitada para a tabela, e é possível aplicar a política usando comandos da AWS CLI. A política pode ser ainda mais refinada especificando-se o ARN de integração completo para um controle de acesso mais restritivo. Para ter mais informações, consulte Pré-requisitos para configurar uma integração ETL zero.

Criar uma integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse

Depois que você concluir os pré-requisitos de integração, poderá criar, modificar ou excluir a integração ETL zero seguindo as orientações abaixo:

Criar uma integração

Para criar uma integração

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon DynamoDB em https://console.aws.amazon.com/dynamodbv2.
2. No painel de navegação, selecione Integrações.
3. Selecione Criar integração ETL zero com o Amazon SageMaker Lakehouse e escolha Próximo.
4. Para criar uma integração, consulte Criar uma integração.
5. Para modificar uma integração, consulte Modificar uma integração.
6. Para excluir uma integração, consulte Excluir uma integração.
7. Para configurar uma integração entre contas, consulte Configurar uma integração entre contas.

Habilitar a compactação nas tabelas de destino do Amazon S3

É possível habilitar a compactação para melhorar a performance das consultas no Amazon Athena.

Primeiro, conclua a configuração de pré-requisitos para recursos de compactação, como a configuração do perfil do IAM necessário. Consulte a documentação do Lake Formation para ver as etapas detalhadas de configuração do perfil do IAM. Consulte Otimizar tabelas para compactação.

Para habilitar a compactação na tabela do AWS Glue criada durante a integração, siga o processo de habilitação da compactação do Lake Formation. Isso ajudará a otimizar a performance e a eficiência das consultas da tabela.

Visualizar métricas do CloudWatch para integração

Depois que a integração for concluída, você poderá ver essas métricas do CloudWatch e as notificações do EventBridge geradas em sua conta para cada trabalho do AWS Glue. Para ter mais informações, consulte Monitorar uma integração.

Integração ETL zero do DynamoDB com o Amazon OpenSearch Service

O Amazon DynamoDB oferece uma integração ETL zero ao Amazon OpenSearch Service por meio do plug-in do DynamoDB para OpenSearch Ingestion. O Amazon OpenSearch Ingestion oferece uma experiência totalmente gerenciada e sem código para a ingestão de dados no Amazon OpenSearch Service.

Com o plug-in do DynamoDB para OpenSearch Ingestion, é possível usar uma ou mais tabelas do DynamoDB como fonte para ingestão em um ou mais índices do OpenSearch Service. É possível navegar e configurar os pipelines do OpenSearch Ingestion com o DynamoDB como fonte do OpenSearch Ingestion ou do DynamoDB Integrations no AWS Management Console.

- Comece a usar o OpenSearch Ingestion seguindo o Guia de introdução do OpenSearch Ingestion.
- Conheça os pré-requisitos e todas as opções de configuração do plug-in do DynamoDB na documentação do plug-in do DynamoDB para OpenSearch Ingestion.

Como funciona

O plug-in usa a exportação do DynamoDB para Amazon S3 para criar um snapshot inicial para carregar no OpenSearch. Depois que o snapshot for carregado, o plug-in usa o DynamoDB Streams para replicar quaisquer outras alterações quase em tempo real. Cada item é processado como um evento no OpenSearch Ingestion e pode ser modificado com plug-ins de processador. É possível eliminar atributos ou criar atributos compostos e enviá-los para índices diferentes por meio de rotas.

É necessário ter a recuperação para um ponto no tempo (PITR) habilitada para usar a exportação para o Amazon S3. Também é necessário ter o DynamoDB Streams habilitado (com a opção imagens novas e antigas selecionada) para poder usá-lo. É possível criar um pipeline sem tirar um snapshot excluindo as configurações de exportação.

Você também pode criar um pipeline com apenas um snapshot e sem atualizações excluindo as configurações do Streams. O plug-in não usa throughput de leitura ou gravação em sua tabela, portanto, é seguro usá-lo sem afetar o tráfego de produção. Há limites para o número de consumidores paralelos em um fluxo em que você deve pensar antes de criar essa ou outras integrações. Para saber outras considerações, consulte the section called “Práticas recomendadas de integração”.

Para pipelines simples, uma única unidade de computação do OpenSearch (OCU) pode processar cerca de 1 MB por segundo de gravações. Isso equivale a cerca de mil unidades de solicitação de gravação (WCUs). Dependendo da complexidade do pipeline e de outros fatores, você pode conseguir mais ou menos do que isso.

O OpenSearch Ingestion comporta uma fila de mensagens não entregues (DLQ) para eventos que causam erros irrecuperáveis. Além disso, o pipeline pode continuar de onde parou sem a intervenção do usuário, mesmo se houver uma interrupção do serviço com o DynamoDB, o pipeline ou o Amazon OpenSearch Service.

Se a interrupção persistir por mais de 24 horas, isso poderá causar a perda de atualizações. No entanto, o pipeline continuaria processando as atualizações ainda disponíveis quando a disponibilidade fosse restaurada. Você precisaria criar um índice para corrigir quaisquer irregularidades devido aos eventos descartados, a menos que eles estivessem na fila de mensagens não entregues.

Para ver todas as configurações e os detalhes do plug-in, consulte a documentação do plug-in do OpenSearch Ingestion DynamoDB.

Experiência de criação integrada por meio do console

O DynamoDB e o OpenSearch Service têm uma experiência integrada no AWS Management Console, o que simplifica o processo inicial. Quando você passar por essas etapas, o serviço selecionará automaticamente o esquema do DynamoDB e adicionará as informações apropriadas do DynamoDB para você.

Para criar uma integração, acompanhe o Guia de introdução do OpenSearch Ingestion. Quando você chegar à Etapa 3: Criar um pipeline, substitua as etapas 1 e 2 pelas seguintes:

1. Navegue até o console do DynamoDB.
2. No painel de navegação à esquerda, selecione Integração.
3. Selecione a tabela do DynamoDB que você deseja replicar no OpenSearch.
4. Escolha Criar.

A partir daqui, você pode continuar com o restante do tutorial.

Próximas etapas

Para entender melhor como o DynamoDB se integra ao OpenSearch Service, consulte o seguinte:

- Conceitos básicos do Amazon OpenSearch Ingestion
- Configuração e requisitos do plug-in do DynamoDB

Lidar com alterações significativas no índice

O OpenSearch pode adicionar dinamicamente novos atributos ao índice. No entanto, depois que o modelo de mapeamento for definido para uma chave específica, você precisará tomar medidas adicionais para alterá-lo. Além disso, se a alteração exigir o reprocessamento de todos os dados na tabela do DynamoDB, você precisará tomar medidas para iniciar uma nova exportação.

> **Note**
>
> Em todas essas opções, você ainda poderá ter problemas se sua tabela do DynamoDB tiver conflitos de tipo com o modelo de mapeamento especificado. Certifique-se de ter uma fila de mensagens não entregues (DLQ) habilitada (mesmo em desenvolvimento). Isso facilita a compreensão do que pode estar errado com o registro que causa um conflito quando ele está sendo indexado no índice no OpenSearch.

Tópicos

- Como funciona
- Excluir o índice e redefinir o pipeline (opção centrada no pipeline)
- Recriar o índice e redefinir o pipeline (opção centrada no pipeline)
- Criar um índice e um coletor (opção on-line)
- Práticas recomendadas para evitar e depurar conflitos de tipos

Como funciona

Tenha uma visão geral rápida das ações realizadas ao lidar com alterações significativas no índice. Veja os procedimentos passo a passo nas seções a seguir.

- Parar e iniciar o pipeline: essa opção redefine o estado do pipeline, e o pipeline será reiniciado com uma nova exportação completa. Ela não é destrutiva e, portanto, não exclui o índice nem qualquer dado no DynamoDB. Se você não criar um índice antes de fazer isso, poderá ver um grande número de erros de conflitos de versão porque a exportação tentará inserir documentos mais antigos do que a `_version` atual no índice. É possível ignorar esses erros com segurança. Você não será cobrado pelo pipeline enquanto ele estiver parado.
- Atualizar o pipeline: essa opção atualiza a configuração no pipeline com uma abordagem azul/verde, sem perder nenhum estado. Se você fizer alterações significativas no pipeline (como adicionar novas rotas, índices ou chaves aos índices existentes), talvez seja necessário fazer uma redefinição completa do pipeline e recriar o índice. Essa opção não realiza uma exportação completa.
- Excluir e recriar o índice: essa opção remove os dados e as configurações de mapeamento do índice. Você deve fazer isso antes de fazer qualquer alteração significativa nos mapeamentos. Isso interromperá qualquer aplicação que dependa do índice até que este seja recriado e sincronizado. A exclusão do índice não inicia uma nova exportação. Você deve excluir o índice somente depois de atualizar o pipeline. Caso contrário, o índice poderá ser recriado antes de você atualizar suas configurações.

Excluir o índice e redefinir o pipeline (opção centrada no pipeline)

Esse método geralmente é a opção mais rápida se você ainda estiver no desenvolvimento. Você excluirá o índice no OpenSearch Service e, depois, interromperá e iniciará o pipeline para iniciar uma nova exportação de todos os dados. Isso garante que não haja conflitos de modelos de mapeamento com índices existentes e nenhuma perda de dados de uma tabela processada incompleta.

1. Pare o pipeline por meio do AWS Management Console, ou usando a operação de API StopPipeline com a AWS CLI ou um SDK.
2. Atualize a configuração do pipeline com suas novas alterações.
3. Exclua o índice no OpenSearch Service, seja por meio de uma chamada de API REST ou do painel do OpenSearch.

4. Pare o pipeline por meio do console ou usando a operação de API `StartPipeline` com a AWS CLI ou um SDK.

   > **Note**
   >
   > Isso inicia uma nova exportação completa, o que acarretará custos adicionais.

5. Monitore quaisquer problemas inesperados porque uma nova exportação é gerada para criar o índice.
6. Confirme se o índice corresponde às suas expectativas no OpenSearch Service.

Depois que a exportação for concluída e ela retomar a leitura do fluxo, os dados da tabela do DynamoDB agora estarão disponíveis no índice.

Recriar o índice e redefinir o pipeline (opção centrada no pipeline)

Esse método funcionará bem se você precisar fazer várias iterações no design do índice no OpenSearch Service antes de retomar o pipeline do DynamoDB. Isso pode ser útil para o desenvolvimento quando você deseja iterar rapidamente seus padrões de pesquisa e não esperar que novas exportações sejam concluídas entre cada iteração.

1. Pare o pipeline por meio do AWS Management Console ou chamando a operação de API StopPipeline com a AWS CLI ou um SDK.
2. Exclua e recrie o índice no OpenSearch com o modelo de mapeamento que você deseja usar. Você pode inserir manualmente alguns dados de amostra para confirmar que suas pesquisas estão funcionando conforme o esperado. Se os dados de amostra entrarem em conflito com qualquer dado do DynamoDB, não se esqueça de excluí-los antes de passar para a próxima etapa.
3. Se você tiver um modelo de indexação no pipeline, remova-o ou substitua-o por aquele que você já criou no OpenSearch Service. Garanta que o nome do índice corresponda ao nome no pipeline.
4. Inicie o pipeline por meio do console ou chamando a operação de API `StartPipeline` com a AWS CLI ou um SDK.

   > **Note**
   >
   > Isso iniciará uma nova exportação completa, o que acarretará custos adicionais.

5. Monitore quaisquer problemas inesperados porque uma nova exportação é gerada para criar o índice.

Depois que a exportação for concluída e ela retomar a leitura do fluxo, os dados da tabela do DynamoDB agora estarão disponíveis no índice.

Criar um índice e um coletor (opção on-line)

Esse método funcionará bem se você precisar atualizar o modelo de mapeamento, mas estiver usando seu índice na produção. Isso cria um índice totalmente novo, para o qual você precisará mover a aplicação depois de sincronizado e validado.

> **Note**
>
> Isso criará outro consumidor no fluxo. Isso pode ser um problema caso você também tenha outros consumidores, como AWS Lambda ou tabelas globais. Talvez seja necessário pausar as atualizações do pipeline existente a fim de criar capacidade para carregar o novo índice.

1. Crie um pipeline com novas configurações e um nome de índice diferente.
2. Monitore o novo índice em busca de problemas inesperados.
3. Troque a aplicação pelo novo índice.
4. Pare e exclua o pipeline antigo depois de validar se tudo está funcionando corretamente.

Práticas recomendadas para evitar e depurar conflitos de tipos

- Sempre use uma fila de mensagens não entregues (DLQ) para facilitar a depuração quando há conflitos de tipos.
- Sempre use um modelo de índice com mapeamentos e defina `include_keys`. Embora o OpenSearch Service associe dinamicamente novas chaves, isso pode causar problemas com comportamentos inesperados (como esperar que algo seja um `GeoPoint`, mas é criado como `string` ou `object`) ou erros (como ter um `number` que seja uma mistura de valores `long` e `float`).
- Se precisar manter o índice existente funcionando na produção, também poderá substituir qualquer uma das etapas anteriores de exclusão do índice simplesmente renomeando o índice

no arquivo de configuração do pipeline. Isso cria um índice totalmente novo. Depois, a aplicação precisará ser atualizada para apontar para o novo índice depois de concluído.

- Se você tiver um problema de conversão de tipo que você corrija com um processador, poderá testar isso com `UpdatePipeline`. Para isso, você precisará interromper e iniciar ou processar as filas de mensagens não entregues para corrigir quaisquer documentos com erros ignorados anteriormente.

Práticas recomendadas para trabalhar com a Integração ETL zero do DynamoDB e o OpenSearch Service

O DynamoDB tem uma integração ETL zero do DynamoDB com o Amazon OpenSearch Service. Para ter mais informações, consulte o plug-in do DynamoDB para ingestão do OpenSearch e as práticas recomendadas específicas para o Amazon OpenSearch Service.

Configuração

- Somente indexe os dados nos quais você precisa realizar pesquisas. Sempre use um modelo de mapeamento (`template_type: index_template` e `template_content`) e `include_keys` para implementar isso.
- Monitore os logs em busca de erros relacionados a conflitos de tipos. O OpenSearch Service espera que todos os valores de uma chave específica tenham o mesmo tipo. Ele gera exceções quando há incompatibilidades. Se você encontrar um desses erros, poderá adicionar um processador para conferir se uma chave específica sempre tem o mesmo valor.
- Geralmente, use o valor de metadados `primary_key` para o valor `document_id`. No OpenSearch Service, o ID do documento é equivalente à chave primária no DynamoDB. O uso da chave primária facilitará a localização do documento e garantirá que as atualizações sejam replicadas de forma consistente e sem conflitos.

  É possível usar a função auxiliar `getMetadata` para ter a chave primária (por exemplo, `document_id: "${getMetadata('primary_key')}"`). Se você estiver usando uma chave primária composta, a função auxiliar as concatenará para você.
- Em geral, use o valor de metadados `opensearch_action` para a configuração `action`. Isso garantirá que as atualizações sejam replicadas de forma que os dados no OpenSearch Service correspondam ao estado mais recente no DynamoDB.

É possível usar a função auxiliar `getMetadata` para ter a chave primária (por exemplo, `action: "${getMetadata('opensearch_action')}"`). Também é possível ter o tipo de evento de fluxos por meio de `dynamodb_event_name` para casos de uso como filtragem. No entanto, normalmente, você não deve usá-lo para a configuração `action`.

Observabilidade

- Sempre use uma fila de mensagens não entregues (DLQ) em seus coletores do OpenSearch para lidar com eventos descartados. O DynamoDB geralmente é menos estruturado do que o OpenSearch Service, e sempre é possível que algo inesperado aconteça. Com uma fila de mensagens não entregues, é possível recuperar eventos individuais e até mesmo automatizar o processo de recuperação. Isso ajudará você a evitar a necessidade de recriar todo o índice.
- Sempre defina alertas de que o atraso na replicação não ultrapassa o valor esperado. Normalmente, é seguro presumir um minuto para que o alerta não seja muito ruidoso. Isso pode variar dependendo da intensidade do tráfego de gravação e das configurações da unidade de computação do OpenSearch (OCU) no pipeline.

  Se o atraso na replicação ultrapassar 24 horas, o fluxo começará a descartar eventos e você terá problemas de precisão, a menos que faça uma recriação completa do índice.

Escalabilidade

- Use o ajuste de escala automático para pipelines a fim de ajudar a aumentar ou reduzir a escala verticalmente das OCUs e melhor atender à workload.
- Para tabelas de throughput provisionadas sem ajuste de escala automático, recomendamos configurar OCUs com base nas unidades de capacidade de gravação (WCUs) divididas por mil. Defina o mínimo como 1 OCU abaixo desse valor (mas pelo menos 1) e defina o máximo como pelo menos 1 OCU acima desse valor.
  - Fórmula:

    ```
    OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
    OCU_maximum = (table_WCU / 1000) + 1
    ```

  - Exemplo: sua tabela tem 25.000 WCUs provisionadas. As OCUs do pipeline devem ser definidas com um mínimo de 24 (25.000/1.000 - 1) e máximo de pelo menos 26 (25.000/1.000 + 1).

- Para tabelas de throughput provisionadas sem ajuste de escala automático, recomendamos configurar OCUs com base nas WCUs divididas por 1.000. Defina o mínimo como 1 OCU abaixo do mínimo do DynamoDB e defina o máximo como pelo menos 1 OCU acima do máximo do DynamoDB.
  - Fórmula:

    ```
    OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
    OCU_maximum = (table_maximum_WCU / 1000) + 1
    ```

  - Exemplo: sua tabela tem uma política de ajuste de escala automático com um mínimo de 8.000 e máximo de 14.000. As OCUs do pipeline devem ser definidas com um mínimo de 7 (8.000/1.000 - 1) e um máximo de 15 (14.000/1.000 + 1).
- Para tabelas de throughput sob demanda, recomendamos configurar OCUs com base em picos e vales típicos para unidades de solicitação de gravação por segundo. Talvez seja necessário calcular a média por um longo período, dependendo da agregação disponível para você. Defina o mínimo como 1 OCU abaixo do mínimo do DynamoDB e defina o máximo como pelo menos 1 OCU acima do máximo do DynamoDB.
  - Fórmula:

    ```
    # Assuming we have writes aggregated at the minute level
    OCU_minimum = GREATEST((min(table_writes_1min) / (60 * 1000)) - 1, 1)
    OCU_maximum = (max(table_writes_1min) / (60 * 1000)) + 1
    ```

  - Exemplo: sua tabela tem um vale médio de 300 unidades de solicitação de gravação por segundo e um pico médio de 4.300. As OCUs do pipeline devem ser definidas com um mínimo de 1 (300/1.000 - 1, mas pelo menos 1) e máximo de 5 (4.300/1.000 + 1).
- Siga as práticas recomendadas para escalar os índices de destino do OpenSearch Service. Se os índices estiverem subescalados, isso diminuirá a ingestão do DynamoDB e poderá causar atrasos.

> **Note**
>
> `GREATEST` é uma função SQL que, considerando-se um conjunto de argumentos, exibe o argumento com o maior valor.

Integrar o DynamoDB com o Amazon EventBridge

O Amazon DynamoDB oferece o DynamoDB Streams para captura de dados de alterações, permitindo a captura de alterações em nível de item nas tabelas do DynamoDB. O DynamoDB Streams pode invocar funções do Lambda para processar essas alterações, possibilitando a integração orientada por eventos a outros serviços e aplicações. O DynamoDB Streams também oferece suporte à filtragem, o que permite um processamento de eventos eficiente e direcionado.

O DynamoDB Streams oferece suporte a até dois consumidores simultâneos por fragmento e oferece suporte à filtragem por meio da filtragem de eventos do Lambda para que somente itens que correspondam a critérios específicos sejam processados. Alguns clientes podem ter requisitos para compatibilidade com mais de dois consumidores. Outros podem precisar enriquecer os eventos de alteração antes de serem processados ou usar filtragem e roteamento mais avançados.

A integração do DynamoDB com o EventBridge pode atender a esses requisitos.

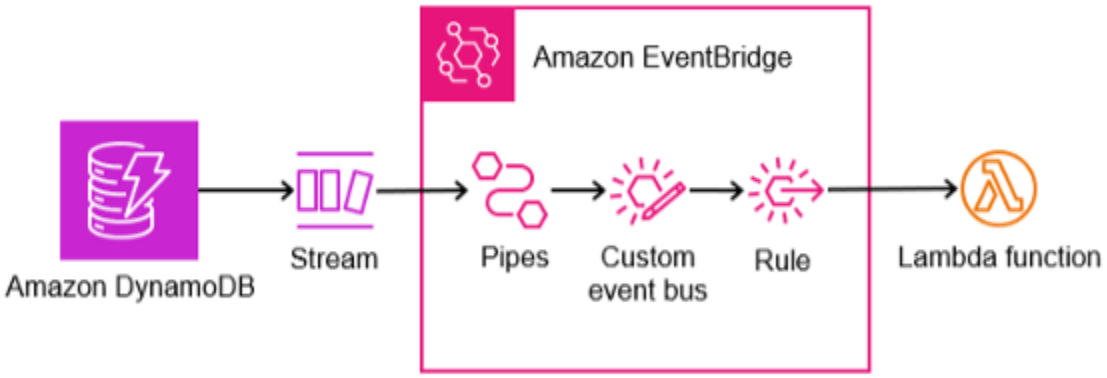
O Amazon EventBridge é um serviço sem servidor que usa eventos para conectar os componentes da aplicação, facilitando a criação de aplicações escaláveis orientadas por eventos. O EventBridge oferece integração nativa com o Amazon DynamoDB por meio de pipes do EventBridge, permitindo um fluxo contínuo de dados do DynamoDB para um barramento do EventBridge. Esse barramento pode se espalhar para várias aplicações e serviços por meio de um conjunto de regras e destinos.

Tópicos

- Como funciona
- Criar uma integração por meio do console
- Próximas etapas

Como funciona

A integração entre o DynamoDB e os pipes do EventBridge usa o DynamoDB Streams para capturar uma sequência em ordem temporal de alterações em nível de item em uma tabela do DynamoDB. Cada registro capturado dessa forma contém os dados modificados na tabela.



Um pipe do EventBridge consome eventos do DynamoDB Streams e os encaminha para um destino, como um barramento do EventBridge (um barramento de eventos é um roteador que recebe eventos e os entrega aos destinos). A entrega é baseada nas regras que correspondem ao conteúdo do evento. O pipe também pode incluir a capacidade de filtrar eventos específicos e realizar enriquecimentos nos dados do evento antes de enviá-los ao destino.

Embora o EventBridge ofereça suporte a vários tipos de destino, uma escolha comum ao implementar um design distribuído é usar uma função do Lambda como destino. O exemplo a seguir demonstra uma integração com uma função do Lambda como destino.

Criar uma integração por meio do console

Siga as etapas abaixo para criar uma integração por meio do AWS Management Console.

1. Habilite o DynamoDB Streams na tabela de origem seguindo as etapas na seção Habilitar um fluxo do Guia do desenvolvedor do DynamoDB. Se o DynamoDB Streams já estiver habilitado na tabela de origem, verifique se há menos de dois consumidores no momento. Os consumidores podem ser funções do Lambda, tabelas globais do DynamoDB, integrações ETL zero do Amazon DynamoDB com o Amazon OpenSearch Service ou aplicações que leem diretamente de fluxos, como por meio do adaptador do DynamoDB Streams Kinesis.
2. Crie um barramento de eventos do EventBridge seguindo as etapas na seção Creating an Amazon EventBridge event bus do Guia do usuário do EventBridge.
   a. Ao criar o barramento de eventos, habilite a opção Descoberta de esquemas.
3. Crie um pipe do EventBridge seguindo as etapas na seção Creating an Amazon EventBridge pipe do Guia do usuário do EventBridge.

a. Ao configurar a origem, selecione DynamoDB no campo Origem e selecione o nome do fluxo da tabela de origem no campo DynamoDB Streams.

b. Ao configurar o destino, selecione Barramento de eventos do EventBridge no campo Serviço de destino e selecione o barramento de eventos criado na etapa 2 no campo Barramento de eventos como destino.

4. Grave um item de exemplo na tabela de origem do DynamoDB para acionar um evento. Isso permitirá que o EventBridge faça a inferência do esquema com base no item de exemplo. Esse esquema pode ser usado para criar regras para eventos de roteamento. Por exemplo, se você estiver implementando um padrão de design que envolve sobrecarga de atributos, talvez queira acionar regras diferentes, dependendo do valor da sua chave de classificação. Detalhes sobre como gravar um item no DynamoDB podem ser encontrados na seção Trabalhar com itens e atributos do Guia do desenvolvedor do DynamoDB.

5. Crie um exemplo de função do Lambda em Python para ser usada como destino, seguindo as etapas na seção Criar funções do Lambda com Python do Guia do desenvolvedor do Lambda. Ao criar sua função, você pode usar o exemplo de código abaixo para demonstrar a integração. Quando invocado, ele imprimirá a `NewImage` e a `OldImage` recebidas com o evento, que podem ser visualizadas no CloudWatch Logs.

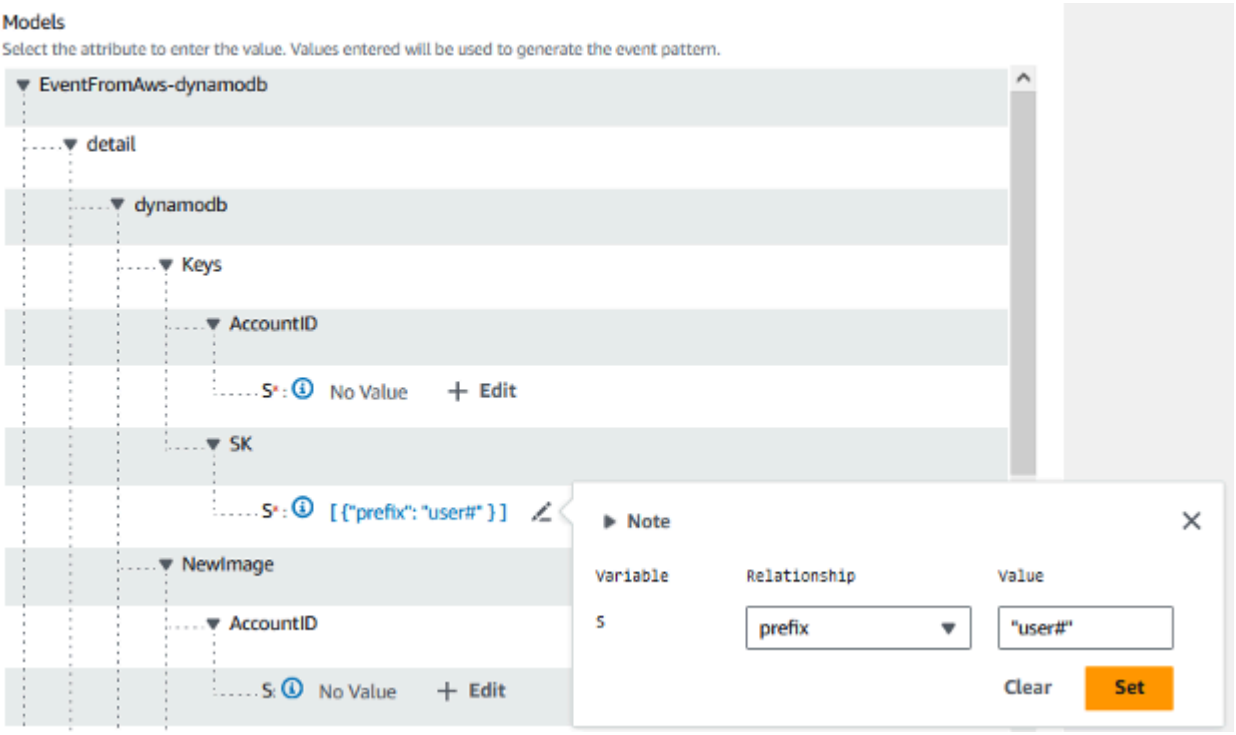
```
import json

def lambda_handler(event, context):
    dynamodb = event.get('detail', {}).get('dynamodb', {})
    new_image = dynamodb.get('NewImage')
    old_image = dynamodb.get('OldImage')

    if new_image:
        print("NewImage:", json.dumps(new_image, indent=2))
    if old_image:
        print("OldImage:", json.dumps(old_image, indent=2))

    return {'statusCode': 200, 'body': json.dumps(event)}
```

6. Crie uma regra do EventBridge que roteará eventos para a nova função do Lambda, seguindo as etapas na seção Create a rule that reacts to events do Guia do usuário do EventBridge.

a. Ao definir os detalhes da regra, selecione o nome do barramento de eventos que você criou na etapa 2 em Barramento de eventos.

b. Ao criar o padrão de eventos, siga o guia para Existing schema. Aqui, você pode selecionar o registro discovered-schemas e o esquema descoberto para o seu evento. Isso permite que

você configure um padrão de eventos específico para seu caso de uso que encaminha somente mensagens que correspondem a atributos específicos. Por exemplo, se quisesse estabelecer correspondência somente com itens do DynamoDB em que o SK começa com "`user#`", você usaria uma configuração como essa.



c. Clique em Gerar padrão de eventos em JSON depois de terminar de criar um padrão em relação ao seu esquema. Em vez disso, se você quiser estabelecer correspondência com todos os eventos que aparecem no DynamoDB Streams, use o JSON a seguir para o padrão de eventos.

```
{
  "source": ["aws.dynamodb"]
}
```

d. Ao selecionar destinos, siga o guia para serviço da AWS. No campo Selecionar um destino, escolha "Função do Lambda". No campo Função, selecione a função do Lambda criada na etapa 5.

7. Agora você pode interromper a descoberta de esquemas em seu barramento de eventos seguindo as etapas na seção Starting or stopping schema discovery on event buses do Guia do usuário do EventBridge.
8. Grave um segundo item de exemplo na tabela de origem do DynamoDB para acionar um evento. Valide se o evento foi processado com sucesso em cada etapa.

a. Veja a métrica PutEventsApproximateSuccessCount do CloudWatch para o barramento de eventos, seguindo a seção Monitoring Amazon EventBridge do Guia do usuário do EventBridge.

b. Veja os logs da sua função do Lambda seguindo a seção Monitoramento e solução de problemas de funções do Lambda do Guia do desenvolvedor do Lambda. Se a função do Lambda usar o exemplo de código fornecido, você deverá ver a `NewImage` e a `OldImage` do DynamoDB Streams impressa no grupo de logs do CloudWatch Logs.

c. Veja a métrica Error count and success rate (%) (Contagem de erros e taxa de sucesso [%]) da função do Lambda seguindo a seção Monitoramento e solução de problemas de funções do Lambda do Guia do desenvolvedor do Lambda.

Próximas etapas

Este exemplo fornece uma integração básica com uma única função do Lambda como destino. Para entender melhor as configurações mais complexas, como criação de várias regras, criação de vários destinos, integração com outros serviços e enriquecimento de eventos, consulte o Guia do usuário do EventBridge completo: Getting started with EventBridge.

> **Note**
>
> Esteja ciente de todas as cotas do EventBridge que possam ser relevantes para sua aplicação. Enquanto a capacidade do DynamoDB Streams escala com a tabela, as cotas do EventBridge são separadas. As cotas comuns a serem observadas em uma aplicação grande são o Limite de controle de utilização de invocações em transações por segundo e o Limite de controle de utilização de PutEvents em transações por segundo. Essas cotas especificam o número de invocações que podem ser enviadas aos destinos e o número de eventos que podem ser colocados no barramento por segundo.

Integrar o DynamoDB com o Amazon Managed Streaming for Apache Kafka

O Amazon Managed Streaming for Apache Kafka (Amazon MSK) facilita a ingestão e o processamento de dados de streaming em tempo real com um serviço Apache Kafka totalmente gerenciado e altamente disponível.

O Apache Kafka é um datastore distribuído otimizado para a ingestão e processamento de dados de streaming em tempo real. O Kafka pode processar fluxos de registros, armazenar efetivamente fluxos de registros na ordem em que eles foram gerados, além de publicar e assinar fluxos de registros.

Por conta desses recursos, o Apache Kafka é frequentemente usado para criar pipelines de dados de streaming em tempo real. Um pipeline de dados processa e movimenta dados de um sistema para outro de maneira confiável e pode ser uma parte importante da adoção de uma estratégia de banco de dados com propósito específico, facilitando o uso de vários bancos de dados, cada um comportando diferentes casos de uso.

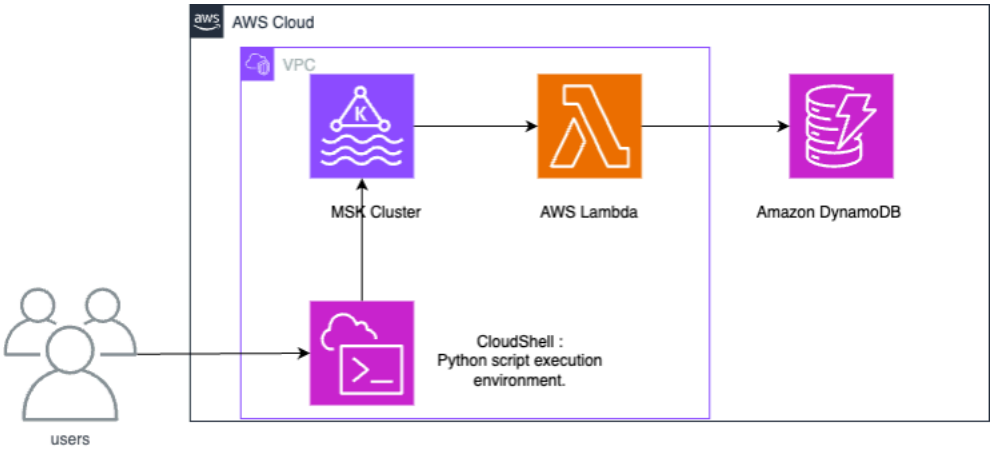
O Amazon DynamoDB é um destino comum nesses pipelines de dados para comportar aplicações que usam modelos de dados de chave-valor ou de documentos e desejam escalabilidade ilimitada com performance consistente de milissegundos de um dígito.

Tópicos

- Como funciona
- Configurar uma integração entre o Amazon MSK e o DynamoDB
- Próximas etapas

Como funciona

Uma integração entre o Amazon MSK e o DynamoDB usa uma função do Lambda para consumir registros do Amazon MSK e gravá-los no DynamoDB.



O Lambda pesquisa internamente por novas mensagens do Amazon MSK e, depois, invoca de maneira síncrona a função do Lambda de destino. A carga útil de eventos da função do Lambda contém lotes de mensagens do Amazon MSK. Para a integração entre o Amazon MSK e o DynamoDB, a função do Lambda grava essas mensagens no DynamoDB.

Configurar uma integração entre o Amazon MSK e o DynamoDB

> **Note**
>
> É possível baixar os recursos usados neste exemplo no repositório do GitHub a seguir.

As etapas abaixo mostram como configurar uma integração de exemplo entre o Amazon MSK e o Amazon DynamoDB. O exemplo representa dados gerados por dispositivos da Internet das Coisas (IoT) e ingeridos no Amazon MSK. À medida que os dados são ingeridos no Amazon MSK, eles podem ser integrados a serviços de analytics ou ferramentas de terceiros compatíveis com o Apache Kafka, possibilitando vários casos de uso de analytics. A integração do DynamoDB também fornece uma pesquisa de chave-valor de registros de dispositivos individuais.

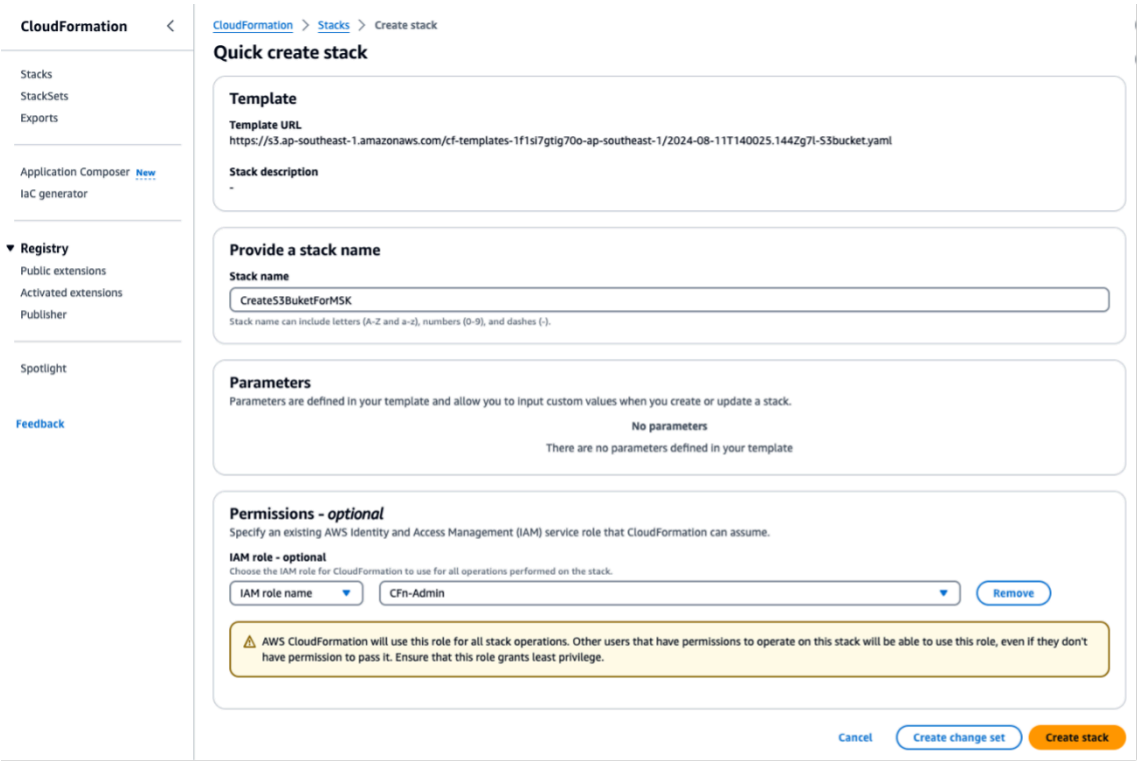
Este exemplo demonstrará como um script Python grava dados de sensores de IoT no Amazon MSK. Depois, uma função do Lambda grava itens com a chave de partição “`deviceid`” no DynamoDB.

O modelo fornecido do CloudFormation criará os seguintes recursos: um bucket do Amazon S3, uma Amazon VPC, um cluster do Amazon MSK e um AWS CloudShell para testar operações de dados.

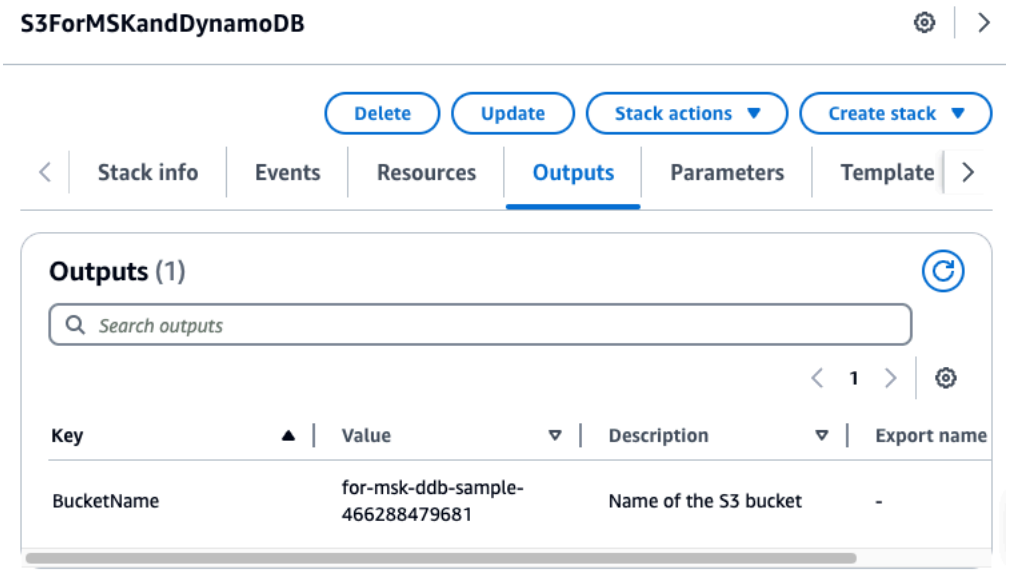
Para gerar dados de teste, crie um tópico do Amazon MSK e, depois, crie uma tabela do DynamoDB. É possível usar o Session Manager no console de gerenciamento para fazer login no sistema operacional do CloudShell e executar scripts Python.

Depois que você executar o modelo do CloudFormation, poderá concluir a criação dessa arquitetura executando as operações a seguir.

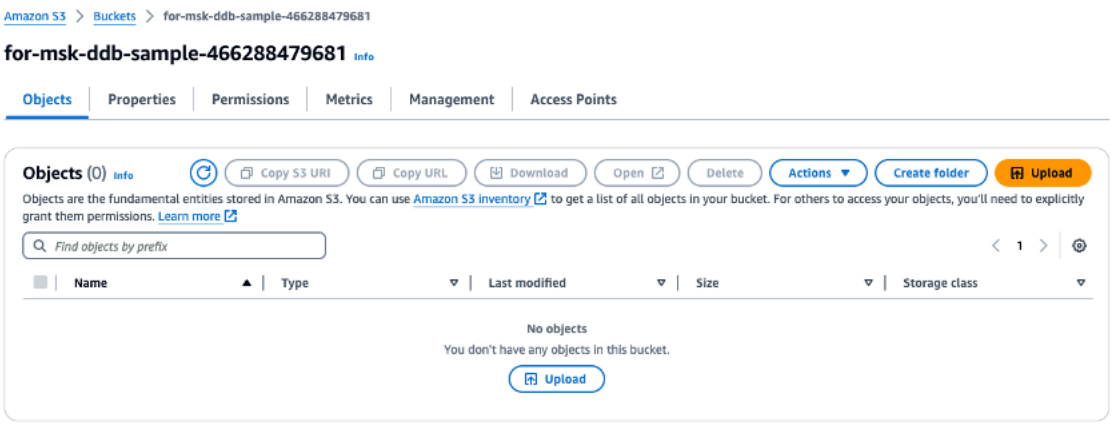
1. Execute o modelo do CloudFormation `S3bucket.yaml` para criar um bucket do S3.
   Para quaisquer scripts ou operações subsequentes, execute-os na mesma região. Insira `ForMSKTestS3` como o nome da pilha do CloudFormation.



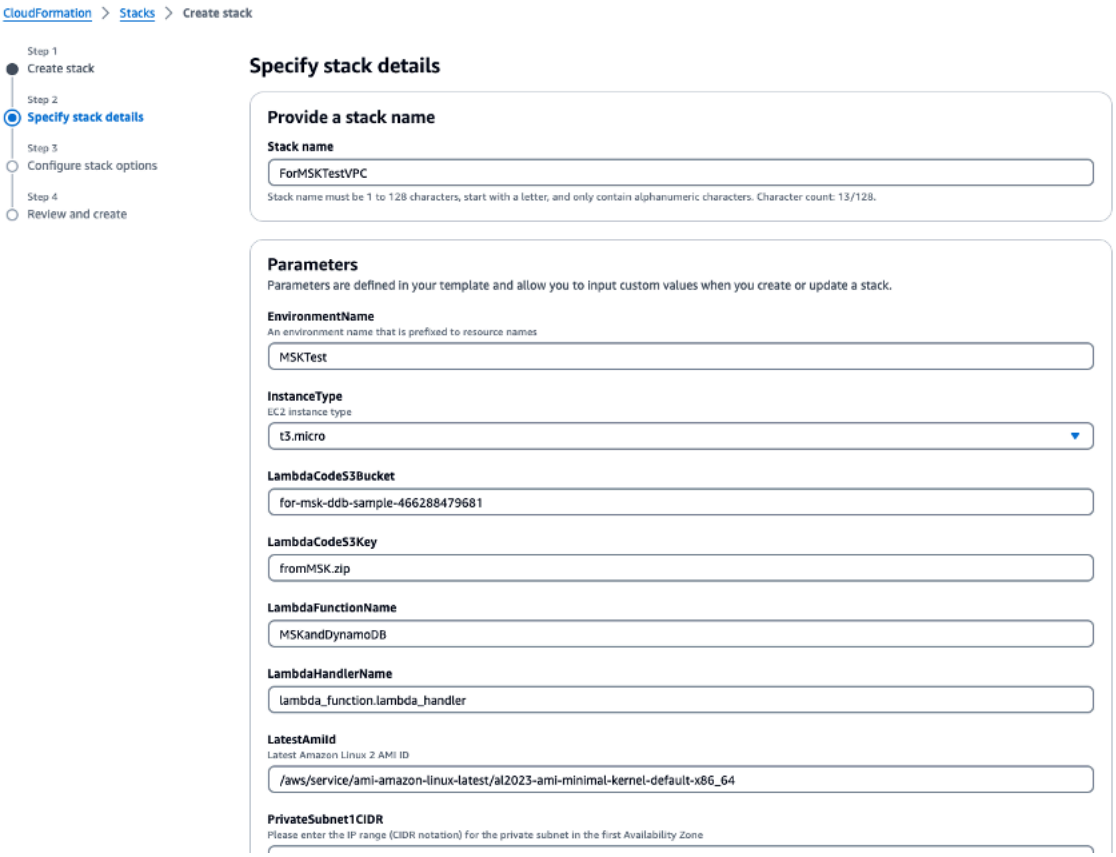
Depois que esse processo for concluído, anote a saída do nome do bucket do S3 em Saídas. Você precisará do nome na Etapa 3.



2. Faça upload do arquivo ZIP `fromMSK.zip` no bucket do S3 recém-criado.



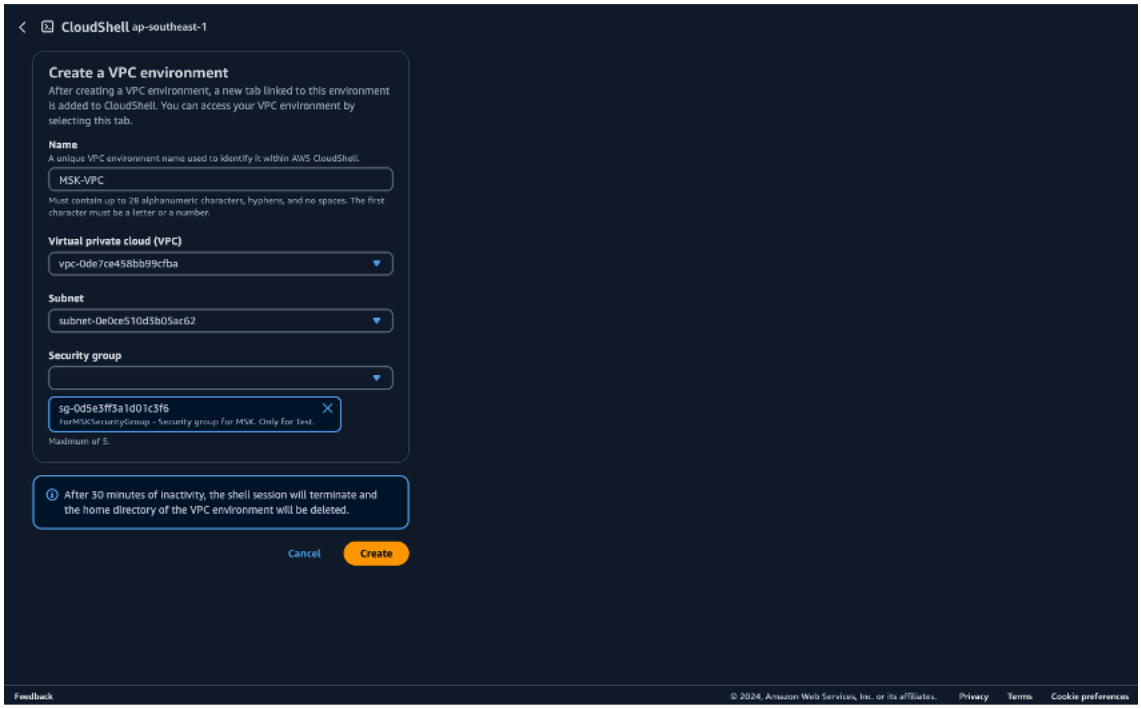
3. Execute o modelo do CloudFormation `VPC.yaml` para criar uma VPC, um cluster do Amazon MSK e uma função do Lambda. Na tela de entrada de parâmetros, insira o nome do bucket do S3 que você criou na Etapa 1, onde o bucket do S3 é solicitado. Defina o nome da pilha do CloudFormation como `ForMSKTestVPC`.



4. Prepare o ambiente para executar scripts Python no CloudShell. É possível usar o CloudShell no AWS Management Console. Para ter mais informações sobre o uso do CloudShell, consulte Getting started with AWS CloudShell. Depois que você iniciar o CloudShell, crie um CloudShell

pertencente à VPC que você acabou de criar para se conectar ao cluster do Amazon MSK. Crie o CloudShell em uma sub-rede privada. Preencha os seguintes campos:

1. Nome: pode ser definido como qualquer nome. Um exemplo é MSK-VPC
2. VPC: selecione MSKTest
3. Sub-rede: selecione Sub-rede privada MSKTest (AZ1)
4. SecurityGroup: selecione ForMSKSecurityGroup



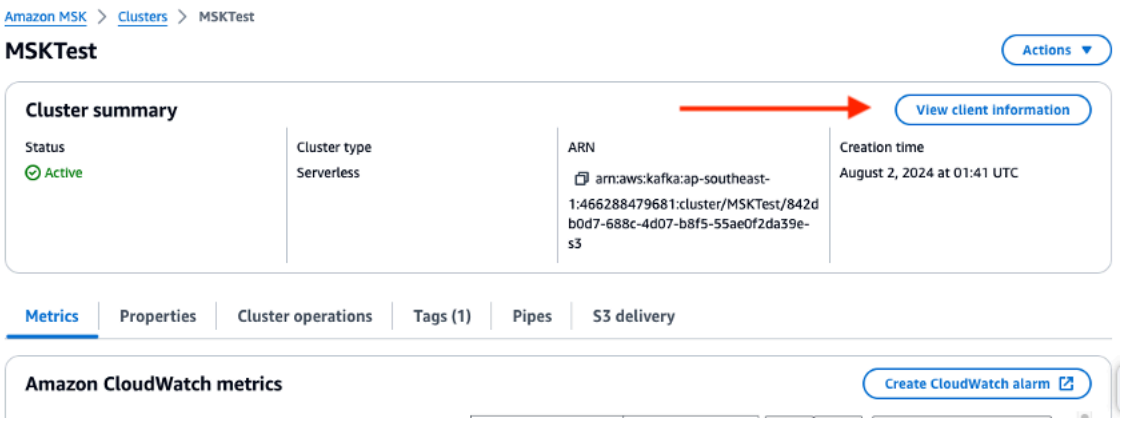
Quando o CloudShell pertencente à sub-rede privada for iniciado, execute o seguinte comando:

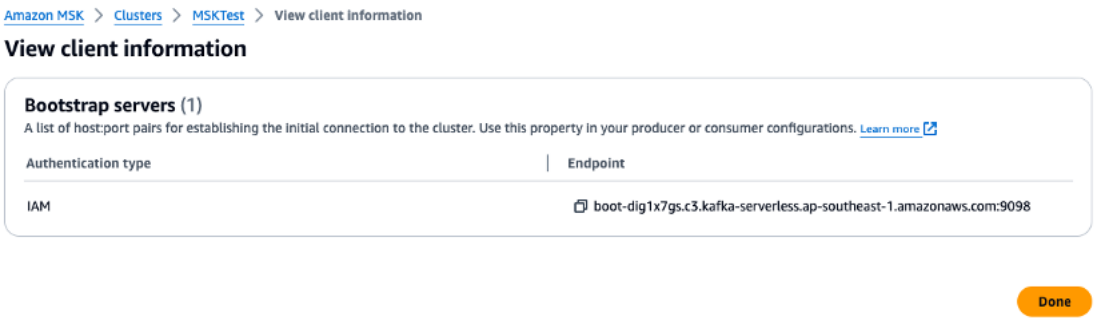
```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. Baixe scripts Python do bucket do S3.

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. Confira o console de gerenciamento e defina as variáveis de ambiente para o URL do agente e o valor da região nos scripts Python. Confira o endpoint do agente do cluster Amazon MSK no console de gerenciamento.





7. Defina as variáveis de ambiente no CloudShell. Se você estiver usando a região do Oeste dos EUA (Oregon):

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. Execute os scripts Python a seguir.

   Crie um tópico do Amazon MSK:

```
python ./createTopic.py
```

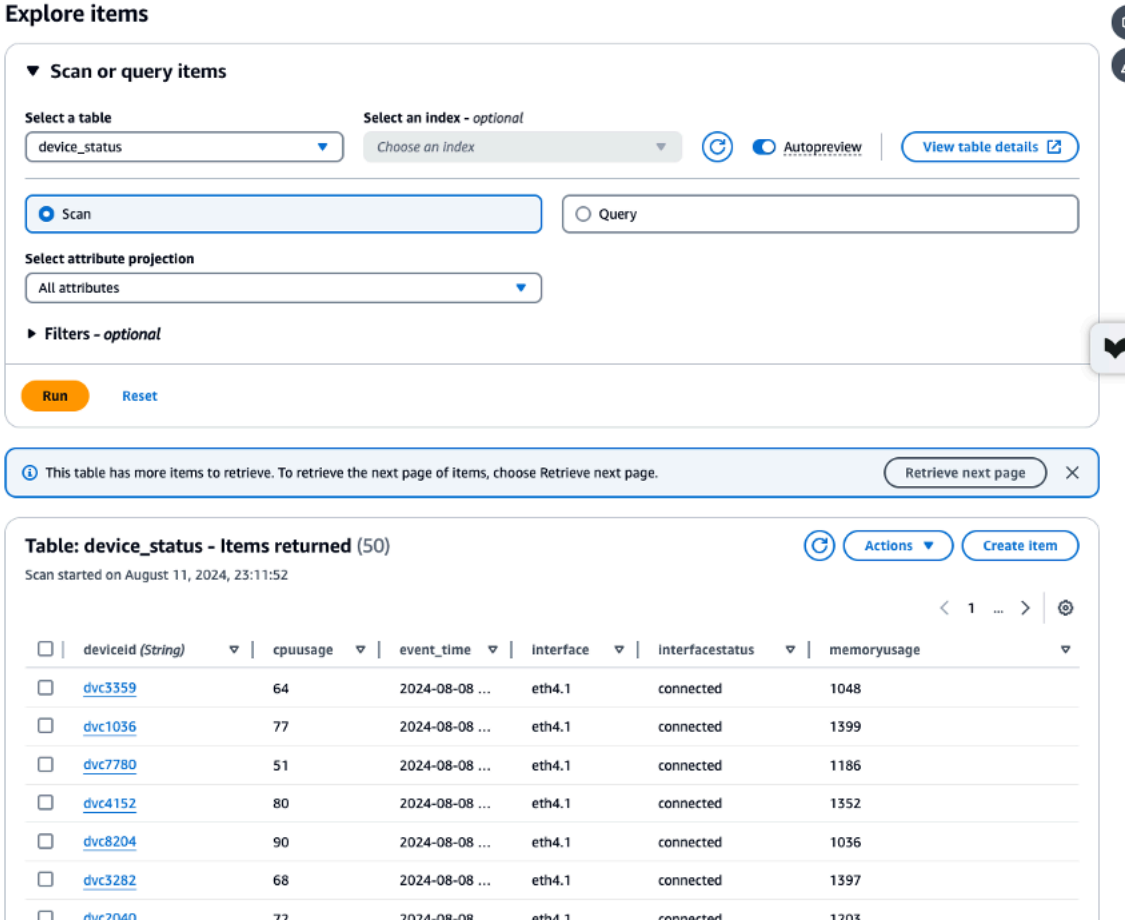
   Crie uma tabela do DynamoDB:

```
python ./createTable.py
```

   Grave os dados de teste no tópico do Amazon MSK:

```
python ./kafkaDataGen.py
```

9. Confira as métricas do CloudWatch para os recursos criados do Amazon MSK, do Lambda e do DynamoDB e verifique os dados armazenados na tabela `device_status` usando o DynamoDB Data Explorer para garantir que todos os processos tenham sido executados corretamente. Se cada processo for executado sem erros, você poderá conferir se os dados de teste gravados do CloudShell no Amazon MSK também foram gravados no DynamoDB.



10. Quando terminar esse exemplo, exclua os recursos criados neste tutorial. Exclua as duas pilhas do CloudFormation: `ForMSKTestS3` e `ForMSKTestVPC`. Se a exclusão das pilhas for concluída com êxito, todos os recursos serão excluídos.

Próximas etapas

> **Note**
>
> Se você criou recursos ao seguir este exemplo, lembre-se de excluí-los para evitar cobranças inesperadas.

A integração identificou uma arquitetura que vincula o Amazon MSK e o DynamoDB para permitir que dados de fluxos comportem workloads OLTP. A partir daqui, pesquisas mais complexas podem ser realizadas vinculando o DynamoDB ao OpenSearch Service. Pense na integração com o EventBridge para necessidades mais complexas baseadas em eventos, e extensões como o Amazon Managed Service for Apache Flink para requisitos de maior throughput e menor latência.

Práticas recomendadas para integração com o DynamoDB

Ao integrar o DynamoDB a outros serviços, é necessário sempre seguir as práticas recomendadas para usar cada serviço. No entanto, há algumas práticas recomendadas específicas de integração nas quais você deve pensar.

Tópicos

- Criar um snapshot no DynamoDB
- Capturar alterações de dados no DynamoDB

Criar um snapshot no DynamoDB

- Em geral, recomendamos o uso da exportação para o Amazon S3 para criar snapshots da replicação inicial. É econômico e não concorrerá com o tráfego da aplicação em termos de throughput. Também é possível pensar em backup e restauração em uma nova tabela, seguidos de uma operação de verificação. Isso evitará a disputa por throughput com sua aplicação, mas geralmente será consideravelmente menos econômico do que uma exportação.
- Sempre defina `StartTime` ao fazer uma exportação. Isso facilita a determinação de onde você iniciará a captura de dados de alteração (CDC).
- Ao usar a exportação para o S3, defina uma ação de ciclo de vida no bucket do S3. Normalmente, uma ação de expiração definida em sete dias é segura, mas é necessário seguir todas as diretrizes que a empresa possa ter. Mesmo que você exclua explicitamente os itens após a ingestão, essa ação pode ajudar a detectar problemas e, portanto, reduzir custos desnecessários e evitar violações de política.

Capturar alterações de dados no DynamoDB

- Se você precisar de uma CDC quase em tempo real, use o DynamoDB Streams ou o Amazon Kinesis Data Streams (KDS). Ao decidir qual deles usar, procure sempre pensar em qual é o mais fácil de usar com o serviço subsequente. Se você precisar fornecer processamento de eventos em ordem em nível de chave de partição ou se tiver itens excepcionalmente grandes, use o DynamoDB Streams.
- Se não precisar de CDC quase em tempo real, poderá usar a exportação para o Amazon S3 com exportações incrementais para exportar somente as alterações que ocorreram entre dois momentos.

Se você usou a exportação para o S3 para gerar um snapshot, isso pode ser especialmente útil porque é possível usar um código semelhante para processar exportações incrementais. Normalmente, a exportação para o S3 é um pouco mais barata do que as opções de streaming anteriores, mas o custo normalmente não é o principal fator para a opção a ser usada.

- Geralmente, só é possível ter dois consumidores simultâneos de um fluxo do DynamoDB. Pense nisso ao planejar a estratégia de integração.
- Não use verificações para detectar alterações. Isso pode funcionar em pequena escala, mas se torna impraticável de modo razoavelmente rápido.

Usar IA generativa com o DynamoDB

O Amazon DynamoDB é um banco de dados sem servidor, NoSQL e totalmente gerenciado com desempenho de latência inferior a dez milissegundos em qualquer escala. O DynamoDB é otimizado para workloads de alto throughput e você pode ampliar os recursos por meio da integração com modelos de IA generativa. Usando modelos de IA generativa, você pode trabalhar com dados armazenados em tabelas do DynamoDB em tempo real e criar aplicações que sejam contextualmente conscientes e altamente personalizadas. Também é possível aprimorar a experiência do usuário final aproveitando totalmente seus dados de negócios, de usuários e de aplicações para personalizar suas soluções de IA generativa.

Para obter mais informações sobre a IA generativa e as soluções fornecidas pela AWS para a criação de aplicações de IA generativa, consulte Transforme sua empresa com a IA generativa.

Tópicos

- Casos de uso de IA generativa para o DynamoDB
- Blogs de IA generativa para o DynamoDB
- Como aproveitar a integração ETL zero do DynamoDB com o OpenSearch Service

Casos de uso de IA generativa para o DynamoDB

O DynamoDB é amplamente usado em aplicações conversacionais baseadas em IA, como chatbots e call centers, desenvolvidas com um modelo de base (FM). É possível acessar FMs por meio do Amazon Bedrock, do Amazon SageMaker AI ou de outros provedores de modelos. Essas aplicações geralmente usam o DynamoDB para melhorar a personalização e aprimorar a experiência do usuário em três padrões de dados: dados da aplicação, dados de negócios e dados dos usuários. Alguns exemplos desses padrões de dados são os seguintes:

- Armazenamento de dados da aplicação, como histórico de mensagens do chat, por meio de integrações com LangChain, LlamaIndex ou um código personalizado. Esse contexto aprimora a experiência do usuário, permitindo que o modelo converse com o usuário.
- Criação de uma experiência personalizada para o usuário, aproveitando dados comerciais, como inventário, preços e documentação.
- Aplicação de dados do usuário, como histórico da web, pedidos anteriores e preferências do usuário, para fornecer respostas personalizadas.

Por exemplo, uma seguradora pode criar um chatbot usando o DynamoDB para fornecer acesso aos dados quase em tempo real para um modelo de IA generativa baseado em geração aumentada via recuperação (RAG). Exemplos desses dados são taxas de hipoteca, preços de produtos, cópia de contratos padrão/em conformidade, histórico da web do usuário e preferências do usuário, tudo em tempo real. A combinação do DynamoDB com RAG adiciona informações detalhadas e atualizadas sobre produtos de seguros e os dados do usuário. Isso enriquece os prompts e as respostas para fornecer aos usuários finais uma experiência precisa, personalizada e quase em tempo real.

Da mesma forma, os clientes do setor de serviços financeiros usam o DynamoDB, as bases de conhecimento do Amazon Bedrock e os agentes do Amazon Bedrock para desenvolver aplicações de IA generativa baseadas em RAG. Essas aplicações podem usar relatórios de rendimentos de código aberto e transcrições de chamadas. Também podem usar o portfólio específico do usuário e o histórico de transações para gerar um resumo sob demanda do portfólio, incluindo uma perspectiva para o futuro.

Blogs de IA generativa para o DynamoDB

Os artigos a seguir apresentam casos de uso detalhados, práticas recomendadas e guias passo a passo para ajudar a utilizar os recursos do DynamoDB na criação de aplicações avançadas com tecnologia de IA.

- Modelos de dados do Amazon DynamoDB para chatbots de IA generativa
- Criar um chatbot escalável e sensível ao contexto com Amazon DynamoDB, Amazon Bedrock e LangChain

Como aproveitar a integração ETL zero do DynamoDB com o OpenSearch Service

Você pode usar o Amazon Bedrock com o DynamoDB para fornecer acesso sem servidor a modelos de base (FMs), como o Amazon Titan e outros modelos de terceiros. Você pode aproveitar a integração ETL zero com o Amazon OpenSearch Service para habilitar recursos de pesquisa vetorial ao desenvolver aplicações de IA generativa. O workshop Generative AI with DynamoDB zero-ETL to OpenSearch integration and Amazon Bedrock (IA generativa com a integração ETL zero do DynamoDB ao OpenSearch e Amazon Bedrock) oferece uma experiência prática na configuração da integração ETL zero do DynamoDB com o OpenSearch. Esse workshop executa as seguintes tarefas:

- Cria um pipeline de uma tabela do DynamoDB para o OpenSearch.
- Cria um conector do Amazon Bedrock no OpenSearch.
- Consulta o Amazon Bedrock usando o OpenSearch como um armazenamento vetorial.
- Usa o Claude FM no Amazon Bedrock para criar uma resposta escrita em inglês simples explicando os resultados da pesquisa retornados pelo OpenSearch.

Esse workshop permite que você integre o DynamoDB ao OpenSearch para desenvolver aplicações de IA generativa. Também demonstra a capacidade flexível de consulta em todos os mecanismos de banco de dados para ajudar você a integrar o DynamoDB ao OpenSearch para casos de uso tradicionais. Esse workshop é um dos sete módulos do Amazon DynamoDB Immersion Day (Dia de imersão no Amazon DynamoDB). Você pode realizar esse workshop em qualquer Conta da AWS.

Também pode consultar a postagem do blog a seguir sobre como configurar uma integração ETL zero entre o DynamoDB e o OpenSearch Service. Essa postagem do blog também descreve como configurar conectores de modelo no OpenSearch Service para gerar incorporações automaticamente usando o Amazon Bedrock para entrada de dados. Vector search for Amazon DynamoDB with zero ETL for Amazon OpenSearch Service (Pesquisa vetorial para o Amazon DynamoDB com ETL zero para o Amazon OpenSearch Service).

Cotas e restrições do Amazon DynamoDB

Este tópico descreve cotas atuais, antes chamadas de limites, no Amazon DynamoDB. Este tópico também descreve como realizar as tarefas de gerenciamento de cotas; por exemplo, visualizar as cotas atuais e solicitar um aumento de cota.

Tópicos

- Realizar tarefas de gerenciamento de cotas no DynamoDB
- Solicitar um aumento de cota no DynamoDB
- Cotas no Amazon DynamoDB
- Restrições no Amazon DynamoDB

Realizar tarefas de gerenciamento de cotas no DynamoDB

O Amazon DynamoDB tem vários componentes de serviço, como tabelas, fluxos, índices etc. Quando você cria a Conta da AWS, há cotas padrão (anteriormente chamadas de limites) definidas para esses componentes. A menos que especificado de outra forma, cada cota é específica da região. É possível solicitar aumento de algumas das cotas. Depois que a cota de um recurso é atingida, as solicitações adicionais para criá-lo falham, exibindo uma exceção.

Acessar as cotas do DynamoDB

É possível trabalhar com o Service Quotas do DynamoDB de qualquer uma das seguintes formas:

- AWS Management Console

  O console do Service Quotas é uma interface baseada em navegador que pode ser usada para gerenciar suas cotas de serviço. É possível acessar o Service Quotas de qualquer página do AWS Management Console escolhendo-o na barra de navegação superior ou pesquisando por Service Quotas no AWS Management Console.
- Ferramentas da AWS Command Line Interface

  Quando são usadas ferramentas da AWS Command Line Interface, é possível emitir comandos na linha de comandos do sistema para executar tarefas do Service Quotas. As ferramentas da linha de comandos serão úteis se você quiser criar scripts que realizem tarefas da AWS.
- SDKs da AWS

É possível usar os SDKs da AWS para várias linguagens de programação e plataformas (por exemplo, Java, Python, Ruby, .NET, iOS e Android, entre outras) para realizar tarefas do Service Quotas.

Se uma cota ajustável não estiver disponível no console do Service Quotas, use o AWS Support Center Console para criar um caso de aumento do Service Quotas.

Visualizar as cotas atuais no console

Como visualizar as cotas atuais do DynamoDB usando o console do Service Quotas

1. Abra o console do Service Quotas em https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/.
2. Na barra de navegação, na parte superior da tela, selecione uma região.
3. O console exibe detalhes sobre o seguinte: Nome da cota do DynamoDB, Valor da cota aplicada em nível de conta, Valor da cota padrão da AWS, Utilização e Ajustabilidade da cota em nível de conta ou de recurso.

   Se o valor da cota aplicada ou a utilização não estiverem disponíveis, o console exibirá Não disponível. É possível solicitar o valor da cota aplicada por meio do console do Support Center.
4. Escolha um nome específico em Nome da cota para visualizar a página Detalhes, que exibe Descrição, Código da cota, ARN da cota, Utilização, Valor da cota aplicada em nível de conta, Ajustabilidade e Valor da cota padrão da AWS dessa cota.

   Se aplicável, a página Detalhes também exibirá quaisquer opções de Monitoramento, Alarmes, Histórico de solicitações e todas as Tags da cota.

Visualizar as cotas atuais usando a AWS CLI

Para visualizar os valores padrão das cotas do DynamoDB:

- Chame a operação `ListDefaultServiceQuotas` com o código de serviço do DynamoDB (dynamodb) para recuperar os valores padrão do Service Quotas do Amazon DynamoDB.

```
$ aws service-quotas list-aws-default-service-quotas \
        --service-code dynamodb
```

```
 {
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        },
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-AB614373",
            "QuotaCode": "L-AB614373",
            "QuotaName": "Table-level write throughput limit",
            "Value": 40000.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        }......
    ]
}
```

Para visualizar os valores das cotas aplicadas:

- Chame a operação `ListServiceQuotas` com o código de serviço do DynamoDB (dynamodb) para recuperar todos os valores da cota aplicada em nível de conta, em nível de recurso ou em todos os níveis, passando `ACCOUNT`, `RESOURCE` ou `ALL`, respectivamente, como o valor do parâmetro `QuotaAppliedAtLevel`. O exemplo da CLI a seguir recupera valores da cota aplicada em nível de conta.

```
$ aws service-quotas list-service-quotas \
        --service-code dynamodb \
        --quota-applied-at-level ACCOUNT


{
```

```
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L-
F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L
-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false,
            "QuotaAppliedAtLevel": "ACCOUNT"
        }.....

        }
    ]
}
```

Solicitar um aumento de cota no DynamoDB

É possível solicitar um aumento de cota para cada região usando o console do Service Quotas, a AWS CLI ou um caso de suporte. Se uma cota ajustável não estiver disponível no console do Service Quotas, use o AWS Support Center Console para criar um caso de aumento do Service Quotas.

O Suporte pode aprovar, negar ou aprovar parcialmente solicitações de aumento de cota. Os aumentos não são concedidos imediatamente e podem levar alguns dias para entrar em vigor.

Para solicitar um aumento, visite o Console do Service Quotas

1. Abra o console do Service Quotas em https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Na barra de navegação, na parte superior da tela, selecione uma região.
3. Filtre a lista por nome de recurso. Por exemplo, insira Sob demanda para localizar as cotas referentes a instâncias sob demanda.
4. Se a cota for ajustável, escolha a cota e, em seguida, escolha Solicitar aumento de cota.
5. Em Alterar valor da cota, insira o novo valor da cota.
6. Escolha Solicitar.
7. Para visualizar quaisquer solicitações pendentes ou resolvidas recentemente no console, escolha Painel no painel de navegação. Para solicitações pendentes, escolha o status da solicitação para abrir o recibo da solicitação. O status inicial de uma solicitação é Pending (Pendente). Depois que o status for alterado para Cota solicitada, você verá o número do caso com o Suporte. Escolha o número do caso para abrir o tíquete de sua solicitação.

Para obter mais informações, incluindo como usar a AWS CLI ou os SDKs para solicitar um aumento de cota, consulte Solicitação de aumento de cota no Guia do usuário do Service Quotas.

Cotas no Amazon DynamoDB

Esta seção descreve cotas atuais, antes chamadas de limites, no Amazon DynamoDB. Salvo indicação em contrário, cada cota aplica-se por região.

Tópicos

- Throughput de leitura e gravação
- Capacidade reservada
- Tabelas
- Tabelas globais
- Índices secundários
- Atributos projetados do índice secundário
- DynamoDB Streams
- Importar do Amazon S3
- Exportação de tabela para o Amazon S3

- Backup e restauração
- Contributor Insights

Throughput de leitura e gravação

Cotas padrão de throughput

A AWS aplica algumas cotas padrão ao throughput que sua conta pode provisionar e consumir em uma região.

O throughput de leitura no nível da conta e as cotas de throughput de gravação no nível da conta se aplicam no nível da conta. Essas cotas no nível da conta se aplicam à soma da capacidade de throughput provisionada para todos os índices secundários globais e tabelas da conta em determinada região. Todo throughput disponível da conta pode ser provisionado para uma única tabela ou entre várias tabelas. Essas cotas só se aplicam a tabelas que usam o modo de capacidade provisionada.

As cotas de throughput de leitura e de gravação no nível da tabela se aplicam de forma diferente a tabelas que usam o modo de capacidade provisionada e tabelas que usam o modo de capacidade sob demanda.

Para tabelas no modo de capacidade provisionada e GSIs, a cota é a quantidade máxima de unidades de capacidade de leitura e gravação que podem ser provisionadas para qualquer tabela ou qualquer um de seus GSIs na região. O total de qualquer tabela individual e de todos os respectivos GSIs também deve permanecer abaixo da cota de throughput de leitura e gravação no nível da conta. Isso se soma à exigência de que o total de todas as tabelas provisionadas e os respectivos GSIs permaneçam abaixo da cota de throughput de leitura e gravação no nível da conta.

Para tabelas no modo de capacidade sob demanda e GSIs, a cota em nível de tabela é a capacidade máxima de leitura e gravação que estão disponíveis para qualquer tabela ou qualquer GSI individual nessa tabela. Nenhuma cota de throughput de leitura e gravação no nível de conta é aplicada às tabelas no modo sob demanda.

Veja a seguir as cotas de throughput que se aplicam à sua conta, por padrão.

> **Note**
>
> É possível solicitar qualquer número de unidades de capacidade de leitura (RCUs) ou de unidades de capacidade de gravação (WCUs) para tabelas do DynamoDB aumentando a

cota de serviço. Os valores listados na tabela a seguir representam as cotas padrão iniciais. Eles não são limites máximos para as tabelas.

Nome da cota de throughput	Sob demanda	Provisionada	Ajustável
Por tabela	40 mil unidades de solicitação de leitura e 40 mil unidades de solicitação de gravação	40.000 unidades de capacidade de leitura e 40.000 unidades de capacidade de gravação	Sim
Por conta	Não aplicável	80.000 unidades de capacidade de leitura e 80.000 unidades de capacidade de gravação	Sim
Throughput mínimo para qualquer tabela ou índice secundário global	Não aplicável	1 unidade de capacidade de leitura e 1 unidade de capacidade de gravação	Sim

Aumentar ou diminuir throughput (para tabelas provisionadas)

Aumentar throughput provisionado

Você pode aumentar as `ReadCapacityUnits` ou as `WriteCapacityUnits` sempre que necessário, usando o AWS Management Console ou a operação `UpdateTable`. Em uma única chamada, é possível aumentar o throughput provisionado de uma tabela, para quaisquer índices secundários globais na tabela ou para qualquer combinação dessas. As novas configurações não têm efeito até que a operação `UpdateTable` esteja concluída.

Você não pode exceder as cotas por conta ao adicionar a capacidade provisionada, e o DynamoDB não permite aumentar a capacidade provisionada de forma extremamente rápida. Além dessas

restrições, você pode aumentar a capacidade provisionada de suas tabelas para o nível que for necessário. Para mais informações sobre cotas por conta, consulte a seção anterior, Cotas padrão de throughput.

Diminuir throughput provisionado

Para cada tabela e índice secundário global em uma operação `UpdateTable`, é possível diminuir `ReadCapacityUnits` ou `WriteCapacityUnits` (ou ambas). As novas configurações não têm efeito até que a operação `UpdateTable` seja concluída.

Há uma cota padrão sobre o número de reduções da capacidade provisionada que você pode executar na tabela do DynamoDB por dia. Um dia é definido de acordo com o Tempo Universal Coordenado (UTC). Em determinado dia, você pode começar realizando até quatro reduções dentro de uma hora, desde que ainda não tenha realizado nenhuma outra redução durante esse dia. Posteriormente, é possível realizar uma redução adicional por hora (uma vez a cada sessenta minutos). Isso efetivamente eleva o número máximo de reduções em um dia para 27 vezes.

> **Important**
>
> Como os limites de diminuição de tabela e índice secundário global são separados, qualquer índice secundário global de uma tabela específica tem os próprios limites de diminuição. No entanto, se uma única solicitação diminuir o throughput de uma tabela e um índice secundário global, ela será rejeitada se exceder os limites atuais. Solicitações não são processadas parcialmente.

Example

Nas primeiras 4 horas de um dia, uma tabela com um índice secundário global pode ser modificada da seguinte forma:

- Diminuir `WriteCapacityUnits` ou `ReadCapacityUnits` (ou ambos) da tabela quatro vezes.
- Diminuir `WriteCapacityUnits` ou `ReadCapacityUnits` (ou ambos) do índice secundário global quatro vezes.

Ao final do mesmo dia, a tabela e o throughput do índice secundário global poderão ser diminuídas 27 vezes cada, no total.

Capacidade reservada

A AWS coloca uma cota padrão na quantidade de capacidade reservada ativa que sua conta pode comprar. O limite de cota é uma combinação de capacidade reservada para unidades de capacidade de gravação (WCUs) e unidades de capacidade de leitura (RCUs).

Cota de capacidade reservada	Capacidade reservada ativa	Ajustável
Por conta	1.000.000 de unidades de capacidade provisionada (WCUs _ RCUs)	Sim

Se você tentar comprar mais de 1.000.000 de unidades de capacidade provisionada em uma única compra, receberá um erro para esse limite de cota de serviço. Se você tiver capacidade reservada ativa e tentar comprar capacidade reservada adicional que resultaria em mais de 1.000.000 de unidades de capacidade provisionada ativas, receberá um erro para esse limite de cota de serviço.

Tabelas

Tamanho da tabela

Não há limite prático para o tamanho de uma tabela. As tabelas não são limitadas em termos de número de itens ou de bytes.

Número máximo de tabelas por conta por região

Para qualquer conta da AWS, há uma cota inicial de 2.500 tabelas por região da AWS.

Se você precisar de mais de 2,5 mil tabelas para uma única conta, entre em contato com a equipe de contas da AWS para examinar um aumento de até 10 mil tabelas. Para mais de 10 mil, a prática recomendada é configurar várias contas. Cada uma pode atender a até 10 mil tabelas.

Tabelas globais

As cotas padrão a seguir se aplicam ao usar tabelas globais.

Cotas padrão da tabela global	Sob demanda	Provisionada
Número de tabelas globais do MRSC (consulte the section called “Modos de consistência”)	Total de 400 tabelas globais MRSC em qualquer modo de capacidade	Total de 400 tabelas globais MRSC em qualquer modo de capacidade
Taxa de transferência por tabela configurada para consistência final multirregional (MREC)	40 mil unidades de solicitação de leitura e 40 mil unidades de solicitação de gravação	40.000 unidades de capacidade de leitura e 40.000 unidades de capacidade de gravação
Taxa de transferência por tabela configurada para consistência forte multirregional (MRSC)	40 mil unidades de solicitação de leitura e 40 mil unidades de solicitação de gravação	40.000 unidades de capacidade de leitura e 40.000 unidades de capacidade de gravação
Dados preenchidos para novas réplicas por conta, por região e por dia	10 TB	10 TB

Note

Pode haver casos em que você precise solicitar um aumento do limite de cota pelo AWS Support. Se alguma das seguintes situações se aplicar a você, consulte https://aws.amazon.com/support:

- As cotas de taxa de transferência da tabela global devem ser iguais ou maiores que as cotas de taxa de transferência por tabela para que a criação da réplica seja bem-sucedida. Existem cotas de taxa de transferência de tabelas globais separadas para tabelas globais MREC e MRSC.
- Se você adicionar uma ou mais réplicas a uma região de destino num período de 24 horas com um total combinado superior a 10 TB, será necessário solicitar um aumento de cota de serviço para a cota de provisionamento de dados de adição de réplica.
- Se encontrar um erro semelhante ao seguinte:

- Não é possível criar uma réplica da tabela “example_table” na região “example_region_A” porque ela excede o limite da sua conta atual na região “example_region_B”.

Índices secundários

É possível definir até cinco índices secundários locais por tabela.

Há uma cota padrão de 20 índices secundários globais por tabela.

Atributos projetados do índice secundário

É possível projetar um total de até cem atributos em todos os índices secundários locais e globais de uma tabela. Essa cota se aplica somente a atributos projetados especificados pelo usuário.

Em uma operação `CreateTable`, se você especificar `ProjectionType` como `INCLUDE`, a contagem total de atributos especificados em `NonKeyAttributes`, somados entre todos os índices secundários, não deverá ser superior a cem. Quando o mesmo nome de atributo é projetado em dois índices diferentes, dois atributos distintos são contabilizados na cota.

Esse limite não se aplica a índices secundários com `ProjectionType` definido como `KEYS_ONLY` ou `ALL`.

DynamoDB Streams

Leitores simultâneos de um fragmento no DynamoDB Streams

Em relação a tabelas de região única que não são tabelas globais, é possível projetar até dois processos simultâneos para ler o mesmo fragmento do DynamoDB Streams. Exceder esse limite pode resultar em controle de utilização de solicitação. Quanto a tabelas globais, recomendamos que você limite o número de leitores simultâneos a 1 para evitar o controle de utilização de solicitações.

Capacidade de gravação máxima para uma tabela com o DynamoDB Streams habilitado

A AWS impõe algumas cotas padrão sobre a capacidade de gravação em tabelas do DynamoDB com o DynamoDB Streams habilitado. Essas cotas padrão são aplicáveis somente para tabelas no modo de capacidade de leitura/gravação provisionada.

- Por tabela: 40.000 unidades de capacidade de gravação

Importar do Amazon S3

A importação do Amazon S3 para o DynamoDB pode comportar até 50 trabalhos de importação simultâneos com um tamanho total de objeto de origem de importação de 15 TB por vez nas regiões us-east-1, us-west-2 e eu-west-1. Em todas as outras regiões, podem ser feitas até 50 tarefas de importação simultâneas com um tamanho total de 1 TB. Cada trabalho de importação pode ter até 50 mil objetos do Amazon S3 em todas as regiões. Para obter mais informações sobre importação e validação, consulte Importar cotas e validação de formatos.

Exportação de tabela para o Amazon S3

Exportação completa: até 300 tarefas simultâneas, ou até um total de 100 TB de todas as exportações de tabela em andamento, podem ser exportadas. Esses dois limites são conferidos antes que uma exportação seja colocada na fila.

Exportação incremental: a exportação incremental do DynamoDB para o Amazon S3 pode comportar até trezentas tarefas de exportação simultâneas ou até um total de 100 TB de todas as exportações de tabela em andamento. Os limites da janela do período de exportação são no mínimo 15 minutos e no máximo 24 horas.

Backup e restauração

O DynamoDB aceita até cinquenta restaurações simultâneas, com um total de 50 TB, por meio de backups sob demanda ou contínuos do DynamoDB. O AWS Backup comporta até cinquenta restaurações simultâneas, com um total de 25 TB.

Contributor Insights

Ao habilitar o Customer Insights na tabela do DynamoDB, observe que os limites das regras do Contributor Insights continuam aplicáveis. Para obter mais informações, consulte Cotas de serviço do CloudWatch.

Restrições no Amazon DynamoDB

Esta seção descreve restrições atuais, antes chamadas de limites, no Amazon DynamoDB.

Tópicos

- Modo de capacidade de leitura/gravação
- Índices secundários
- Chaves de partição e chaves de classificação
- Regras de nomenclatura
- Tipos de dados
- Itens
- Atributos
- Parâmetros de expressão
- Transações do DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)
- Restrições específicas da API
- Criptografia em repouso do DynamoDB

Modo de capacidade de leitura/gravação

É possível alternar as tabelas do modo de capacidade provisionada para o modo sob demanda até quatro vezes em um período contínuo de 24 horas. É possível alternar as tabelas do modo sob demanda para o modo de capacidade provisionada a qualquer momento.

Para ter mais informações sobre como alternar entre os modos de capacidade de leitura e de gravação, consulte Considerações ao alternar os modos de capacidade no DynamoDB.

Tamanhos de unidade de capacidade (para tabelas provisionadas)

Uma unidade de capacidade de leitura = uma leitura altamente consistente por segundo, ou duas leituras finais consistentes por segundo, para itens até 4 KB de tamanho.

Uma unidade de capacidade de gravação = uma gravação por segundo para itens de até 1 KB de tamanho.

As solicitações de leitura transacional exigem duas unidades de capacidade de leitura para executar uma leitura por segundo para itens de até 4 KB.

As solicitações de gravação transacional exigem duas unidades de capacidade de gravação para executar uma gravação por segundo para itens de até 1 KB.

Tamanhos de unidade de solicitação (para tabelas sob demanda)

Uma unidade de solicitação de leitura = uma leitura altamente consistente por segundo, ou duas leituras finais consistentes por segundo, para itens até 4 KB de tamanho.

Uma unidade de solicitação de gravação = uma gravação por segundo para itens com até 1 KB de tamanho.

As solicitações de leitura transacionais exigem duas unidades de solicitação de leitura para executar uma leitura por segundo para itens de até 4 KB.

As solicitações de gravação transacionais exigem duas unidades de solicitação de gravação para executar uma gravação por segundo para itens de até 1 KB.

Índices secundários

Atributos do índice secundário projetados por tabela

Você pode projetar um total de até 100 atributos em todos os índices secundários locais e globais de uma tabela. Isso se aplica somente a atributos projetados especificados pelo usuário.

Em uma operação `CreateTable`, se você especificar um `ProjectionType` de `INCLUDE`, a contagem total de atributos especificados em`NonKeyAttributes`, somados entre todos os índices secundários, não deve exceder 100. Se você projetar o mesmo nome de atributo em dois índices diferentes, isso conta como dois atributos distintos ao determinar o total.

Esse limite não se aplica a índices secundários com `ProjectionType` de `KEYS_ONLY` ou `ALL`.

Chaves de partição e chaves de classificação

Tamanho da chave de partição

O tamanho mínimo de um valor de chave de partição é 1 byte. O comprimento máximo é de 2048 bytes.

Valores de chave de partição

Não há um limite prático para o número de valores de chave de partição distintos, para tabelas ou para índices secundários.

Tamanho da chave de classificação

O tamanho mínimo de um valor de chave de classificação é 1 byte. O comprimento máximo é de 1024 bytes.

Valores de chave de classificação

Em geral, não há limite prático para o número de valores de chave de classificação distintos por valor de chave de partição.

Há exceção para tabelas com índices secundários. Coleção de itens é o conjunto de itens que têm o mesmo valor do atributo de chave de partição. Em um índice secundário global, a coleção de itens é independente da tabela-base (e pode ter um atributo de chave de partição diferente). Entretanto, em um índice secundário local, a exibição indexada é colocada na mesma partição do item na tabela e compartilha o mesmo atributo de chave de partição. Em decorrência dessa localidade, quando uma tabela tem um ou mais LSIs, a coleção de itens não pode ser distribuída por várias partições.

Para uma tabela com um ou mais LSIs, as coleções de itens não podem exceder 10 GB de tamanho. Inclui todos os itens da tabela de base e todas as visualizações de LSI projetadas que têm o mesmo valor do atributo de chave de partição. O tamanho máximo de uma partição é 10 GB. Para obter mais informações detalhadas, consulte Limite de tamanho de conjunto de itens.

Regras de nomenclatura

Nomes de tabela e nomes de índices secundários

Nomes de tabelas e índices secundários devem ter pelo menos 3 caracteres, mas não podem ser maiores do que 255 caracteres. Veja a seguir os caracteres permitidos:

- `A-Z`
- `a-z`
- `0-9`
- `_` (sublinhado)
- `-` (hífen)
- `.` (ponto)

Nomes de atributo

Em geral, um nome de atributo deve ter pelo menos 64 caracteres, mas não pode ser maior do que 64 KB.

Veja as exceções a seguir. Estes nomes de atributo não devem ser maiores do que 255 caracteres:

- Nomes de chaves de partição de índices secundários.
- Nomes de chaves de classificação de índices secundários.
- Os nomes de atributos projetados especificados pelo usuário (aplicável apenas a índices secundários locais). Em uma operação `CreateTable`, se você especificar um `ProjectionType` como `INCLUDE`, os nomes dos atributos no parâmetro `NonKeyAttributes` terão tamanho restrito. Os tipos de projeção `KEYS_ONLY` e `ALL` não são afetados.

Esses nomes de atributos devem ser codificados usando UTF-8 e o tamanho total de cada nome (após a codificação) não pode exceder 255 bytes.

Tipos de dados

String

O tamanho de uma string é restrito pelo tamanho de item máximo de 400 KB.

Strings são Unicode com codificação binária UTF-8. Como a codificação UTF-8 tem largura variável, o DynamoDB determina o tamanho de uma string usando seus bytes UTF-8.

Número

Um número pode ter até 38 dígitos de precisão, e pode ser positivo, negativo ou zero.

- Intervalo positivo: `1E-130` a `9.99999999999999999999999999999999999E+125`
- Intervalo negativo: `-9.99999999999999999999999999999999999E+125` a `-1E-130`

O DynamoDB usa strings JSON para representar dados de números em solicitações e respostas. Para obter mais informações, consulte API de baixo nível do DynamoDB.

Se a precisão de número for importante, você deverá passar os números para o DynamoDB usando strings convertidas por você de um tipo de número.

Binário

O tamanho de um binário é restrito pelo tamanho de item máximo de 400 KB.

As aplicações que funcionam com atributos binários devem codificar os dados em formato base64 antes de enviá-los para o DynamoDB. Após o recebimento dos dados, o DynamoDB os decodifica em uma matriz de bytes não assinados e usa esse resultado como o tamanho do atributo.

Itens

Tamanho de item

O tamanho de item máximo no DynamoDB é 400 KB, o que inclui o tamanho binário do nome do atributo (tamanho UTF-8) e os tamanhos dos valores dos atributos (novamente, tamanho binário). O nome do atributo conta para o limite de tamanho.

Por exemplo, considere um item com dois atributos: um atributo chamado “cor-camisa” com o valor “R” e outro atributo chamado “tamanho-camisa” com o valor “M”. O tamanho total desse item é 23 bytes.

Tamanho de item para tabelas com índices secundários locais

Para cada índice secundário local em uma tabela, existe um limite de 400 KB no total do seguinte:

- O tamanho dos dados de um item na tabela.
- O tamanho das entradas correspondentes (inclusive os valores de chaves e atributos projetados) em todos os índices secundários locais.

Atributos

Pares de nome-valor de atributo por item

O tamanho cumulativo de atributos por item deve se adequar ao tamanho máximo de item do DynamoDB (400 KB).

Número de valores em lista, mapa ou conjunto

Não há limite para o número de valores em uma lista, um mapa ou um conjunto, desde que o item que contém os valores permaneça no limite de tamanho de item de 400 KB.

Valores de atributo

Valores de atributos binários e de string vazios serão permitidos se o atributo não for usado como um atributo de chave para uma tabela ou um índice. Valores binários e de string vazios são permitidos dentro dos tipos de conjunto, lista e mapa. Um valor de atributo não pode ser um conjunto vazio (conjunto de strings, conjunto de número ou conjunto binário). Entretanto, Lists e Maps vazios são permitidos.

Profundidade de atributo aninhado

O DynamoDB oferece suporte a atributos aninhados com até 32 níveis de profundidade.

Parâmetros de expressão

Os parâmetros de expressão incluem `ProjectionExpression`, `ConditionExpression`, `UpdateExpression` e `FilterExpression`.

Tamanhos

O tamanho máximo de qualquer string de expressão é 4 KB. Por exemplo, o tamanho de `ConditionExpression a=b` é 3 bytes.

O tamanho máximo de qualquer nome de atributo de expressão única ou valor de atributo de expressão é 255 bytes. Por exemplo, `#name` é 5 bytes; `:val` é 4 bytes.

O tamanho máximo de todas as variáveis de substituição em uma expressão é 2 MB. Esta é a soma dos tamanhos de todos os `ExpressionAttributeNames` e `ExpressionAttributeValues`.

Operadores e operandos

O número máximo de operadores ou funções permitidos em uma `UpdateExpression` é 300. Por exemplo, a UpdateExpression `SET a = :val1 + :val2 + :val3` contém dois operadores “+“.

O número máximo de operandos do comprador `IN` é 100.

Palavras reservadas

O DynamoDB não impede uso de nomes conflitantes com palavras reservadas. (Para obter uma lista completa, consulte Palavras reservadas no DynamoDB.)

No entanto, caso use uma palavra reservada em um parâmetro de expressão, você também deverá especificar `ExpressionAttributeNames`. Para obter mais informações, consulte Nomes (aliases) de atributo de expressão no DynamoDB.

Transações do DynamoDB

As operações de API transacionais do DynamoDB têm as seguintes restrições:

- Uma transação não pode conter mais de 100 itens únicos.
- Uma transação não pode conter mais de 4 MB de dados.
- Duas ações em uma transação não podem funcionar contra o mesmo item na mesma tabela. Por exemplo, você não pode usar `ConditionCheck` e `Update` no mesmo item em uma transação.
- Uma transação não pode operar nas tabelas em mais de uma conta ou região da AWS.
- As operações transacionais fornecem garantia de atomicidade, consistência, isolamento e durabilidade (ACID) somente na região da AWS em que a gravação é originalmente realizada. As transações não são compatíveis entre regiões em tabelas globais. Por exemplo, suponha que você tenha uma tabela global com réplicas nas regiões Leste dos EUA (Ohio) e Oeste dos EUA (Oregon) e execute uma operação `TransactWriteItems` na região Leste dos EUA (Norte da Virgínia). Nesse caso, você pode observar transações parcialmente concluídas na região Oeste dos EUA (Oregon) conforme as alterações são replicadas. As alterações só são replicadas para outras regiões quando forem confirmadas na região de origem.

DynamoDB Streams

Leitores simultâneos de um fragmento no DynamoDB Streams

Para tabelas de região única que não são tabelas globais, você pode projetar até dois processos ao mesmo tempo para ler o mesmo fragmento do DynamoDB Streams. Exceder esse limite pode resultar em controle de utilização de solicitação. Quanto a tabelas globais, recomendamos que você limite o número de leitores simultâneos a 1 para evitar o controle de utilização de solicitações.

DynamoDB Accelerator (DAX)

Disponibilidade de regiões do AWS

Para obter uma lista de regiões da AWS em que o DAX está disponível, consulte DynamoDB Accelerator (DAX) na Referência geral da AWS.

Nós

Um cluster do DAX consiste em exatamente um nó primário e entre zero e dez nós de réplica de leitura.

O número total de nós (por conta da AWS) não pode exceder 50 em uma única região da AWS.

Grupos de parâmetros

Você pode criar até 20 grupos de parâmetros do DAX por região.

Grupos de sub-redes

Você pode criar até 50 grupos de sub-redes do DAX por região.

Em um grupo de sub-redes, você pode definir até 20 sub-redes.

> **Important**
>
> Um cluster do DAX comporta no máximo quinhentas tabelas do DynamoDB. Ao ultrapassar quinhentas tabelas do DynamoDB, o cluster poderá sofrer degradação na disponibilidade e na performance.

Restrições específicas da API

`CreateTable`/`UpdateTable`/`DeleteTable`/`PutResourcePolicy`/`DeleteResourcePolicy`

Em geral, é possível ter até quinhentas solicitações CreateTable, UpdateTable, DeleteTable, PutResourcePolicy e DeleteResourcePolicy sendo executadas simultaneamente em qualquer combinação. Por esse motivo, o número total de tabelas no estado `CREATING`, `UPDATING` ou `DELETING` não pode exceder 500.

É possível enviar até 2.500 solicitações por segundo de solicitações mutáveis (`CreateTable`, `DeleteTable`, `UpdateTable`, `PutResourcePolicy` e `DeleteResourcePolicy`) da API do ambiente de gerenciamento em um grupos de tabelas. No entanto, as solicitações `PutResourcePolicy` e `DeleteResourcePolicy` têm limites individuais mais baixos. Para ter mais informações, consulte os detalhes de cotas a seguir para `PutResourcePolicy` e `DeleteResourcePolicy`.

As solicitações `CreateTable` e `PutResourcePolicy` que incluam uma política baseada em recursos contarão como duas solicitações adicionais para cada KB da política. Por exemplo, uma solicitação `CreateTable` ou `PutResourcePolicy` com uma política de tamanho 5 KB contará como 11 solicitações. Uma para a solicitação `CreateTable` e 10 para a política baseada em recursos (2 x 5 KB). Da mesma forma, uma política com 20 KB de tamanho contará como 41 solicitações. Uma para a solicitação `CreateTable` e quarenta para a política baseada em recursos (2 x 20 KB).

`PutResourcePolicy`

É possível enviar até 25 solicitações de API `PutResourcePolicy` por segundo em um grupo de tabelas. Depois de uma solicitação bem-sucedida de uma tabela individual, nenhuma nova solicitação `PutResourcePolicy` é aceita nos 15 segundos seguintes.

O tamanho máximo aceito para um documento de política baseado em recursos é de 20 KB. O DynamoDB conta espaços em branco ao calcular o tamanho de uma política em relação a esse limite.

`DeleteResourcePolicy`

É possível enviar até cinquenta solicitações de API `DeleteResourcePolicy` por segundo em um grupo de tabelas. Depois de uma solicitação `PutResourcePolicy` bem-sucedida de uma tabela individual, nenhuma nova solicitação `DeleteResourcePolicy` é aceita nos 15 segundos seguintes.

`BatchGetItem`

Uma única operação `BatchGetItem` pode recuperar no máximo 100 itens. O tamanho total de todos os itens recuperados não pode exceder 16 MB.

`BatchWriteItem`

Uma única operação `BatchWriteItem` pode conter até 25 solicitações `PutItem` ou `DeleteItem`. O tamanho total de todos os itens gravados não pode exceder 16 MB.

`DescribeStream`

É possível chamar `DescribeStream` com uma taxa máxima de dez vezes por segundo.

DescribeTableReplicaAutoScaling

O método DescribeTableReplicaAutoScaling oferece suporte a apenas 10 solicitações por segundo.

DescribeLimits

DescribeLimits deve ser chamado apenas periodicamente. Você pode esperar erros de controle de utilização se chamá-lo mais de uma vez em um minuto.

DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights

DescribeContributorInsights, ListContributorInsights e UpdateContributorInsights devem ser chamados apenas periodicamente. O DynamoDB oferece suporte a cinco solicitações por segundo para cada uma dessas APIs.

DescribeTable/ListTables/GetResourcePolicy

É possível enviar até 2.500 solicitações por segundo de uma combinação de solicitações de API do ambiente de gerenciamento (DescribeTable, ListTables e GetResourcePolicy) somente leitura. A API GetResourcePolicy tem um limite individual inferior de cem solicitações por segundo.

DescribeTimeToLive

A operação DescribeTimeToLive é limitada a dez unidades de solicitação de leitura por segundo. Se você exceder esse limite, o DynamoDB exibirá um erro ThrottlingException.

Query

O conjunto de resultados de uma Query é limitado a 1 MB por chamada. Você pode usar a LastEvaluatedKey da resposta da consulta para recuperar mais resultados.

Scan

O conjunto de resultados de uma Scan é limitado a 1 MB por chamada. Você pode usar a LastEvaluatedKey da resposta da verificação para recuperar mais resultados.

`UpdateKinesisStreamingDestination`

Ao realizar operações `UpdateKinesisStreamingDestination`, é possível definir `ApproximateCreationDateTimePrecision` como um novo valor no máximo 3 vezes em um período de 24 horas.

`UpdateTableReplicaAutoScaling`

O método `UpdateTableReplicaAutoScaling` oferece suporte a apenas dez solicitações por segundo.

`UpdateTableTimeToLive`

O método `UpdateTableTimeToLive` é compatível apenas com uma solicitação para habilitar ou desabilitar `Time to Live (TTL)` por tabela especificada por hora. O processamento completo dessa alteração pode levar até uma hora. Todas as chamadas `UpdateTimeToLive` adicionais para a mesma tabela durante esse período de uma hora geram uma ValidationException.

Criptografia em repouso do DynamoDB

É possível alternar entre uma Chave pertencente à AWS, uma Chave gerenciada pela AWS e uma chave gerenciada pelo cliente até quatro vezes e a qualquer momento durante um período de 24 horas, por tabela, a partir do momento da criação da tabela. Se não houve mudança nas últimas seis horas, uma alteração adicional será permitida. Isso leva o número máximo de alterações em um dia para oito vezes (quatro alterações nas primeiras seis horas e uma alteração para cada uma das janelas de seis horas subsequentes em um dia).

É possível alternar as chaves de criptografia para usar uma Chave pertencente à AWS tantas vezes quanto necessário, mesmo que a cota acima tenha sido esgotada.

Estas são as cotas, a menos que você solicite uma quantidade maior. Para solicitar um aumento da cota de serviço, consulte https://aws.amazon.com/support.

Referência de API do DynamoDB

A Amazon DynamoDB API Reference (Referência da API do Amazon DynamoDB) contém uma lista completa de operações compatíveis com:

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)

Solução de problemas do Amazon DynamoDB

Os tópicos a seguir fornecem orientações para a solução de erros e problemas que você pode encontrar ao usar o Amazon DynamoDB. Se encontrar um problema que não esteja listado aqui, você poderá usar o botão de feedback desta página para relatá-lo.

Para obter mais orientações sobre solução de problemas e respostas a perguntas comuns de suporte, acesse a Central de Conhecimento da AWS.

Tópicos

- Solução de problemas de erros internos do servidor no Amazon DynamoDB
- Solução de problemas de latência no Amazon DynamoDB
- Solução de problemas de controle de utilização no Amazon DynamoDB

Solução de problemas de erros internos do servidor no Amazon DynamoDB

No DynamoDB, erros internos do servidor (500 erros) indicam que o serviço não consegue atender à solicitação. Esses erros podem ocorrer por vários motivos, como problemas transitórios de rede na frota, problemas de infraestrutura, problemas relacionados ao nó de armazenamento e muito mais.

Você pode encontrar alguns erros internos do servidor durante o ciclo de vida da sua tabela do DynamoDB. Isso está previsto devido à natureza distribuída do serviço e geralmente não deve ser motivo de preocupação. O DynamoDB repara e corrige automaticamente quaisquer problemas transitórios com o serviço em tempo real, sem exigir sua intervenção. No entanto, se você observar um número consistentemente alto de erros internos do servidor nas solicitações à tabela (conforme visto na métrica the section called “SystemErrors”), investigue mais a fundo.

Tópicos

- Investigar erros internos do servidor
- Minimizar o impacto dos erros internos do servidor
- Melhorar o reconhecimento operacional

Investigar erros internos do servidor

Se você encontrar erros internos do servidor na tabela do DynamoDB, considere estas opções:

1. Verifique o AWS Health Dashboard.

   Para identificar o problema, a primeira etapa é verificar o AWSService Health Dashboard e o AWS Account Health Dashboard. Esses painéis fornecem informações valiosas sobre quaisquer problemas no serviço como um todo, tabelas afetadas, problemas contínuos e a causa raiz depois que o problema é resolvido.

   A análise dos detalhes nesses painéis fornecerá uma melhor compreensão do status atual dos Serviços da AWS que você está usando e de quaisquer possíveis problemas que estejam afetando sua conta. Essas informações podem ajudar você a determinar as próximas etapas para resolver o problema e minimizar quaisquer interrupções nas operações.

2. Entre em contato com o Suporte.

   Se você observar erros prolongados e persistentes nas solicitações, isso pode indicar um problema com o serviço. Como regra geral, se você observar uma taxa geral de falhas de 1% ou mais nos últimos 15 minutos, é o momento apropriado para encaminhar o problema para a equipe do AWS Support. Consulte o Contrato de nível de serviço do Amazon DynamoDB para saber mais.

   Ao abrir um caso com a equipe do AWS Support, forneça os seguintes detalhes para ajudar a acelerar o processo de solução de problemas:

   - DDB afetado; tabelas ou índices secundários
   - Janela de tempo em que os erros foram observados
   - IDs de solicitação do DynamoDB, como `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG`, que você pode encontrar nos logs da aplicação.

   Incluir esses detalhes no caso de suporte ajudará a equipe da AWS a entender o problema e fornecer uma solução mais rápida. Se você não tiver os IDs de solicitação, ainda assim deverá registrar o caso com os outros detalhes disponíveis.

Minimizar o impacto dos erros internos do servidor

Se ocorrerem erros internos do servidor ao usar o DynamoDB, minimize o impacto deles na aplicação. Considere as seguintes práticas recomendadas:

- Use recuos e novas tentativas: os comportamentos padrão do SDK foram projetados para encontrar o equilíbrio certo para a maioria das aplicações em termos de estratégia de recuo e nova tentativa. No entanto, você pode ajustar essas configurações com base na tolerância da aplicação aos requisitos de tempo de inatividade e desempenho. Saiba mais sobre recuos e novas tentativas para entender como você pode ajustar as respectivas configurações.
- Use leituras finais consistentes: se a aplicação não exigir leituras altamente consistentes, considere usar as leituras finais consistentes. Essas leituras custam menos e têm menor probabilidade de apresentar problemas transitórios devido a erros internos do servidor, pois seriam atendidas por qualquer um dos nós de armazenamento disponíveis. Para ter mais informações, consulte Consistência de leitura do DynamoDB.

Melhorar o reconhecimento operacional

Manter a alta disponibilidade e confiabilidade das aplicações é crucial no cenário digital atual. Um aspecto importante disso é o monitoramento proativo de erros internos do servidor (ISEs) nas tabelas do DynamoDB e nos índices secundários globais (GSIs). Ao criar alarmes do CloudWatch para monitorar esses erros, você pode obter melhor reconhecimento operacional e receber alertas sobre possíveis problemas antes que eles afetem os usuários finais. Essa abordagem se alinha ao pilar de excelência operacional do AWS Well-Architected Framework, garantindo que o workload do DynamoDB seja otimizado para apresentar desempenho, segurança e confiabilidade.

Criar alarmes do CloudWatch

Você deve ter os alarmes do CloudWatch definidos nas tabelas do DynamoDB para receber notificações sobre números consistentemente altos de erros internos do servidor, em vez de observar as métricas manualmente. Isso está relacionado ao pilar de excelência operacional da estrutura Well-Architected para qualquer workload em AWS. Consulte Usar Lentes do Well-Architected para DynamoDB para otimizar uma workload do DynamoDB para saber mais sobre Well-Architecting de tabelas do DynamoDB.

Esses alarmes usam métrica (matemática) personalizada para calcular a porcentagem de solicitações com falha em uma janela de cinco minutos. A prática recomendada é configurar o alarme

para entrar no estado ALARM em que 3 pontos de dados consecutivos ultrapassam o limite de 1%, o que significa que 1% das solicitações falham em um período de 15 minutos.

O exemplo abaixo é um modelo do AWS CloudFormation que pode ajudar você a criar alarmes do CloudWatch na tabela e o GSI na tabela.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Sample template for monitoring DynamoDB
Parameters:
 DynamoDBProvisionedTableName:
    Description: Name of DynamoDB Provisioned Table to create
    Type: String
    MinLength: 3
    MaxLength: 255
    ConstraintDescription : https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/Limits.html#limits-naming-rules
  DynamoDBSNSEmail:
    Description : Email Address subscribed to newly created SNS Topic
    Type: String
    AllowedPattern: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$"
    MinLength: 1
    MaxLength: 255

Resources:
  DynamoDBMonitoringSNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      DisplayName: DynamoDB Monitoring SNS Topic
      Subscription:
        - Endpoint: !Ref DynamoDBSNSEmail
          Protocol: email
      TopicName: dynamodb-monitoring

  DynamoDBTableSystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
      AlarmName: 'DynamoDBTableSystemErrorAlarm'
      AlarmDescription: 'Alarm when system errors exceed 1% of total number of requests
 for 15 minutes'
      AlarmActions:
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
```

```
          Expression: 'm1/(m1+m2+m3)'
          Label: SystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
      EvaluationPeriods: 3
      Threshold: 1.0
      ComparisonOperator: 'GreaterThanThreshold'
  DynamoDBGSISystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
```

```
      AlarmName: 'DynamoDBGSISystemErrorAlarm'
      AlarmDescription: 'Alarm when GSI system errors exceed 2% of total number of
 requests for 15 minutes'
      AlarmActions:
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
          Expression: 'm1/(m1+m2+m3)'
          Label: GSISystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
```

```
                Value: !Ref DynamoDBProvisionedTableName
              - Name: 'GlobalSecondaryIndexName'
                Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
          Period: 300
          Stat: 'SampleCount'
          Unit: 'Count'
        ReturnData: False
    EvaluationPeriods: 3
    Threshold: 1.0
    ComparisonOperator: 'GreaterThanThreshold'
```

Solução de problemas de latência no Amazon DynamoDB

Se a workload parece estar com alta latência, você poderá analisar a métrica `SuccessfulRequestLatency` do CloudWatch e conferir a latência média e a latência mediana por meio de métricas de percentil (p50) para ver se ela está relacionada ao DynamoDB. Alguma variabilidade na `SuccessfulRequestLatency` relatada é normal, e picos ocasionais (particularmente na estatística `Maximum` e nos percentis altos) não devem ser motivo de preocupação. No entanto, se a estatística `Average` ou p50 (mediana) mostrar um aumento acentuado e persistir, você deverá conferir o AWS Service Health Dashboard e o Personal Health Dashboard para ter mais informações. Algumas causas possíveis são o tamanho do item na tabela (um item de 1 KB e um item de 400 KB variam em latência) ou o tamanho da consulta (dez itens em comparação a cem itens).

As métricas de percentil (p99, p90 etc.) podem ajudar você a entender melhor sua distribuição de latência. Por exemplo:

- p50 (mediana) mostra a latência típica da workload.
- p90 mostra que 90% das solicitações são mais rápidas do que esse valor.
- p99 ajuda a identificar o pior caso de latência que afeta 1% das solicitações.

Valores altos de p99 com valores normais de p50 podem indicar problemas esporádicos que afetam uma pequena parte das solicitações, enquanto valores de p50 consistentemente elevados podem sugerir alguma degradação da performance.

> **Note**
>
> Para analisar valores de percentis personalizados (como p99,9), você pode inserir manualmente o percentil desejado (p. ex., p99,9) no campo de estatística de métrica do CloudWatch. Isso permite avaliar as distribuições de latência que transcendem os percentis padrão listados no menu suspenso.

Algumas variações nas métricas de latência, especialmente em percentis mais altos, são esperadas e podem ser vistas como resultado de problemas transitórios de infraestrutura ou de operações em segundo plano orientadas pelo DynamoDB que ajudam a manter a alta disponibilidade e durabilidade dos dados armazenados nas tabelas do DynamoDB.

Se necessário, considere abrir um caso de suporte com o AWS Support e continue a avaliar todas as opções alternativas disponíveis para sua aplicação (como a evacuação de uma região, se você tiver uma arquitetura multirregional) de acordo com seus runbooks. É necessário registrar em log os IDs de solicitação para solicitações lentas a fim de fornecer esses IDs para AWS Support ao abrir um caso de suporte.

A métrica `SuccessfulRequestLatency` mede apenas a latência interna ao serviço do DynamoDB. A atividade do lado do cliente e os tempos de ida e volta da rede não estão incluídos. Para saber mais sobre a latência geral das chamadas do seu cliente para o serviço DynamoDB, é possível habilitar o registro de métricas de latência no AWS SDK.

> **Note**
>
> Para a maioria das operações de um único item (operações que se aplicam a um único item por meio da especificação do valor total da chave primária), o DynamoDB fornece `Average SuccessfulRequestLatency` de milissegundos de um dígito. Esse valor não inclui a sobrecarga de transporte do código do chamador que acessa o endpoint do DynamoDB. Para operações de dados com vários itens, a latência variará com base em fatores como o tamanho do conjunto de resultados, a complexidade das estruturas de dados retornadas e as expressões de condição e de filtro aplicadas. Para operações repetidas de vários itens no mesmo conjunto de dados com os mesmos parâmetros, o DynamoDB fornecerá `Average SuccessfulRequestLatency` altamente consistente.

Considere uma ou mais das seguintes estratégias para reduzir a latência:

- Reutilize conexões: as solicitações do DynamoDB são feitas por meio de uma sessão autenticada por HTTPS por padrão. Para iniciar a conexão, são necessárias várias idas e voltas, e isso toma tempo; por isso, a latência da primeira solicitação é maior do que a das solicitações subsequentes que reutilizam a conexão. As solicitações por meio de uma conexão já inicializada oferecem baixa latência consistente do DynamoDB. Para evitar a sobrecarga de latência do estabelecimento de novas conexões, é aconselhável implementar um mecanismo de "keep-alive" enviando uma solicitação `GetItem` a cada 30 segundos se nenhuma outra solicitação for feita.
- Use leituras finais consistentes: se a aplicação não exigir leituras altamente consistentes, considere usar as leituras finais consistentes comuns. As leituras finais consistentes têm um custo menor e podem vir de várias zonas de disponibilidade, permitindo a seleção de uma zona de disponibilidade na mesma localização do solicitante, o que diminui a latência. Para obter mais informações, consulte Consistência de leitura do DynamoDB.
- Implemente o hedging de solicitações: para requisitos de latência p99 muito baixos, considere a possibilidade de implementar o hedging de solicitações. Com o hedging de solicitações, se a solicitação inicial não receber uma resposta rápida o suficiente, envie uma segunda solicitação equivalente e deixe-as competir. A primeira resposta vence. Isso melhora a latência de cauda às custas de algumas solicitações extras. Você pode decidir quanto tempo esperar antes de enviar a segunda solicitação. O hedging é mais fácil para leituras. No caso de gravações, use ordenação baseada em carimbo de data/hora para garantir que as solicitações com hedging sejam tratadas como se estivessem ocorrendo no momento da primeira tentativa, evitando atualizações fora de ordem. Essa abordagem foi discutida em Timestamp writes for write hedging in Amazon DynamoDB.
- Ajuste o tempo limite da solicitação e o comportamento de repetição: o caminho do cliente até o DynamoDB atravessa muitos componentes, cada um projetado com a redundância em mente. Considere os seguintes aspectos:
  - Resiliência de rede
  - Tempos limite de pacotes TCP
  - Arquitetura distribuída do DynamoDB

  Os comportamentos padrão do SDK são otimizados para a maioria das aplicações. No entanto, você pode implementar uma estratégia para antecipar-se à falha e ajustar as configurações de tempo limite. As solicitações que demoram bem mais que o normal têm menor probabilidade de serem bem-sucedidas. Ao antecipar-se à falha e tentar novamente, você pode ter êxito rapidamente por um caminho diferente. Isso é semelhante ao hedging de solicitações, mas encerra a primeira solicitação em vez de permitir que ela prossiga.

Evite definir um valor de tempo limite muito alto. Tempos limite excessivamente baixos podem gerar problemas de disponibilidade induzidos pelo cliente. Por exemplo, um tempo limite de soquete de 50 milissegundos pode causar erros de conexão durante picos de latência da rede, como quando se chega próximo aos limites de largura de banda da instância do Amazon EC2 para tráfego de fluxo único. Avalie cuidadosamente os benefícios de um tempo limite mais baixo em relação aos possíveis riscos à disponibilidade da aplicação e prefira o hedging aos tempos limite curtos.

Para ver uma discussão útil sobre esse problema, consulte Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications.

- Reduza a distância entre o cliente e o endpoint do DynamoDB: se você tiver usuários dispersos globalmente, considere usar Tabelas globais: replicação multiativa e multirregional. Ao criar tabelas globais, é possível replicar a tabela para as regiões da AWS especificadas em que você deseja que ela esteja disponível. Você pode colocar uma cópia dos dados mais perto do usuário final para reduzir a latência de rede durante operações de leitura e gravação. Para ter mais informações sobre o uso eficaz das tabelas globais do DynamoDB, consulte Using Amazon DynamoDB global tables em “Recomendações da AWS”.
- Use o armazenamento em cache: se houver muitas operações de leitura no tráfego, considere usar um serviço de armazenamento em cache:
  - O DynamoDB Accelerator (DAX) é um cache em memória totalmente gerenciado e altamente disponível para o DynamoDB que oferece um desempenho até dez vezes melhor, de milissegundos para microssegundos, mesmo com milhões de solicitações por segundo. Para ter mais informações sobre o DAX, consulte Aceleração em memória com o DynamoDB Accelerator (DAX).
  - Amazon ElastiCache: um serviço de armazenamento em cache em memória totalmente gerenciado que pode ser integrado ao DynamoDB para melhorar o desempenho de leitura usando o padrão de carregamento lento (cache-aside). Para ter mais informações, consulte Integrating Amazon DynamoDB and Amazon ElastiCache by using read-through caching em “Recomendações da AWS”.

Solução de problemas de controle de utilização no Amazon DynamoDB

O DynamoDB implementa o controle de utilização para duas finalidades principais: manter o desempenho geral do serviço e controlar os custos. O controle de utilização serve como uma proteção intencional que evita a degradação do desempenho quando as taxas de consumo ultrapassam a capacidade ou como um mecanismo de controle de custos quando você atinge os limites máximos de throughput ou cota de serviço. Quando ocorre controle de utilização, o DynamoDB exibe exceções específicas com informações detalhadas sobre por que a solicitação sofreu controle de utilização e qual recurso foi afetado. Cada motivo de controle de utilização corresponde a métricas específicas do CloudWatch que fornecem insights adicionais sobre a frequência e os padrões dos eventos de controle de utilização.

O seguinte diagrama mostra os quatro cenários principais em que o DynamoDB implementa o controle de utilização de proteção:

1. Intervalo de chaves de throughput excedido (em ambos os modos):

   O consumo direcionado a partições específicas excede os limites internos de throughput em nível de partição.

2. Throughput provisionado excedido (no modo provisionado):

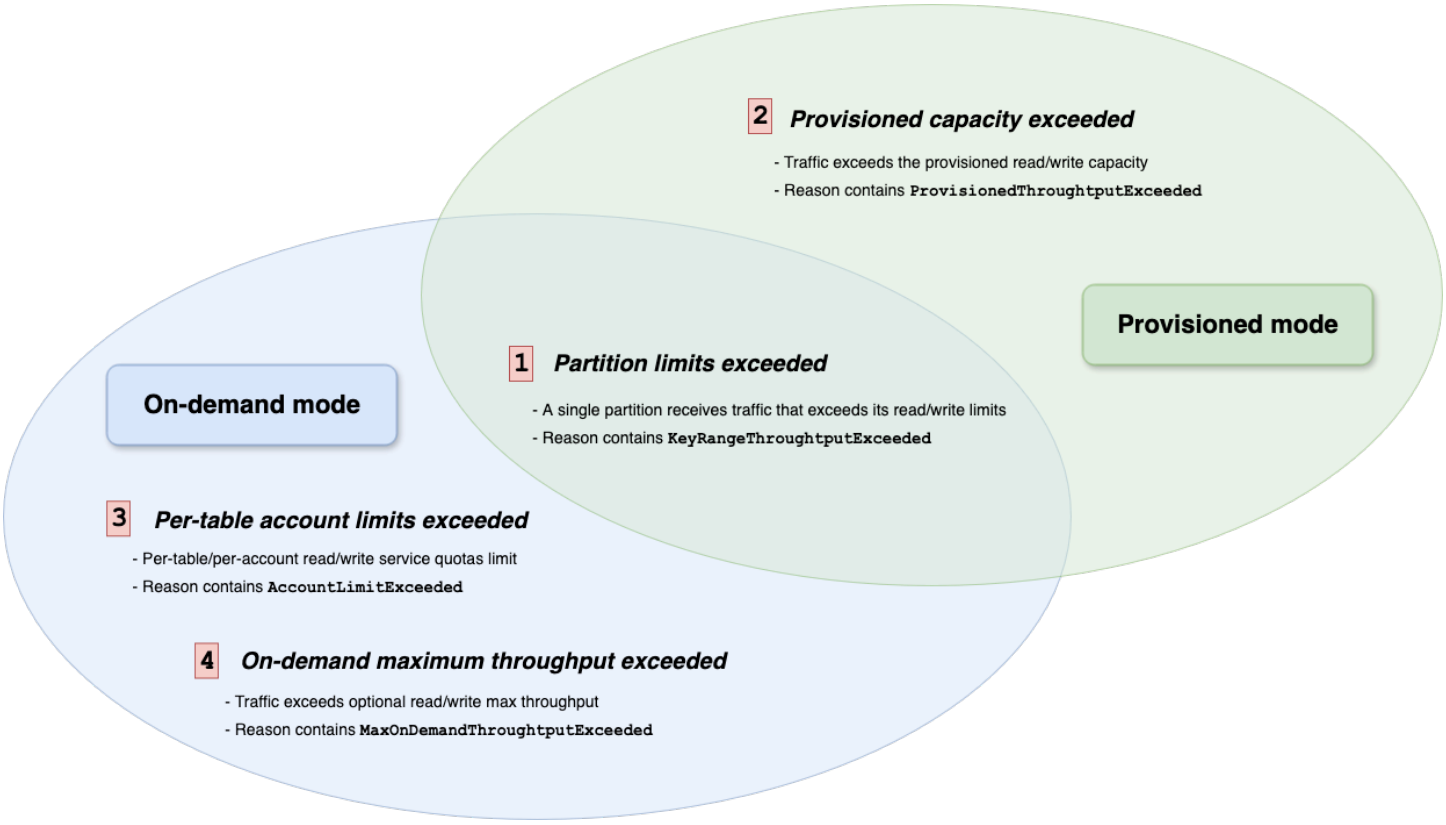
   O consumo excede as unidades de capacidade provisionada (leitura ou gravação) configuradas para uma tabela ou um índice secundário global (GSI).

3. Cotas de serviço em nível de conta excedidas (no modo sob demanda):

   O consumo faz com que uma tabela ou um GSI exceda as cotas de serviço em nível de conta por tabela para o throughput de leitura/gravação na região da AWS atual. Essas cotas são flexíveis e podem ser aumentadas.

4. Throughput máximo sob demanda excedido (no modo sob demanda):

   O consumo excede os limites máximos de throughput configurados para uma tabela ou um GSI. Esses são os limites que você configura especificamente para fins de controle de custos.



Este guia é organizado para ajudar você a entender e trabalhar com o controle de utilização no DynamoDB. Primeiro, ajudamos você a identificar o tipo específico de controle de utilização que afeta sua workload por meio de uma estrutura de diagnóstico.

Em seguida, a seção Guia de resolução oferece orientações específicas para cada cenário de controle de utilização, inclusive métricas do CloudWatch para monitorar a detecção e análise e as etapas recomendadas para otimização. Ao seguir essa abordagem estruturada, você pode diagnosticar melhor a causa raiz do controle de utilização e implementar a solução apropriada para garantir que suas tabelas do DynamoDB operem com eficiência.

Para começar, siga the section called “Diagnosticar o controle de utilização” para saber como identificar qual tipo de controle de utilização está afetando sua workload e implementar a estratégia de resolução recomendada.

Tópicos

- Diagnosticar o controle de utilização
- Guia de resolução de controle de utilização do DynamoDB
- Esclarecimentos sobre o controle de utilização de gravação e a contrapressão do índice secundário global (GSI) no DynamoDB

- Métricas de controle de utilização do CloudWatch

Diagnosticar o controle de utilização

Quando sua aplicação sofre controle de utilização, o DynamoDB fornece informações detalhadas sobre exceções e métricas específicas do CloudWatch para ajudar você a diagnosticar esses eventos.

Esta seção apresenta uma abordagem sistemática para entender os eventos de controle de utilização em aplicações do DynamoDB. Ela mostra como interpretar exceções de controle de utilização, correlacioná-las às métricas do CloudWatch para obter insights mais detalhados e entender quais alterações podem reduzir o controle de utilização em suas aplicações do DynamoDB.

Conceitos básicos sobre as exceções de controle de utilização

Quando o DynamoDB controla uma solicitação, ele exibe exceções específicas com informações detalhadas de diagnóstico. Por exemplo, em Java, essas exceções incluem `ProvisionedThroughputExceededException`, `RequestLimitExceeded` ou `ThrottlingException`.

Cada exceção inclui `ThrottlingReasons`, um conjunto de `ThrottlingReason` individual que contém dois campos principais para ajudar você a identificar e entender o controle de utilização:

- Um motivo: um campo concatenado que segue o formato `<ResourceType><OperationType><LimitType>`.
- ARN do recurso: o nome do recurso da Amazon (ARN) da tabela ou do índice afetado

O campo do motivo segue um padrão consistente que ajuda você a entender exatamente o que está acontecendo:

- ResourceType (o que está sofrendo controle de utilização): `Table` ou `Index`.
- OperationType (o tipo de operação): `Read` ou `Write`.
- LimitType (por que o controle de utilização ocorreu):
  - `KeyRangeThroughputExceeded`: isso ocorre quando uma partição específica que abriga sua tabela ou índice consome uma capacidade de leitura ou gravação que excede os limites internos de throughput por partição.

- `ProvisionedThroughputExceeded`: isso ocorre em uma tabela provisionada ou índice secundário global quando a taxa de consumo de leitura ou gravação excede o valor provisionado.
- `AccountLimitExceeded`: isso ocorre em uma tabela ou índice sob demanda quando a taxa de consumo de leitura ou gravação excede a taxa máxima de consumo de uma tabela e dos respectivos índices, conforme definido no nível da conta. É possível solicitar um aumento dessa cota.
- `MaxOnDemandThroughputExceeded`: isso ocorre em uma tabela ou índice sob demanda quando a taxa de consumo de leitura ou gravação excede a taxa máxima de consumo que o usuário configurou para a tabela ou o índice. Você pode aumentar esse valor por conta própria para qualquer valor até o limite da conta ou definir como -1 para indicar que não há limite fornecido pelo usuário.

O ARN do recurso identifica exatamente qual tabela ou índice está sofrendo controle de utilização:

- Para tabelas: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]`.
- Para índices: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]/index/[index-name]`.

Exemplos de motivos completos de controle de utilização:

- `TableReadProvisionedThroughputExceeded`
- `IndexWriteAccountLimitExceeded`

Isso ajuda a identificar exatamente qual recurso está sofrendo controle de utilização, que tipo de operação causou isso e por que o controle de utilização ocorreu.

Exemplos de exceção

Exemplo 1: capacidade provisionada excedida em um GSI

```
{
    "ThrottlingReasons": [
        {
            "reason": "IndexWriteProvisionedThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders/
index/OrderDateIndex"
```

```
        }
    ],
    "awsErrorDetails": {
        "errorCode": "ProvisionedThroughputExceeded",
        "errorMessage": "The level of configured provisioned throughput for the index
 was exceeded",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

Neste exemplo, a aplicação recebe uma `ProvisionedThroughputExceededException` com o motivo `IndexWriteProvisionedThroughputExceeded`. As gravações no `OrderDateIndex` estão sofrendo controle de utilização porque o consumo de gravação excedeu a capacidade de gravação provisionada configurada do GSI.

Exemplo 2: throughput máximo sob demanda excedido

```
{
    "ThrottlingReasons": [
        {
            "reason": "TableReadMaxOnDemandThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserSessions"
        }
    ],
    "awsErrorDetails": {
        "errorMessage": "Throughput exceeds the maximum OnDemandThroughput configured
 on table or index",
        "errorCode": "ThrottlingException",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

Neste exemplo, as leituras da tabela `UserSessions` estão sofrendo controle de utilização porque excedem o limite máximo de throughput sob demanda configurado na tabela.

Estrutura de diagnóstico de controle de utilização do DynamoDB

Quando sua aplicação se deparar com algum evento de controle de utilização, siga as etapas abaixo para diagnosticar e resolver o problema.

Etapa 1: analisar os detalhes de **`ThrottlingReason`**

1. Verifique o campo Motivo para identificar o motivo específico do controle de utilização. O motivo detalha o tipo de recurso com controle de utilização (tabela ou índice), o tipo de operação que está causando o evento de controle de utilização (leitura ou gravação) e o tipo de limite que foi excedido (partição, throughput provisionado e limite de conta).
2. Verifique o campo resourceArn para identificar qual recurso (tabela ou GSI) está sofrendo controle de utilização.
3. Use essas informações associadas para entender o contexto completo do problema de controle de utilização.

   Por exemplo, considere o cenário abaixo, em que você recebe a seguinte exceção `ProvisionedThroughputExceededException` pelo motivo de controle de utilização `TableWriteKeyRangeThroughputExceeded`: O resourceARN afetado é `arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders`.

   Essa combinação informa que as operações de gravação em sua tabela `CustomerOrders` estão sofrendo controle de utilização. O controle de utilização está ocorrendo em nível de partição (não em nível de tabela, caso em que seria exibido como `TableWriteProvisionedThroughputExceeded`). A causa raiz é que você excedeu a capacidade de throughput máxima em relação a um valor ou intervalo de chave de partição específico, indicando um problema de partição com acesso frequente.

   Compreender essa relação entre os elementos da exceção ajuda a implementar a estratégia de mitigação apropriada. Nesse caso, lidar com a partição com acesso frequente em vez de aumentar a capacidade geral provisionada da tabela.

Etapa 2: identificar e analisar as métricas relacionadas do CloudWatch

1. Identifique suas métricas: cada motivo de controle de utilização no DynamoDB corresponde diretamente a métricas específicas do CloudWatch que podem ser monitoradas para rastrear e analisar eventos de controle de utilização. É possível deduzir sistematicamente os nomes apropriados das métricas do CloudWatch com base no motivo do controle de utilização.

2. Correlacione o motivo do controle de utilização às métricas correspondentes do CloudWatch usando esta tabela de referência:

   Motivos completos de controle de utilização e referência de métricas do CloudWatch

Categoria	Motivo do controle de utilização	Principais métricas do CloudWatch
Capacidade provisionada excedida	TableReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents
	TableWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents
	IndexReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents (GSI)
	IndexWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents (GSI)
Limites de partição excedidos	TableReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents
	TableWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents
	IndexReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents (GSI)
	IndexWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents (GSI)

Categoria	Motivo do controle de utilização	Principais métricas do CloudWatch
Máximo sob demanda excedido	TableReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents
	TableWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents
	IndexReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents (GSI)
	IndexWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents (GSI)
Limites da conta excedidos	TableReadAccountLimitExceeded	ReadAccountLimitThrottleEvents
	TableWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents
	IndexReadAccountLimitExceeded	ReadAccountLimitThrottleEvents (GSIs)
	IndexWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents (GSIs)

Por exemplo, se você recebeu `IndexWriteProvisionedThroughputExceeded`, no mínimo você deve monitorar a métrica `WriteProvisionedThroughputThrottleEvents` do CloudWatch para o índice específico identificado no `ResourceArn`.

3. Monitore essas métricas no CloudWatch para entender a frequência e o momento dos eventos de controle de utilização, distinguir se o controle de utilização é de leitura ou de gravação, identificar padrões nos horários em que o controle de utilização aumenta e acompanhar suas tendências de utilização de capacidade.

   O DynamoDB publica métricas detalhadas para cada tabela e índice secundário global. As métricas (`ReadThrottleEvents`, `WriteThrottleEvents` e `ThrottledRequests`) agregam todos os eventos de controle de utilização em sua tabela e nos respectivos índices.

Etapa 3: identificar as chaves com controle de utilização e altas taxas de acesso usando o CloudWatch Contributor Insights (para controle de utilização relacionado a partições)

Se você identificou problemas relacionados à partição na Etapa 1 (como erros `KeyRangeThroughputExceeded`), o CloudWatch Contributor Insights para DynamoDB pode ajudar você a diagnosticar quais chaves específicas estão direcionando o tráfego e enfrentando eventos de controle de utilização em sua tabela ou índice.

1. Habilite o CloudWatch Contributor Insights para sua tabela ou índice que está sofrendo controle de utilização com base em seu `ResourceARN`.

   Você pode escolher o modo Chaves limitadas para se concentrar exclusivamente nas chaves com utilização mais limitada. Esse modo é ideal para monitoramento contínuo, pois só processa eventos quando ocorre controle de utilização. Como alternativa, o modo Chaves acessadas e limitadas ajuda você a procurar padrões nas chaves mais acessadas.
2. Analise os relatórios para identificar padrões problemáticos. Procure chaves com taxas de acesso ou controle de utilização desproporcionalmente altas e correlacione o controle de utilização e os padrões de tráfego. É possível criar painéis integrados associando os grafos do Contributor Insights e as métricas do CloudWatch para o DynamoDB.

Para ter informações detalhadas sobre o CloudWatch Contributor Insights, consulte Analisar acesso a dados usando o CloudWatch Contributor Insights para DynamoDB.

Etapa 4: determinar a solução apropriada

Depois de diagnosticar a causa específica do controle de utilização, implemente a solução recomendada com base em seu contexto específico. A solução apropriada depende de vários fatores, como cenário do controle de utilização, modo de capacidade da tabela, decisões de design

de tabela e chave, padrões de acesso e eficiência das consultas, configuração do índice global e secundário e arquitetura geral e pontos de integração do sistema.

Caso queira ver soluções detalhadas para abordar seus cenários específicos de controle de utilização, consulte a seção the section called “Guia de resolução”. Esse recurso oferece estratégias de correção direcionadas, personalizadas de acordo com seu motivo específico de controle de utilização e a configuração do modo de capacidade.

Etapa 5: monitorar seu progresso

1. Acompanhe as métricas do CloudWatch que correspondem ao seu cenário de controle de utilização.
2. Valide se suas estratégias de mitigação são eficazes observando se há uma diminuição nos eventos de controle de utilização.

Guia de resolução de controle de utilização do DynamoDB

Esta seção fornece orientações sobre resolução direcionadas a cada motivo específico de controle de utilização que o DynamoDB pode exibir. Cada entrada inclui abordagens de resolução sugeridas com base nas práticas recomendadas e nas métricas correspondentes do CloudWatch que devem ser monitoradas.

O DynamoDB implementa dezesseis motivos distintos de controle de utilização em quatro categorias principais. Use os motivos de controle de utilização da exceção da sua aplicação para acessar diretamente a orientação relevante.

Throughput de intervalo de chaves excedido (partições com acesso frequente)

Estes motivos de controle de utilização ocorrem quando partições individuais excedem os respectivos limites de throughput, afetando os modos provisionado e sob demanda:

- TableReadKeyRangeThroughputExceeded
- TableWriteKeyRangeThroughputExceeded
- IndexReadKeyRangeThroughputExceeded
- IndexWriteKeyRangeThroughputExceeded

Throughput provisionado excedido

Estes motivos de controle de utilização ocorrem quando as taxas de consumo excedem os limites de capacidade provisionada no modo provisionado:

- TableReadProvisionedThroughputExceeded
- TableWriteProvisionedThroughputExceeded
- IndexReadProvisionedThroughputExceeded
- IndexWriteProvisionedThroughputExceeded

Limites da conta excedidos

Estes motivos de controle de utilização ocorrem quando as taxas de consumo excedem as cotas de throughput em nível de conta em sua região da AWS:

- TableReadAccountLimitExceeded
- TableWriteAccountLimitExceeded
- IndexReadAccountLimitExceeded
- IndexWriteAccountLimitExceeded

Throughput máximo sob demanda excedido

Estes motivos de controle de utilização ocorrem quando as taxas de consumo excedem os limites máximos de throughput configurados no modo sob demanda:

- TableReadMaxOnDemandThroughputExceeded
- TableWriteMaxOnDemandThroughputExceeded
- IndexReadMaxOnDemandThroughputExceeded
- IndexWriteMaxOnDemandThroughputExceeded

1. Throughput de intervalo de chaves excedido (partições com acesso frequente)

O Amazon DynamoDB impõe limites de throughput específicos em nível de partição para a tabela e o índice secundário global (GSI). Cada partição tem um número máximo de unidades de capacidade de leitura (RCUs) e unidades de capacidade de gravação (WCUs) por segundo. Quando as partições recebem tráfego concentrado que excede esses limites, elas sofrem controle de utilização, enquanto

outras operações podem permanecer subutilizadas, criando “partições com acesso frequente”. O controle de utilização em nível de partição do DynamoDB opera de forma independente para leituras e gravações. Uma partição pode controlar a utilização de leituras enquanto as gravações continuam normalmente ou vice-versa. Esse controle de utilização pode ocorrer mesmo quando a tabela ou o GSI tem capacidade geral suficiente. Para saber mais a respeito de:

- Limites de partição do DynamoDB e design eficaz da chave de partição que abordam a prevenção de partições com acesso frequente, consulte Práticas recomendadas para projetar e usar chaves de partição de maneira eficaz no DynamoDB.
- Conceitos gerais sobre partição e distribuição de dados, consulte Partitions and data distribution in DynamoDB.
- Orientações adicionais e situações reais para gerenciar chaves de partição e throughput, consulte the section called “Recursos adicionais”.

Quando partições individuais excedem os respectivos limites de throughput, o DynamoDB exibe um tipo de motivo de controle de utilização `KeyRangeThroughputExceeded` na exceção de controle de utilização. As informações identificam que uma partição está enfrentando alto tráfego e qual tipo de operação (leitura ou gravação) está causando o problema.

O throughput do intervalo de chaves excedeu as medidas de mitigação.

Esta seção apresenta diretrizes de resolução para cenários de controle de utilização em nível de partição. Antes de usar este guia, identifique os motivos específicos de controle de utilização do tratamento de exceções da sua aplicação e determine o nome do recurso da Amazon (ARN) do recurso afetado. Para ter informações sobre como recuperar os motivos de controle de utilização e identificar os recursos com controle de utilização, consulte the section called “Estrutura de diagnóstico”.

Antes de se aprofundar em cenários específicos de controle de utilização, verifique se o problema é resolvido automaticamente:

- O DynamoDB geralmente se adapta a partições com acesso frequente por meio de seu mecanismo automático de divisão por aquecimento. Se você observar eventos de controle de utilização que se interrompem após um curto período, talvez sua tabela já tenha se adaptado dividindo a partição com acesso frequente. Quando as partições são divididas, cada nova partição lida com uma seção menor do espaço de chaves, o que pode ajudar a distribuir a carga de maneira mais uniforme. Em muitos casos, nenhuma ação adicional é necessária, pois o DynamoDB já resolveu o problema automaticamente.

Para ter mais informações sobre o mecanismo de divisão por aquecimento, consulte the section called “Recursos adicionais”.

Se o controle de utilização persistir, consulte os seguintes cenários específicos de controle de utilização para ver as opções de correção específicas:

- the section called “TableReadKeyRangeThroughputExceeded”
- the section called “TableWriteKeyRangeThroughputExceeded”
- the section called “IndexReadKeyRangeThroughputExceeded”
- the section called “IndexWriteKeyRangeThroughputExceeded”

TableReadKeyRangeThroughputExceeded

Quando isso ocorre

A taxa de consumo de uma ou mais partições na tabela do DynamoDB excede o limite de throughput de leitura da partição. Esse controle de utilização ocorre independentemente da capacidade total provisionada da tabela e afeta tanto as tabelas provisionadas quanto as sob demanda. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

Para os modos provisionado e sob demanda:

- Capacidade antes do aquecimento: se o controle de utilização persistir, verifique se sua tabela está limitada por sua capacidade. Consulte the section called “Throughput a quente”. Use o throughput a quente ou aumente a capacidade provisionada de leitura com antecedência para aumentos esperados no tráfego. Aumentar o throughput a quente melhora a capacidade da tabela para lidar com picos repentinos no tráfego antes que ocorra controle de utilização. Com o tempo, se o throughput real se aproximar consistentemente dos níveis de throughput a quente, o DynamoDB poderá dividir as partições muito usadas com base nos padrões de uso observados.
- Identifique as chaves com acesso frequente: se a tabela não resolver o problema automaticamente e o throughput a quente for alto ou se aumentá-lo não corrigir o problema, será necessário identificar chaves de acesso frequente específicas. Use the section called “Identificação de chaves

com acesso frequente usando o CloudWatch Contributor Insights" para determinar se algum valor específico de chave de partição indica acesso frequente. Essa é a primeira etapa para direcionar suas iniciativas de mitigação de forma eficaz. Observe que a identificação nem sempre é simples, principalmente no caso de partições em que o acesso frequente é revezado (quando partições diferentes recebem acesso frequente ao longo do tempo) ou quando o controle de utilização é acionado por determinadas operações, como verificações. Para esses cenários complexos, talvez seja necessário analisar os padrões de acesso da aplicação e correlacioná-los ao momento do evento de controle de utilização.

- Dependendo do seu caso de uso, considere a possibilidade de usar leituras finais consistentes: alterne de leituras altamente consistentes para leituras finais consistentes, que consomem metade das RCUs e podem dobrar imediatamente sua capacidade efetiva de leitura. Para ver as práticas recomendadas sobre a implementação de leituras finais consistentes e reduzir o consumo da capacidade de leitura, consulte the section called "Consistência de leitura do DynamoDB".
- Melhore o design da chave de partição: como uma solução de longo prazo, considere the section called "Melhorar o design da chave de partição" para distribuir o acesso de maneira mais uniforme entre as partições. Essa abordagem geralmente oferece a solução mais abrangente para problemas de partição com acesso frequente abordando a causa raiz. No entanto, requer um planejamento cuidadoso, pois envolve desafios significativos de migração.

TableWriteKeyRangeThroughputExceeded

Quando isso ocorre

A taxa de consumo de uma ou mais partições na tabela do DynamoDB excede o limite de throughput de gravação da partição. Esse controle de utilização ocorre independentemente da capacidade total provisionada da tabela e afeta tanto as tabelas provisionadas quanto as sob demanda. É possível monitorar as métricas do CloudWatch em the section called "Diagnóstico e monitoramento comuns" para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

Para os modos provisionado e sob demanda:

- Capacidade antes do aquecimento: se o controle de utilização persistir, verifique se sua tabela está limitada por sua capacidade. Consulte Noções básicas sobre o throughput a quente do DynamoDB. Use o throughput a quente ou aumente a capacidade provisionada de gravação com

antecedência para aumentos esperados no tráfego. Aumentar o throughput a quente melhora a capacidade da tabela para lidar com picos repentinos no tráfego antes que ocorra controle de utilização. Com o tempo, se o throughput real se aproximar consistentemente dos níveis de throughput a quente, o DynamoDB poderá dividir as partições muito usadas com base nos padrões de uso observados.

- Identifique as chaves com acesso frequente: se a tabela não resolver o problema automaticamente e o throughput a quente for alto ou se aumentá-lo não corrigir o problema, será necessário identificar chaves de acesso frequente específicas. Use the section called “Identificação de chaves com acesso frequente usando o CloudWatch Contributor Insights” para determinar se algum valor específico de chave de partição indica acesso frequente. Essa é a primeira etapa para direcionar suas iniciativas de mitigação de forma eficaz. Considere estes padrões comuns:
  - Se você vir que a mesma chave de partição aparece com frequência em seus dados de controle de utilização, isso indica que ela é uma chave de acesso concentrado.
  - Se você não vir chaves repetidas, mas a gravação de dados estiver ocorrendo de forma ordenada (como carimbos de data/hora sequenciais ou operações baseadas em verificação que seguem a ordem do espaço de chaves), é provável que haja partições com acesso frequente contínuo, caso em que diferentes chaves passam a ter acesso frequente ao longo do tempo à medida que suas gravações se movem pelo espaço de chaves.

  Observe que o controle de utilização de gravação também pode ocorrer com operações como `BatchWriteItem` ou transações que afetam vários itens simultaneamente. Quando itens individuais em uma solicitação `BatchWriteItem` sofrem controle de utilização, o DynamoDB não propaga esses erros de controle de utilização para o código da aplicação. Em vez disso, o DynamoDB exibe informações sobre os itens não processados na resposta, com os quais sua aplicação deve lidar ao tentar novamente esses itens específicos. Para transações, a operação inteira falhará com uma `TransactionCanceledException` se algum item sofrer controle de utilização. Diante desses cenários complexos, talvez seja necessário analisar os padrões de gravação e os fluxos de trabalho de ingestão de dados da aplicação, correlacioná-los com o momento dos eventos de controle de utilização e implementar estratégias apropriadas para lidar com novas tentativas.
- Melhore o design da chave de partição: como uma solução de longo prazo, considere the section called “Melhorar o design da chave de partição” para distribuir o acesso de maneira mais uniforme entre as partições. Essa abordagem geralmente oferece a solução mais abrangente para problemas de partição com acesso frequente abordando a causa raiz. No entanto, requer um planejamento cuidadoso, pois envolve desafios significativos de migração.

IndexReadKeyRangeThroughputExceeded

Quando isso ocorre

A taxa de consumo de uma ou mais partições no GSI do DynamoDB excede o limite de throughput de leitura da partição. Esse controle de utilização ocorre independentemente da capacidade total provisionada do GSI e afeta tanto as tabelas provisionadas quanto as sob demanda. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

- Capacidade antes do aquecimento: se o controle de utilização persistir, verifique se o GSI está limitado por sua capacidade. Consulte Noções básicas sobre o throughput a quente do DynamoDB. Use o throughput a quente ou aumente a capacidade provisionada de leitura com antecedência para aumentos esperados no tráfego. Aumentar o throughput a quente melhora a capacidade do GSI para lidar com picos repentinos no tráfego antes que ocorra controle de utilização. Com o tempo, se o throughput real se aproximar consistentemente dos níveis de throughput a quente, o DynamoDB poderá dividir as partições muito usadas com base nos padrões de uso observados.
- Identifique as chaves com acesso frequente: se o GSI não resolver automaticamente o problema e o throughput a quente for alto ou aumentá-la não corrigir o problema, você precisará identificar chaves com acesso frequente específicas. Use the section called “Identificação de chaves com acesso frequente usando o CloudWatch Contributor Insights” para determinar se algum valor específico de chave de partição indica acesso frequente. Essa é a primeira etapa para direcionar suas iniciativas de mitigação de forma eficaz. Observe que, para GSIs, a distribuição de chaves de partição pode diferir significativamente da tabela base, criando diferentes padrões de chaves com acesso frequente.
- Redefina as chaves de partição do GSI: considere se o design da chave do GSI pode estar criando pontos com acesso frequente não naturais (como sinalizadores de status, chaves somente de data ou atributos boolianos) que concentram as leituras em um pequeno número de partições. Considere a possibilidade de usar chaves compostas que combinem o atributo de baixa cardinalidade com um atributo de alta cardinalidade (p. ex., “ACTIVE#customer123” em vez de apenas “ACTIVE”) ou aplique as técnicas descritas em the section called “Fragmentação de gravação” aos itens da tabela base que afetam a distribuição do GSI para distribuir gravações em várias partições. Embora a consulta de dados fragmentados exija lógica adicional da aplicação

para agregar resultados, essa abordagem evita o controle de utilização ao distribuir os padrões de acesso de maneira mais uniforme.

IndexWriteKeyRangeThroughputExceeded

Quando isso ocorre

A taxa de consumo de uma ou mais partições no GSI do DynamoDB excede o limite de throughput de gravação da partição. Esse controle de utilização ocorre independentemente da capacidade total provisionada do GSI e afeta tanto as tabelas provisionadas quanto as sob demanda. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

- Redefina a chave de partição do GSI: analise o design da chave de partição do GSI para verificar se ela tem cardinalidade (exclusividade) suficiente para distribuir as gravações uniformemente. Uma causa comum de controle de utilização de gravação do GSI é o uso de atributos de baixa cardinalidade como chaves de partição do GSI (p. ex., sinalizadores de status com apenas alguns valores possíveis). Mesmo quando a tabela base tem chaves de partição bem distribuídas, ainda assim o GSI poderá experimentar partições com acesso frequente se a respectiva chave de partição concentrar gravações em um pequeno número de valores. Por exemplo, se 80% dos seus itens tiverem status="ACTIVE", isso criará uma partição com acesso frequente grave em um GSI baseado em status. Considere a possibilidade de usar chaves compostas que combinem o atributo de baixa cardinalidade com um atributo de alta cardinalidade (p. ex., “ACTIVE#customer123” em vez de apenas “ACTIVE”) ou aplique as técnicas descritas em the section called “Fragmentação de gravação” aos itens da tabela base que afetam a distribuição do GSI para distribuir gravações em várias partições. Embora a consulta de dados fragmentados exija lógica adicional da aplicação para agregar resultados, essa abordagem evita o controle de utilização ao distribuir os padrões de acesso de maneira mais uniforme.
- Capacidade antes do aquecimento: verifique se o GSI está limitado por sua capacidade. Consulte Noções básicas sobre o throughput a quente do DynamoDB. Use o throughput a quente ou aumente a capacidade provisionada de leitura com antecedência para aumentos esperados no tráfego. Aumentar o throughput a quente melhora a capacidade do GSI para lidar com picos repentinos no tráfego antes que ocorra controle de utilização. Com o tempo, se o throughput real

se aproximar consistentemente dos níveis de throughput a quente, o DynamoDB poderá dividir as partições muito usadas com base nos padrões de uso observados.

- Otimize as projeções do GSI: aplique as técnicas descritas em the section called “Otimizar as projeções do GSI” para reduzir o volume de gravação nos GSIs. Projetar menos atributos pode reduzir significativamente a capacidade de gravação consumida por cada atualização do GSI.

Diagnóstico e monitoramento comuns

Ao solucionar problemas de controle de utilização em nível de partição, várias métricas do CloudWatch podem ajudar a identificar partições com acesso frequente e confirmar a causa raiz.

Métricas essenciais do CloudWatch

Monitore estas métricas essenciais para diagnosticar o controle de utilização em nível de partição:

- Eventos de controle de utilização em nível de partição: `ReadKeyRangeThroughputThrottleEvents` e `WriteKeyRangeThroughputThrottleEvents` rastreiam quando partições individuais excedem os respectivos limites de throughput. `ReadThrottleEvents` e `WriteThrottleEvents` monitoram quando qualquer solicitação de leitura ou gravação excede a capacidade provisionada.
- Consumo de capacidade: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` mostram os padrões gerais de uso.

Procedimentos de resolução

Identificação de chaves com acesso frequente usando o CloudWatch Contributor Insights

Use esse procedimento para identificar quais chaves de partição estão causando controle de utilização.

1. Habilite o CloudWatch Contributor Insights em sua tabela ou GSI para rastrear as chaves com utilização mais limitada. Considere manter o CloudWatch Contributor Insights habilitado continuamente para alertas de controle de utilização em tempo real usando o modo Chaves limitadas. Esse modo se concentra exclusivamente em solicitações com controle de utilização, processando eventos somente quando ocorre controle. Esse monitoramento direcionado é uma forma econômica de manter a visibilidade contínua dos problemas de controle de utilização.
2. Identifique quais chaves estão causando os problemas de partição com acesso frequente.

3. (Se o modo completo Chaves acessadas e limitadas estiver habilitado.) Analise os padrões de acesso ao longo do tempo para determinar se as chaves com acesso frequente são consistentes ou ocorrem durante períodos específicos.

Melhorar o design da chave de partição

Use essa abordagem você puder modificar o esquema da tabela para distribuir melhor o tráfego entre as partições. Quando possível, essa é a solução de longo prazo mais eficaz para problemas de partição com acesso frequente. De preferência, o design da chave de partição deve ser cuidadosamente considerado durante a fase inicial de design da tabela.

A redefinição da chave de partição representa uma alteração fundamental em seu modelo de dados, pois afeta todo o ecossistema de aplicações. Antes de prosseguir com essa abordagem, considere cuidadosamente estas limitações importantes:

- Complexidade da migração de dados: a redefinição das chaves de partição exige a migração de todos os dados existentes, o que pode consumir muitos recursos e tempo no caso de tabelas grandes.
- Alterações no código da aplicação: é necessário atualizar todo o código da aplicação que lê ou grava na tabela para usar a nova estrutura de chaves.
- Impacto na produção: a migração para um novo design de chave geralmente exige tempo de inatividade ou estratégias complexas de dupla gravação durante a transição.

Para ver orientações e princípios abrangentes sobre design de chave de partição, consulte the section called “Design de chave de partição” e the section called “Distribuir workloads”.

Otimizar as projeções do GSI

Analise os padrões de consulta da aplicação para determinar exatamente quais atributos precisam estar disponíveis ao consultar o GSI e limite as projeções apenas a esses atributos. Quando você atualiza atributos que não são projetados em um GSI, nenhuma operação de gravação ocorre nesse GSI, reduzindo o consumo de throughput de gravação durante as atualizações. Essa estratégia de projeção direcionada otimiza o desempenho e o custo e, ao mesmo tempo, atende aos requisitos de consulta da aplicação. Observe que projetar menos atributos reduz o consumo da capacidade de gravação, mas pode exigir leituras adicionais da tabela base.

Para ter mais informações sobre estratégias de projeção eficientes, consulte Práticas recomendadas para uso de índices secundários no DynamoDB.

Recursos adicionais

As seguintes publicações de blog apresentam exemplos e detalhes práticos dos conceitos abordados neste guia:

- Para ter mais informações sobre o uso de GSIs para distribuir tráfego de leitura, consulte Using Global Secondary Indexes to create eventually consistent secondary indexes in Amazon DynamoDB.
- Para ver orientações práticas sobre como escalar o DynamoDB e gerenciar partições com acesso frequente, consulte Scaling DynamoDB: How partitions, hot keys, and split for heat impact performance (Part 1: Loading).
- Para ter informações detalhadas sobre como o mecanismo de divisão por aquecimento do DynamoDB funciona, seus benefícios e detalhes de implementação, consulte Scaling DynamoDB: How partitions, hot keys, and split for heat impact performance (Part 3: Summary and best practices).
- Para obter informações sobre estratégias detalhadas de fragmentação de gravação, consulte the section called “Fragmentação de gravação”.

2. Throughput provisionado excedido

O controle de utilização de capacidade provisionada ocorre quando a taxa de consumo da aplicação excede as unidades de capacidade de leitura ou gravação (RCUs/WCUs) configuradas para tabelas ou índices secundários globais. Embora o DynamoDB ofereça capacidade de expansão para lidar com picos de tráfego ocasionais, solicitações prolongadas além dos limites provisionados resultam em controle de utilização. Quando isso acontece, o DynamoDB exibe um tipo de motivo `ProvisionedThroughputExceeded` de controle de utilização na exceção de controle. O motivo identifica se o problema está nas operações de leitura ou gravação e se ele afeta a tabela base ou um índice secundário global.

O controle de utilização pode ocorrer independentemente de o ajuste de escala automático estar habilitado. O ajuste de escala automático se adapta aos aumentos no consumo, mas não responde imediatamente e é restrito pelos limites máximos de capacidade que você configura. Isso significa que o controle de utilização ainda pode ocorrer durante picos repentinos de tráfego ou quando o consumo excede seus limites máximos de ajuste de escala automático.

O throughput provisionado excedeu as medidas de mitigação

Esta seção apresenta orientações sobre resolução para cenários de controle de utilização de capacidade provisionada. Antes de usar este guia, identifique o motivo específico do controle de utilização no tratamento de exceções da sua aplicação e determine o nome do recurso da Amazon (ARN) do recurso afetado. Para ter informações sobre como recuperar os motivos de controle de utilização e identificar os recursos com controle de utilização, consulte the section called “Estrutura de diagnóstico”.

Antes de se aprofundar em cenários específicos de controle de utilização, considere se o controle é realmente um problema que precisa ser resolvido:

- O controle de utilização ocasional é normal e esperado em aplicações bem otimizadas do DynamoDB. Controle de utilização significa simplesmente que você está consumindo 100% do que provisionou. Se sua aplicação lida com o controle de utilização normalmente com novas tentativas e seu desempenho geral atende aos requisitos, o controle de utilização pode não exigir ação imediata.
- No entanto, se o controle de utilização estiver causando uma latência inaceitável do lado do cliente, degradando a experiência do usuário ou impedindo que operações essenciais sejam concluídas em tempo hábil, prossiga com as opções de mitigação a seguir.

Quando precisar resolver problemas de controle de utilização, primeiro determine se o controle é causada por:

- Picos de tráfego temporários: aumentos de curta duração no tráfego que excedem sua capacidade provisionada, mas não são prolongados. Isso requer estratégias diferentes das aplicadas a tráfego intenso contínuo.
- Alto tráfego contínuo: workloads prolongadas que excedem consistentemente a capacidade provisionada.

Para picos de tráfego, considere as estratégias apresentadas no blog Handle traffic spikes with Amazon DynamoDB provisioned capacity em the section called “Recursos adicionais”.

Para tráfego intenso contínuo, considere as seguintes opções de ajuste de capacidade:

- the section called “TableReadProvisionedThroughputExceeded”
- the section called “TableWriteProvisionedThroughputExceeded”

- the section called “IndexReadProvisionedThroughputExceeded”
- the section called “IndexWriteProvisionedThroughputExceeded”

TableReadProvisionedThroughputExceeded

Quando isso ocorre

A taxa de consumo de leitura da aplicação excede as unidades de capacidade de leitura (RCUs) provisionadas configuradas para a tabela. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Abordagem de resolução

Considere estas estratégias para resolver o controle de utilização de capacidade de leitura:

- Mude para o modo de capacidade sob demanda: considere mudar a tabela para o modo sob demanda se você enfrentar controles de utilização frequentes devido a picos de tráfego. O modo sob demanda elimina as preocupações com provisionamento e escala automaticamente com a workload.
- Se a opção de manter o modo provisionado e o ajuste de escala automático não estiver habilitada:
  - Considere a possibilidade de aumentar a capacidade de leitura da tabela.
  - Habilite o ajuste de escala automático da capacidade de leitura em sua tabela.
- Se o ajuste de escala automático estiver habilitado (o padrão para tabelas criadas no console):
  - Otimize os parâmetros de ajuste de escala automático de leitura da sua tabela.

TableWriteProvisionedThroughputExceeded

Quando isso ocorre

A taxa de consumo de gravação da aplicação excede as unidades de capacidade de gravação (WCUs) provisionadas configuradas para a tabela. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Abordagem de resolução

Considere estas estratégias para resolver o controle de utilização de capacidade de gravação:

- Mude para o modo de capacidade sob demanda: considere mudar a tabela para o modo sob demanda se você enfrentar controles de utilização frequentes devido a picos de tráfego. O modo sob demanda elimina as preocupações com provisionamento e escala automaticamente com a workload.
- Se a opção de manter o modo provisionado e o ajuste de escala automático não estiver habilitada:
  - Considere a possibilidade de aumentar a capacidade de gravação da tabela.
  - Habilite o ajuste de escala automático da capacidade de gravação em sua tabela.
- Se o ajuste de escala automático estiver habilitado (o padrão para tabelas criadas no console):
  - Otimize os parâmetros de ajuste de escala automático de gravação da sua tabela.

IndexReadProvisionedThroughputExceeded

Quando isso ocorre

O consumo de leitura em um índice secundário global (GSI) excede as unidades de capacidade de leitura provisionadas (RCUs) do GSI. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Abordagem de resolução

Considere estas estratégias para resolver o controle de utilização de capacidade de leitura do GSI:

- Mude para o modo de capacidade sob demanda: considere mudar a tabela base para sob demanda se você experimentar controles de utilização frequentes devido a picos de tráfego. O modo sob demanda elimina as preocupações com provisionamento e escala automaticamente com a workload.
- Se a opção de manter o modo provisionado e o ajuste de escala automático não estiver habilitada:
  - Considere a possibilidade de aumentar a capacidade de leitura do GSI.
  - Habilite o ajuste de escala automático da capacidade de leitura em seu GSI.
- Se o ajuste de escala automático estiver habilitado (o padrão para tabelas criadas no console):
  - Otimize os parâmetros de ajuste de escala automático de leitura do seu GSI.

IndexWriteProvisionedThroughputExceeded

Quando isso ocorre

As atualizações dos itens na tabela base acionam gravações em um GSI que excedem a capacidade de gravação provisionada do GSI. Isso causa controle de utilização de contrapressão nas gravações da tabela base. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Abordagem de resolução

Considere estas estratégias para resolver o controle de utilização de capacidade de gravação do GSI:

- Mude para o modo de capacidade sob demanda: considere mudar a tabela base para sob demanda se você experimentar controles de utilização frequentes devido a picos de tráfego. O modo sob demanda elimina as preocupações com provisionamento e escala automaticamente com a workload.
- Se a opção de manter o modo provisionado e o ajuste de escala automático não estiver habilitada:
  - Considere a possibilidade de aumentar a capacidade de gravação do GSI.
  - Habilite o ajuste de escala automático da capacidade de gravação em seu GSI.
- Se o ajuste de escala automático estiver habilitado (o padrão para tabelas criadas no console):
  - Otimize os parâmetros de ajuste de escala automático de gravação do seu GSI.

Diagnóstico e monitoramento comuns

Ao solucionar erros de throughput, várias métricas do CloudWatch podem ajudar a identificar a causa raiz.

Métricas essenciais do CloudWatch

Monitore estas métricas essenciais para diagnosticar o controle de utilização de capacidade provisionada:

- Eventos de controle de utilização: `ReadProvisionedThroughputThrottleEvents` e `WriteProvisionedThroughputThrottleEvents` rastreiam quando as solicitações sofrem controle de utilização por esse motivo. `ReadThrottleEvents` e `WriteThrottleEvents` monitoram quando qualquer solicitação de leitura ou gravação excede a capacidade provisionada.
- Consumo de capacidade: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` mostram o uso real.
- Capacidade provisionada: `ProvisionedReadCapacityUnits` e `ProvisionedWriteCapacityUnits` mostram os limites configurados.

Procedimentos de resolução

Aumentar a capacidade de throughput da tabela

Use esse procedimento quando o ajuste de escala automático não estiver habilitado e você precisar de um aumento imediato de capacidade.

1. Atualize a capacidade provisionada da tabela usando o console do DynamoDB, a AWS CLI ou o SDK:
   - Para capacidade de leitura: aumente o parâmetro `ReadCapacityUnits`, que especifica o número máximo de leituras altamente consistentes consumidas por segundo antes de o DynamoDB limitar as solicitações.
   - Para capacidade de gravação: aumente o parâmetro `WriteCapacityUnits`, que especifica o número máximo de gravações consumidas por segundo antes de o DynamoDB limitar as solicitações.
2. Verifique se as novas configurações de capacidade não excedem as cotas de throughput por tabela e se o consumo total da sua conta permanece abaixo das cotas de throughput por conta da sua região. Entretanto, se você estiver se aproximando desses limites, considere a possibilidade de mudar para o modo de capacidade sob demanda.

Configurar o ajuste de escala automático de tabela para ajustar a capacidade de leitura ou gravação de sua tabela ou GSI

Configure o ajuste de escala automático do DynamoDB para ajustar automaticamente a capacidade de leitura ou gravação com base nos padrões de tráfego. Você pode configurar o ajuste de escala automático de forma independente para tabelas e GSIs, com controles separados para unidades de capacidade de leitura e gravação.

1. Habilite o ajuste de escala automático de capacidade de leitura, de capacidade de gravação ou de ambas em sua tabela ou GSI.
2. Defina a porcentagem de utilização pretendida com margem para picos de tráfego.

> **Note**
>
> Uma porcentagem de utilização pretendida menor aumenta os custos e a frequência de ajuste de escala. Porcentagens de utilização pretendida abaixo de 40% podem

causar excesso de provisionamento. Monitore os padrões de uso e os custos para contrabalançar desempenho e eficiência.

3. Defina limites de capacidade:

   - RCUs/WCUs mínimas: mantêm capacidade suficiente durante períodos de baixo tráfego.
   - RCUs/WCUs máximas: atendem a demandas de pico de tráfego e impedem eventos de ajuste de escala descontrolados.

Para obter orientação sobre como configurar e gerenciar o ajuste de escala automático do DynamoDB Auto, consulte Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB.

> **Note**
>
> O ajuste de escala automático normalmente leva vários minutos para responder a mudanças no tráfego. Para picos repentinos de tráfego, a capacidade de expansão da tabela oferece proteção imediata, enquanto o ajuste de escala automático se adapta aos picos. Configure a utilização pretendida com uma margem adequada para dar tempo para as operações de ajuste escala e preservar a capacidade de expansão no caso de demandas inesperadas.

Otimizar as configurações de ajuste de escala automático de leitura ou gravação de sua tabela ou índice

Use esse procedimento quando o ajuste de escala automático estiver habilitado, mas ainda houver controle de utilização. Você pode definir o ajuste de escala automático de forma independente para tabelas e índices secundários globais (GSIs), com controles separados para unidades de capacidade de leitura e gravação.

- Ajuste a utilização pretendida: considere a possibilidade de reduzir a utilização pretendida da tabela ou dos GSIs para acionar o ajuste de escala antes que o controle de utilização ocorra. Monitore o tráfego depois de fazer esses ajustes. Consulte the section called “Configurar o ajuste de escala automático de tabela para ajustar a capacidade de leitura ou gravação de sua tabela ou GSI” para ter mais informações sobre o consumo de capacidade e as implicações de custo.
- Analise os limites de capacidade: garanta que as configurações de capacidade mínima e máxima estejam alinhadas aos seus padrões reais de workload.

Alternar para o modo de capacidade sob demanda

Para ter informações gerais sobre como alternar entre os modos de capacidade, consulte the section called “Alternar modos de capacidade”. Consulte o Service Quotas para saber mais sobre restrições específicas ao alternar entre os modos.

Aumentar a capacidade de throughput do GSI

Use esse procedimento quando o ajuste de escala automático não estiver habilitado em seu GSI e você precisar de um aumento imediato de capacidade.

1. Atualize a capacidade provisionada do GSI usando o console do DynamoDB, a AWS CLI ou o SDK:
   - Para capacidade de leitura: aumente o parâmetro `ReadCapacityUnits`, que especifica o número máximo de leituras que o GSI pode consumir por segundo antes de o DynamoDB limitar as solicitações. Observe que os GSIs permite apenas leituras finais consistentes.
   - Para capacidade de gravação: aumente o parâmetro `WriteCapacityUnits`, que especifica o número máximo de gravações que o GSI pode consumir por segundo antes de o DynamoDB limitar as solicitações.
2. Garanta que a capacidade de throughput provisionado do GSI permaneça de acordo com as cotas de throughput por conta e por tabela.

Recursos adicionais

- Para ter informações detalhadas sobre como lidar com picos de tráfego nas tabelas de capacidade provisionada do DynamoDB, inclusive várias estratégias, desde a utilização do ajuste de escala automático e da capacidade de expansão até o gerenciamento estratégico do controle de utilização, consulte Handle traffic spikes with Amazon DynamoDB provisioned capacity.
- Para obter informações sobre como usar uma expressão cron para programar uma política de ajuste de escala, consulte Optimize costs by scheduling provisioned capacity for Amazon DynamoDB.
- Para ter informações práticas sobre monitoramento e análise dos padrões de utilização de throughput de tabelas do DynamoDB no modo de capacidade provisionada, consulte How to evaluate throughput utilization for Amazon DynamoDB tables in provisioned mode.

3. Limites da conta excedidos

As tabelas sob demanda não têm níveis de capacidade provisionada para gerenciar, mas o DynamoDB impõe limites de throughput em nível de conta para evitar a execução descontrolada e garantir o uso justo de recursos entre todos os clientes. Esses limites de conta por tabela servem como proteções ajustáveis, definidas para cada combinação de conta e região. Quando sua taxa de consumo de leitura ou gravação excede esses limites, o DynamoDB exibe um tipo de motivo de controle de utilização `AccountLimitExceeded` na respectiva exceção. Os limites padrão de conta por tabela se aplicam automaticamente quando as tabelas não têm configurações personalizadas definidas de throughput máximo. Opcionalmente, você pode definir as configurações de throughput máximo para melhorar o controle de custos e a previsibilidade ou solicitar aumentos de cota por meio do console do the section called “Cotas” se os requisitos da aplicação excederem os limites padrão.

O limite da conta excedeu as medidas de mitigação

Esta seção apresenta diretrizes de resolução para cenários de controle de utilização de limites de conta. Antes de usar este guia, identifique os motivos específicos de controle de utilização do tratamento de exceções da sua aplicação e determine o nome do recurso da Amazon (ARN) do recurso afetado. Para ter informações sobre como recuperar os motivos de controle de utilização e identificar os recursos com controle de utilização, consulte the section called “Estrutura de diagnóstico”.

Antes de se aprofundar em cenários específicos de controle de utilização, primeiro determine se uma ação é realmente necessária:

- Avalie o impacto no desempenho: verifique se, apesar do controle de utilização, sua aplicação ainda está atendendo aos requisitos de desempenho. Muitas aplicações operam com sucesso nos limites ou próximo aos limites, especialmente durante operações ou migrações de dados em massa.
- Analise os padrões de controle de utilização: se o controle de utilização for intermitente e sua aplicação lidar com novas tentativas de forma eficaz, os limites atuais podem ser suficientes para sua workload.

Se sua aplicação tiver um desempenho aceitável mesmo quando ocasionalmente atingir os limites da conta, você poderá optar por simplesmente monitorar a situação em vez de implementar alterações imediatas.

Se você determinar que o controle de utilização está causando problemas de desempenho ou de confiabilidade inaceitáveis, selecione um motivo específico de controle abaixo para encontrar as opções de mitigação recomendadas:

- the section called “TableReadAccountLimitExceeded”
- the section called “TableWriteAccountLimitExceeded”
- the section called “IndexReadAccountLimitExceeded”
- the section called “IndexWriteAccountLimitExceeded”

TableReadAccountLimitExceeded

Quando isso ocorre

O consumo de leitura da tabela excedeu a cota de throughput de leitura por tabela em nível de conta para sua região. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Abordagem de resolução

Use as seguintes etapas para resolver esse controle de utilização:

- Solicitar aumentos de cota:

  Solicite um aumento no limite de throughput de leitura por tabela (código de cota L-CF0CBE56). Para ver as etapas detalhadas sobre como enviar a solicitação, consulte Solicitar aumentos de cota por tabela.

TableWriteAccountLimitExceeded

Quando isso ocorre

O consumo de gravação da tabela excedeu a cota de throughput de gravação por tabela em nível de conta para sua região. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Abordagem de resolução

Use as seguintes etapas para resolver esse controle de utilização:

- Solicitar aumentos de cota: solicite um aumento no limite de throughput de gravação por tabela (código de cota L-AB614373). Para ver as etapas detalhadas sobre como enviar a solicitação, consulte Solicitar aumentos de cota por tabela.

IndexReadAccountLimitExceeded

Quando isso ocorre

As operações de leitura direcionadas a um índice secundário global (GSI) consomem mais throughput do que a cota de leitura por tabela da sua conta permite na sua região da AWS atual. A cota de throughput de leitura por tabela em nível de conta se aplica conjuntamente a uma tabela e a todos os respectivos GSIs. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Abordagem de resolução

Escolha a resolução apropriada com base na distribuição de capacidade da sua conta:

- Solicitar aumentos de cota: solicite um aumento no limite de throughput de leitura por tabela (código de cota L-CF0CBE56). Para ver as etapas detalhadas sobre como enviar a solicitação, consulte Solicitar aumentos de cota por tabela.
- Otimizar o uso do GSI: analise o design do GSI e os padrões de consulta para reduzir o consumo desnecessário da capacidade de leitura.

IndexWriteAccountLimitExceeded

Quando isso ocorre

As operações de gravação na tabela base geram atualizações correspondentes em um GSI que, coletivamente, excedem a cota de throughput de gravação por tabela em nível de conta para sua região da AWS. Cada gravação em um item da tabela base que contém atributos indexados por um GSI aciona uma operação de gravação correspondente a esse GSI. Essas operações de gravação combinadas são contabilizadas em sua cota de throughput de gravação por tabela. Você pode monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar os padrões e o tempo desses eventos de controle de utilização e identificar quais operações estão causando a atividade excessiva de gravação do GSI.

Abordagem de resolução

Escolha a resolução apropriada com base na distribuição de capacidade da sua conta:

- Solicitar aumentos de cota: solicite um aumento no limite de throughput de gravação por tabela (código de cota L-AB614373) para atender a um tráfego maior de gravação do GSI das operações da tabela base. A cota de throughput de gravação por tabela se aplica à tabela inteira, inclusive a todos os GSIs. Para ver as etapas detalhadas sobre como enviar a solicitação, consulte Solicitar aumentos de cota por tabela.
- Otimizar as projeções de GSI: analise as projeções e o design do GSI para reduzir o volume de gravação nos GSIs.

Diagnóstico e monitoramento comuns

Ao solucionar problemas de eventos de controle de utilização de limite de conta excedido, várias métricas do CloudWatch podem ajudar a identificar se você está atingindo os limites por tabela ou em toda a conta e entender seus padrões de distribuição de capacidade.

Métricas essenciais do CloudWatch

Monitore estas métricas essenciais para diagnosticar o controle de utilização do limite da conta:

- Eventos de controle de utilização de limites de conta: `ReadAccountLimitThrottleEvents` e `WriteAccountLimitThrottleEvents` rastreiam quando as solicitações sofrem controle de utilização devido a limites em nível de conta. `ReadThrottleEvents` e `WriteThrottleEvents` monitoram quando qualquer solicitação de leitura ou gravação excede a capacidade provisionada.
- Consumo de capacidade por recurso: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` para cada tabela e o GSI mostram o uso individual dos recursos.
- Consumo em toda a conta: use expressões matemáticas de métricas do CloudWatch para somar a capacidade consumida em todas as tabelas e GSIs e monitorar o uso total da conta.

Procedimentos de resolução

Solicitar aumentos de cota por tabela

Se suas aplicações precisarem operar além dos limites atuais de throughput por tabela, você deverá enviar uma solicitação de aumento de cota usando o procedimento a seguir. Cada tabela do DynamoDB em sua conta da AWS (com todos os GSIs associados) está sujeita a essas cotas de throughput em uma região específica. Essas cotas representam a capacidade máxima de leitura ou

gravação que qualquer tabela individual e os respectivos GSIs podem consumir conjuntamente e se aplicam de forma independente a cada tabela, e não como um total em todas as tabelas da sua conta.

Também é possível definir limites mais baixos por tabela ou por GSI definindo as respectivas configurações máximas de throughput sob demanda.

1. Identifique a cota específica que precisa ser aumentada:
   - Limite de throughput de leitura por tabela (código de cota L-CF0CBE56): 40 mil RCUs padrão por tabela.
   - Limite de throughput de gravação por tabela (código de cota L-AB614373): 40 mil WCUs padrão por tabela.
2. Use o console do AWS Service Quotas para solicitar um aumento:
   - Navegue até o serviço do DynamoDB no Service Quotas.
   - Encontre a cota apropriada usando o código da cota.
   - Solicite um aumento com base no pico de uso previsto.
3. Forneça uma justificação para o aumento, como:
   - padrões de uso atuais e requisitos de pico de tráfego;
   - justificação comercial para o aumento da capacidade;
   - cronograma referente a quando o aumento de capacidade é necessário.

> **Note**
>
> Os aumentos de cota normalmente levam de 24 a 48 horas para serem processados. Planeje suas solicitações adequadamente e considere estratégias temporárias de mitigação enquanto aguarda a aprovação.

Otimizar as projeções e o design do GSI

Otimize as projeções e o design do índice secundário global (GSI) para reduzir o consumo de capacidade e melhorar o desempenho.

Estratégias de projeção seletiva

Se suas consultas precisam acessar apenas alguns recursos, é possível reduzir a quantidade de dados gravados no GSI quando os itens da tabela base mudam projetando somente esses recursos. Para obter detalhes sobre os tipos de projeção, consulte Projections for Global Secondary Indexes.

1. Analise os padrões de consulta: revise os padrões de consulta da aplicação para identificar quais atributos são realmente acessados por meio do GSI.
2. Use projeções seletivas: projete apenas os atributos que são realmente necessários nas consultas para reduzir o volume de gravação.
3. Considere usar `KEYS_ONLY`: se suas consultas precisarem apenas dos atributos principais, use a projeção `KEYS_ONLY` para minimizar o volume de gravação.
4. Contrabalançar as concessões entre leitura e gravação: projetar menos atributos reduz o consumo da capacidade de gravação, mas pode exigir leituras adicionais da tabela base.

Implementação de GSIs esparsos

Os GSIs esparsos contêm somente itens que têm o atributo indexado, em vez de todos os itens da tabela base. Isso reduz a densidade da partição e melhora o desempenho quando você consulta com frequência subconjuntos específicos de dados.

1. Crie GSIs que incluam somente itens com valores de atributo específicos.
2. Implemente a indexação condicional definindo somente o atributo da chave de partição do GSI nos itens que devem ser indexados.
3. Use chaves compostas em GSIs esparsos (p. ex., status#timestamp) para distribuir ainda mais o tráfego no subconjunto de itens indexados.

Para ter mais informações sobre como implementar essas estratégias, consulte Práticas recomendadas para uso de índices secundários no DynamoDB.

4. Throughput máximo sob demanda excedido

Ao configurar uma tabela sob demanda ou um GSI, é possível, opcionalmente, definir um limite máximo de throughput (`MaxReadRequestUnits` e `MaxWriteRequestUnits`) em nível de tabela ou de índice para evitar custos excessivos ou impedir que os sistemas subsequentes fiquem sobrecarregados. Para ter mais informações sobre throughput máximo, consulte the section called “Throughput máximo do DynamoDB para tabelas sob demanda”.

Quando o consumo de leitura ou gravação excede esses limites autoimpostos, as solicitações adicionais que excedem o limite recebem respostas rápidas de controle de utilização. O DynamoDB exibe exceções com um tipo de motivo `MaxOnDemandThroughputExceeded` controle de utilização, indicando qual recurso atingiu o respectivo limite de throughput.

Controle de utilização de throughput máximo sob demanda excedido

Esta seção oferece orientações sobre resolução para cenários de controle de utilização de throughput máximo sob demanda excedido. Antes de usar este guia, identifique os motivos específicos de controle de utilização do tratamento de exceções da sua aplicação e determine o nome do recurso da Amazon (ARN) do recurso afetado. Para ter informações sobre como recuperar os motivos de controle de utilização e identificar os recursos com controle de utilização, consulte the section called “Estrutura de diagnóstico”.

Antes de se aprofundar em cenários específicos de controle de utilização, considere se uma ação é realmente necessária:

- Avalie suas configurações de throughput máximo: esses limites foram intencionalmente configurados para controlar os custos ou proteger sistemas subsequentes. Se você está recebendo eventos de controle de utilização `MaxOnDemandThroughputExceeded`, seus limites estão funcionando conforme planejado. Considere se o aumento desses limites está alinhado com suas metas originais de controle de custos ou de proteção do sistema.
- Avalie o impacto da aplicação: determine se o controle de utilização está realmente causando problemas para suas aplicações ou usuários. Se suas aplicações lidam com novas tentativas de forma eficaz e atendem aos requisitos de desempenho apesar do controle de utilização ocasional, manter seus limites atuais pode ser a escolha apropriada.
- Analise os padrões de tráfego: analise se o controle de utilização representa um padrão de tráfego esperado ou um pico incomum. Para padrões de tráfego previsíveis e recorrentes que excedem consistentemente seus limites, talvez seja necessário ajustar as configurações throughput máximo. Para picos temporários, implementar melhores estratégias de distribuição de solicitações pode ser mais apropriado do que aumentar os limites.

Se, depois de considerar, você determinar que suas configurações de throughput máximo precisam ser ajustadas, consulte os seguintes cenários específicos de controle de utilização para ver as opções de correção específicas:

- the section called “TableReadMaxOnDemandThroughputExceeded”
- the section called “TableWriteMaxOnDemandThroughputExceeded”

- the section called “IndexReadMaxOnDemandThroughputExceeded”
- the section called “IndexWriteMaxOnDemandThroughputExceeded”

TableReadMaxOnDemandThroughputExceeded

Quando isso ocorre

Sua tabela sob demanda excedeu a capacidade de throughput máximo de leitura configurada. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

- Aumente o limite máximo de throughput: use o console do DynamoDB, a AWS CLI ou a API `UpdateTable` do DynamoDB para aumentar o valor de `MaxReadRequestUnits` para a tabela afetada e, seguida, monitore e faça ajustes. Isso permite que a tabela lide com um throughput de leitura mais alto antes que ocorra controle de utilização.
- Remova o limite máximo: defina `MaxReadRequestUnits` como `-1` para remover o teto, permitindo o ajuste de escala com base na demanda até atingir as cotas de throughput em nível de conta. Isso remove seu limite personalizado, mas ainda mantém as proteções em nível de conta da AWS. No entanto, é importante monitorar os gastos rigorosamente depois de remover esse limite, pois daí em diante sua tabela poderá consumir uma capacidade significativamente maior antes de atingir as cotas em nível de conta.

TableWriteMaxOnDemandThroughputExceeded

Quando isso ocorre

Sua tabela sob demanda excedeu a capacidade de throughput máximo de gravação configurada. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

- Aumente o limite máximo de throughput: use o console do DynamoDB, a AWS CLI ou a API `UpdateTable` do DynamoDB para aumentar o valor de `MaxWriteRequestUnits` para a tabela afetada e, seguida, monitore e faça ajustes.
- Remova o limite máximo: defina `MaxWriteRequestUnits` como `-1` para remover o teto, permitindo o ajuste de escala com base na demanda até atingir as cotas de throughput em nível de conta. Isso remove seu limite personalizado, mas ainda mantém as proteções em nível de conta da AWS. No entanto, é importante monitorar os gastos rigorosamente depois de remover esse limite, pois daí em diante sua tabela poderá consumir uma capacidade significativamente maior antes de atingir as cotas em nível de conta.

IndexReadMaxOnDemandThroughputExceeded

Quando isso ocorre

As solicitações de leitura a um GSI no modo sob demanda excederam a capacidade de throughput máximo de leitura configurada para o GSI. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

- Aumente o limite máximo de throughput do GSI: use o console do DynamoDB, a AWS CLI ou a API `UpdateTable` do DynamoDB para aumentar o valor de `MaxReadRequestUnits` para o GSI afetado e, em seguida, monitore e faça ajustes.
- Remova o limite máximo do GSI: defina `MaxReadRequestUnits` como `-1` para o GSI a fim de remover o teto, permitindo o ajuste de escala com base na demanda até atingir as cotas de throughput em nível de conta. Isso remove seu limite personalizado, mas ainda mantém as proteções em nível de conta da AWS. No entanto, é importante monitorar os gastos rigorosamente depois de remover esse limite.

IndexWriteMaxOnDemandThroughputExceeded

Quando isso ocorre

As atualizações dos itens na tabela base acionam gravações em um GSI no modo sob demanda que excedem a capacidade de throughput máximo de gravação configurada do GSI, causando controle

de utilização de contrapressão. É possível monitorar as métricas do CloudWatch em the section called “Diagnóstico e monitoramento comuns” para analisar o evento de controle de utilização.

Opções de correção

Considere estas etapas para lidar com eventos de controle de utilização:

- Aumente o limite máximo de throughput do GSI: use o console do DynamoDB, a AWS CLI ou a API `UpdateTable` do DynamoDB para aumentar o valor de `MaxWriteRequestUnits` para o GSI afetado e, em seguida, monitore e faça ajustes.
- Remova o limite máximo do GSI: defina `MaxWriteRequestUnits` como `-1` para o GSI a fim de remover o teto, permitindo o ajuste de escala com base na demanda até atingir as cotas de throughput em nível de conta. Isso remove seu limite personalizado, mas ainda mantém as proteções em nível de conta da AWS. No entanto, é importante monitorar os gastos rigorosamente depois de remover esse limite.

Diagnóstico e monitoramento comuns

Ao solucionar problemas de controle de utilização de throughput máximo sob demanda excedido, várias métricas do CloudWatch podem ajudar a identificar a causa raiz e os padrões de ajuste de escala.

Métricas essenciais do CloudWatch

Monitore estas métricas essenciais para diagnosticar o controle de utilização de throughput máximo sob demanda excedido:

- Eventos de controle de utilização de throughput máximo: `ReadMaxOnDemandThroughputThrottleEvents` e `WriteMaxOnDemandThroughputThrottleEvents` rastreiam quando as solicitações sofrem controle de utilização por excederem os limites máximos. `ReadThrottleEvents` e `WriteThrottleEvents` monitoram quando qualquer solicitação de leitura ou gravação excede a capacidade provisionada.
- Throughput máximo atual configurado para uma tabela ou um índice secundário global: `OnDemandMaxReadRequestUnits` e `OnDemandMaxWriteRequestUnits` mostram os limites de capacidade máxima atuais.
- Consumo real de capacidade: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` mostram os padrões reais de uso.

Abordagem de análise

Siga estas etapas para confirmar o diagnóstico de throughput máximo sob demanda excedido:

1. Compare a capacidade consumida com os limites máximos de capacidade. Verifique se o consumo se aproxima ou excede consistentemente os limites máximos.
2. Analise a frequência e o momento dos eventos de controle de utilização para identificar padrões. Procure aumentos repentinos na capacidade consumida que coincidam com o respectivo evento de controle de utilização.
3. Use o CloudWatch Contributor Insights para identificar quais itens ou chaves de partição consomem mais capacidade.

Esclarecimentos sobre o controle de utilização de gravação e a contrapressão do índice secundário global (GSI) no DynamoDB

O controle de utilização de contrapressão do GSI representa um dos cenários de controle de utilização mais complexos no DynamoDB porque cria uma relação indireta entre as operações de gravação e o controle. A aplicação grava em uma tabela base, mas sofre controle devido a restrições de capacidade em um ou vários índices.

Esclarecimentos sobre o controle de utilização de contrapressão do GSI

Quando você grava uma tabela no DynamoDB, quaisquer índices secundários globais (GSIs) nessa tabela são atualizados de forma assíncrona usando um modelo com consistência final. Se um GSI não tiver capacidade suficiente para lidar com essas atualizações, o DynamoDB limitará as gravações na tabela base para manter a consistência de dados. Isso é chamado de contrapressão do GSI. Para ter mais informações sobre como os GSIs funcionam, consulte Global Secondary Indexes in DynamoDB.

Ao contrário controle de utilização direto da tabela, em que o recurso que está sendo acessado também é o recurso que causa o controle, a contrapressão do GSI cria uma dependência entre a tabela base e os respectivos índices. Mesmo que a tabela base tenha ampla capacidade, as gravações sofrerão controle de utilização se algum GSI associado não conseguir lidar com o volume de atualização. É particularmente importante entender essa relação porque as restrições em nível de partição se aplicam de forma independente à tabela base e a cada GSI. Cada um tem sua própria estrutura de partição e limites de throughput correspondentes.

O particionamento do GSI é baseado na chave de partição do GSI, que geralmente é diferente da chave de partição da tabela base. Mesmo que o acesso à tabela base esteja perfeitamente distribuído entre partições, as atualizações do GSI ainda assim podem se concentrar em partições específicas, criando pontos de acesso frequente no GSI. Para ver as práticas recomendadas gerais sobre design de chave de partição para tabelas e GSIs, consulte DynamoDB partition key design.

Por exemplo, se a tabela base usa `customerId` como chave de partição (distribuída uniformemente), mas o GSI usa `status` como chave de partição (com valores possíveis limitados, como “ativo”, “pendente” e “fechado”), as atualizações em itens com valores de status conhecidos podem criar partições com acesso frequente (quentes) do GSI mesmo quando o acesso à tabela base está equilibrado. Isso cria um cenário particularmente desafiador, em que a aplicação pode sofrer controle de utilização devido a partições com acesso frequente do GSI, mesmo que a tabela base e o GSI tenham capacidade geral suficiente e o padrão de acesso da tabela base pareça bem distribuído.

Embora a exceção de controle de utilização aponte para o GSI (por meio do `ResourceArn`), a operação real que está sofrendo controle de utilização é a gravação na tabela base. Isso pode ser confuso porque a aplicação está gravando na tabela base e recebendo uma exceção relacionada ao GSI.

Tipos de controle de utilização do GSI

O controle de utilização de contrapressão do GSI manifesta-se por meio de diferentes tipos de exceção, dependendo da restrição específica de capacidade:

- Capacidade provisionada do GSI excedida: ocorre quando o GSI não tem unidades de capacidade de gravação suficientes para lidar com as atualizações das operações da tabela base. Isso gera uma `ProvisionedThroughputExceededException` com o motivo `IndexWriteProvisionedThroughputExceeded`, e o `ResourceArn` aponta diretamente para o GSI específico que está enfrentando restrições de capacidade.
- Throughput máximo sob demanda do GSI excedido: ocorre quando as operações de gravação do GSI ultrapassam os limites máximos configurados nas tabelas sob demanda. Isso gera uma `ThrottlingException` com o motivo `IndexWriteMaxOnDemandThroughputExceeded`, identificando o GSI específico com restrições de throughput configuradas.
- Limites de partição do GSI excedidos: ocorre quando partições individuais do GSI excedem os respectivos limites de throughput (partições com acesso frequente), mesmo que a capacidade geral do GSI pareça suficiente. Isso gera um `ThrottlingException` com o motivo `IndexWriteKeyRangeThroughputExceeded`, indicando problemas de partição com acesso

frequente no GSI específico identificado no `ResourceArn`. Isso é particularmente importante porque a distribuição de partições do GSI pode diferir significativamente da distribuição de partições da tabela base, criando pontos de acesso frequente no GSI mesmo quando o acesso à tabela base é distribuído uniformemente.

- Limites da conta do GSI excedidos: esses limites são acionados quando as operações de gravação em um GSI específico excedem os limites regionais de throughput por tabela (ou de qualquer GSI individual dentro dessa tabela) definidos no nível da conta. O DynamoDB exibe uma `ThrottlingException` com o motivo `IndexWriteAccountLimitExceeded`, apontando para o GSI que ultrapassou os limites de uso da conta. Esse controle de utilização ocorre de forma independente para cada GSI que ultrapassa o limite. Para ter informações sobre cotas por tabela, por conta, regionais e de serviço, consulte the section called “Cotas”.

Métricas de controle de utilização do CloudWatch

Esta página oferece um guia abrangente de métricas do CloudWatch projetado especificamente para ajudar você a identificar, diagnosticar e resolver problemas de controle de utilização em tabelas e índices do DynamoDB.

Métricas gerais de controle de utilização

- `ThrottledRequests`
  - São acrescidas de um quando qualquer evento em uma solicitação sofre controle de utilização, independentemente de quantos eventos individuais sejam controlados nessa solicitação. Por exemplo, quando se atualiza um item em uma tabela com índices secundários globais (GSIs), ocorrem vários eventos: uma operação de gravação na tabela base e uma operação de gravação em cada índice. Se algum desses eventos individuais sofrer controle de utilização, a métrica `ThrottledRequests` será aumentada apenas uma vez.

    É importante entender esse comportamento ao monitorar e solucionar problemas de desempenho do DynamoDB, pois ele pode mascarar a verdadeira extensão do controle de utilização. Para ter informações mais abrangentes, compare a métrica `ThrottledRequests` com as métricas específicas em nível de evento, como `ReadThrottleEvents` e `WriteThrottleEvents`, e as métricas direcionadas, como `ReadKeyRangeThroughputThrottleEvents`. A lista completa dessas métricas com causa específica está disponível nesta página. Cada métrica corresponde a motivos específicos de controle de utilização que são capturados na exceção de controle. Para obter

orientação sobre como recuperar e interpretar esses motivos durante eventos de controle de utilização, consulte a seção the section called “Diagnosticar o controle de utilização”, que apresenta instruções para identificar e resolver as causas básicas dos problemas de controle de utilização.

- `ReadThrottleEvents`
  - Observe as solicitações que excedem o RCU provisionado para uma tabela ou GSI.
- `WriteThrottleEvents`
  - Observe as solicitações que excedem o WCU provisionado para uma tabela ou GSI.

Métricas detalhadas de controle de utilização por causa

Controle de utilização de throughput sob demanda

- `ReadMaxOnDemandThroughputThrottleEvents`
  - Número de solicitações de leitura com controle de utilização devido a throughput máximo sob demanda.
- `WriteMaxOnDemandThroughputThrottleEvents`
  - Número de solicitações de gravação com controle de utilização devido a throughput máximo sob demanda.

Controle de utilização em nível de conta

- `ReadAccountLimitThrottleEvents`
  - Número de solicitações de leitura com controle de utilização devido aos limites da conta.
- `WriteAccountLimitThrottleEvents`
  - Número de solicitações de gravação com controle de utilização devido aos limites da conta.

Controle de utilização em nível de partição

- `ReadKeyRangeThroughputThrottleEvents`
  - Número de solicitações de leitura com controle de utilização devido aos limites de partição.
- `WriteKeyRangeThroughputThrottleEvents`
  - Número de solicitações de gravação com controle de utilização devido aos limites de partição.

Métricas de análise de capacidade

- `OnlineIndexConsumedWriteCapacity`

- Quando você adiciona um novo GSI a uma tabela existente, o DynamoDB executa uma operação de preenchimento que copia os dados da tabela base para o novo índice. Esse processo consome unidades de capacidade de gravação. A métrica `OnlineIndexConsumedWriteCapacity` rastreia esse consumo específico.

  Esse consumo é separado e adicional às operações de gravação regulares monitoradas por `ConsumedWriteCapacityUnits`. A métrica regular `ConsumedWriteCapacityUnits` para um GSI não inclui a capacidade de gravação consumida durante a criação inicial do índice.

- `ProvisionedReadCapacityUnits` e `ProvisionedWriteCapacityUnits`
  - Veja quantas unidades de capacidade de leitura ou gravação provisionadas foram consumidas durante o período especificado, para uma tabela ou um índice secundário global especificado.
  - Observe que a dimensão `TableName` retorna `ProvisionedReadCapacityUnits` para a tabela somente por padrão. Para visualizar o número de unidades de capacidade de leitura ou de gravação provisionadas para um índice secundário global, você deve especificar `TableName` e `GlobalSecondaryIndexName`.
- `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits`
  - Veja quantas unidades de capacidade de leitura ou gravação foram consumidas durante o período especificado. `ConsumedWriteCapacityUnits` não inclui a capacidade de gravação consumida durante o processo inicial de criação do índice.

Para ter mais informações sobre métricas do DynamoDB CloudWatch, consulte Métricas e dimensões do DynamoDB.

Apêndice do DynamoDB

Tópicos

- Solução de problemas de estabelecimento de conexão SSL/TLS com o DynamoDB
- Exemplos de tabelas e dados para uso no DynamoDB
- Criar exemplos de tabelas e fazer upload de dados no DynamoDB
- Exemplo de aplicação do DynamoDB usando o AWS SDK for Python (Boto): Jogo da velha
- Palavras reservadas no DynamoDB
- Exemplos do AWS SDK for Java 1.x
- Exemplos do SDK da AWS para Go 1.x
- Exemplos do SDK da AWS para Node.js 2.x

Solução de problemas de estabelecimento de conexão SSL/TLS com o DynamoDB

O Amazon DynamoDB está meio ao processo de migração de nossos endpoints para certificados confiáveis assinados pela autoridade de certificação Amazon Trust Services (ATS), em vez de uma autoridade de certificação externa. Em dezembro de 2017, lançamos na região EU-WEST-3 (Paris) os certificados confiáveis emitidos pela Amazon Trust Services. Todas as novas regiões lançadas depois de dezembro de 2017 têm endpoints com certificados emitidos pela Amazon Trust Services. Este guia mostra como você valida e resolve problemas de conexão SSL/TLS.

Como testar seu aplicativo ou serviço

A maioria dos SDKs e interfaces de linha de comando (CLIs) da AWS é compatível com a autoridade de certificação Amazon Trust Services. Se estiver usando uma versão do AWS SDK for Python ou CLI lançada antes de 29 de outubro de 2013, você deverá atualizá-la. Os SDKs e CLIs para .NET, Java, PHP, Go, JavaScript e C++ não contêm nenhum certificado. Os certificados provém do sistema operacional subjacente. O SDK para Ruby tem incluído pelo menos uma das CAs exigidas desde o 10 de junho de 2015. Antes dessa data, o SDK para Ruby V2 não continha certificados. Caso você use uma versão não compatível, personalizada ou alterada do AWS SDK ou caso use um armazenamento personalizado de confiança, é provável que não precise da compatibilidade necessária para a autoridade de certificação Amazon Trust Services.

Para validar o acesso aos endpoints do DynamoDB, você precisará desenvolver um teste que acesse a API do DynamoDB ou a API do DynamoDB Streams na região EU-WEST-3 e validar se o handshake do TLS tem êxito. Os endpoints específicos que você precisará acessar nesse teste são:

- DynamoDB: https://dynamodb.eu-west-3.amazonaws.com
- DynamoDB Streams: https://streams.dynamodb.eu-west-3.amazonaws.com

Se seu aplicativo não for compatível com a autoridade de certificação Amazon Trust Services, você verá uma das falhas a seguir:

- Erros de negociação de SSL/TLS
- Um longo atraso antes de o software receber um erro, o que indica uma falha de negociação de SSL/TLS. O tempo de atraso depende da estratégia de repetição e de configuração de tempo limite do cliente.

Como testar o navegador cliente

Para verificar se seu navegador consegue se conectar ao Amazon DynamoDB, abra o URL a seguir: https://dynamodb.eu-west-3.amazonaws.com. Se o teste for bem-sucedido, você verá uma mensagem como esta:

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

Se o teste não for bem-sucedido, será exibido um erro semelhante a este: https://untrusted-root.badssl.com/.

Como atualizar o aplicativo de software cliente

Os aplicativos que acessam endpoints de API do DynamoDB ou DynamoDB Streams (seja por meio de navegadores ou programaticamente) precisarão atualizar a lista de CA confiável nas máquinas clientes se ainda não forem compatíveis com as CAs a seguir:

- Amazon Root CA 1
- Starfield Services Root Certificate Authority – G2
- Starfield Class 2 Certification Authority

Se os clientes já confiarem em QUALQUER uma das três CAs acima, confiarão nos certificados usados pelo DynamoDB e nenhuma ação será necessária. Contudo, se os clientes ainda não confiarem em nenhuma das CAs acima, as conexões HTTPS com as APIs do DynamoDB ou do DynamoDB Streams falharão. Para obter mais informações, visite esta publicação de blog: https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its-own-certificate-authority/.

Como atualizar o navegador cliente

Você pode atualizar o pacote de certificado em seu navegador simplesmente atualizando seu navegador. Instruções para os navegadores mais comuns podem ser encontradas no site do respectivo navegador:

- Chrome: https://support.google.com/chrome/answer/95414?hl=en
- Firefox: https://support.mozilla.org/en-US/kb/update-firefox-latest-version
- Safari: https://support.apple.com/en-us/HT204416
- Internet Explorer: https://support.microsoft.com/en-us/help/17295/windows-internet-explorer-which-version#ie=other

Como atualizar manualmente seu pacote de certificado

Se você não conseguir acessar a API do DynamoDB ou do DynamoDB Streams, precisará atualizar o pacote de certificado. Para isso, você precisará importar pelo menos uma das CAs exigidas. Você pode encontrá-las em https://www.amazontrust.com/repository/.

Os seguintes sistemas operacionais e linguagens de programação são compatíveis com os certificados da Amazon Trust Services:

- Versões do Microsoft Windows com atualizações em janeiro de 2005 ou mais recentes instaladas, Windows Vista, Windows 7, Windows Server 2008 e versões mais recentes.
- MacOS X 10.4 com Java para MacOS X 10.4 Release 5, MacOS X 10.5 e versões mais recentes.
- Red Hat Enterprise Linux 5 (março de 2007), Linux 6 e Linux 7 e CentOS 5, CentOS 6 e CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (todas as versões)
- Java 1.4.2_12, Java 5 atualização 2 e todas as versões mais recentes, incluindo Java 6, Java 7 e Java 8

Se ainda assim não conseguir se conectar, consulte a documentação do software ou o fornecedor do sistema operacional ou entre em contato com o AWS Support em https://aws.amazon.com/support para receber assistência adicional.

Exemplos de tabelas e dados para uso no DynamoDB

O Guia do desenvolvedor do Amazon DynamoDB usa tabelas de exemplo para ilustrar diversos aspectos do DynamoDB.

Nome da tabela	Chave primária
ProductCatalog	Chave primária simples: • `Id` (telefone)
Forum	Chave primária simples: • `Name` (String)
Thread	Chave primária composta: • `ForumName` (String) • `Subject` (String)
Reply	Chave primária composta: • `Id` (String) • `ReplyDateTime` (String)

A tabela Reply tem um índice global secundário chamado PostedBy-Message-Index. Esse índice facilitará as consultas em dois atributos que não são de chave da tabela Reply.

Nome do índice	Chave primária
PostedBy-Message-Index	Chave primária composta: • `PostedBy` (String) • `Message` (String)

Para obter mais informações sobre essas tabelas, consulte Etapa 1: crie uma tabela no DynamoDB e Etapa 2: grave dados em uma tabela do DynamoDB.

Arquivos de dados de amostra

Tópicos

- ProductCatalog - dados de amostra
- Forum - dados de amostra
- Thread - dados de amostra
- Reply - dados de amostra

As seções a seguir mostram os arquivos de dados de exemplo que são usados para carregar as tabelas ProductCatalog, Forum, Thread e Reply.

Cada arquivo de dados contém vários elementos `PutRequest`, cada um dos quais contêm um único item. Esses elementos `PutRequest` são usados como entrada para a operação `BatchWriteItem`, usando AWS Command Line Interface (AWS CLI).

ProductCatalog - dados de amostra

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "101"
                    },
                    "Title": {
                        "S": "Book 101 Title"
                    },
                    "ISBN": {
                        "S": "111-1111111111"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            }
                        ]
```

```
                },
                "Price": {
                    "N": "2"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 0.5"
                },
                "PageCount": {
                    "N": "500"
                },
                "InPublication": {
                    "BOOL": true
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "102"
                },
                "Title": {
                    "S": "Book 102 Title"
                },
                "ISBN": {
                    "S": "222-2222222222"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        },
                        {
                            "S": "Author2"
                        }
                    ]
                },
                "Price": {
                    "N": "20"
                },
```

```
                "Dimensions": {
                    "S": "8.5 x 11.0 x 0.8"
                },
                "PageCount": {
                    "N": "600"
                },
                "InPublication": {
                    "BOOL": true
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "103"
                },
                "Title": {
                    "S": "Book 103 Title"
                },
                "ISBN": {
                    "S": "333-3333333333"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        },
                        {
                            "S": "Author2"
                        }
                    ]
                },
                "Price": {
                    "N": "2000"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 1.5"
                },
                "PageCount": {
```

```
                    "N": "600"
                },
                "InPublication": {
                    "BOOL": false
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "201"
                },
                "Title": {
                    "S": "18-Bike-201"
                },
                "Description": {
                    "S": "201 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Mountain A"
                },
                "Price": {
                    "N": "100"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
```

```
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "202"
                },
                "Title": {
                    "S": "21-Bike-202"
                },
                "Description": {
                    "S": "202 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company A"
                },
                "Price": {
                    "N": "200"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
```

```
                "Id": {
                    "N": "203"
                },
                "Title": {
                    "S": "19-Bike-203"
                },
                "Description": {
                    "S": "203 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company B"
                },
                "Price": {
                    "N": "300"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "204"
                },
                "Title": {
```

```
                    "S": "18-Bike-204"
                },
                "Description": {
                    "S": "204 Description"
                },
                "BicycleType": {
                    "S": "Mountain"
                },
                "Brand": {
                    "S": "Brand-Company B"
                },
                "Price": {
                    "N": "400"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "205"
                },
                "Title": {
                    "S": "18-Bike-204"
                },
                "Description": {
                    "S": "205 Description"
                },
                "BicycleType": {
                    "S": "Hybrid"
                },
                "Brand": {
                    "S": "Brand-Company C"
```

```
                    },
                    "Price": {
                        "N": "500"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        }
    ]
}
```

Forum - dados de amostra

```
{
    "Forum": [
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon DynamoDB"},
                    "Category": {"S":"Amazon Web Services"},
                    "Threads": {"N":"2"},
                    "Messages": {"N":"4"},
                    "Views": {"N":"1000"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon S3"},
                    "Category": {"S":"Amazon Web Services"}
```

```
                }
            }
        }
    ]
}
```

Thread - dados de amostra

```
{
    "Thread": [
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 1"
                    },
                    "Message": {
                        "S": "DynamoDB thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-22T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "index"
                            },
                            {
```

```
                            "S": "primarykey"
                        },
                        {
                            "S": "table"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon DynamoDB"
                },
                "Subject": {
                    "S": "DynamoDB Thread 2"
                },
                "Message": {
                    "S": "DynamoDB thread 2 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-15T19:58:22.514Z"
                },
                "Views": {
                    "N": "3"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
                    "N": "0"
                },
                "Tags": {
                    "L": [
                        {
                            "S": "items"
                        },
                        {
                            "S": "attributes"
```

```
                        },
                        {
                            "S": "throughput"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon S3"
                },
                "Subject": {
                    "S": "S3 Thread 1"
                },
                "Message": {
                    "S": "S3 thread 1 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-29T19:58:22.514Z"
                },
                "Views": {
                    "N": "0"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
                    "N": "0"
                },
                "Tags": {
                    "L": [
                        {
                            "S": "largeobjects"
                        },
                        {
                            "S": "multipart upload"
                        }
```

```
                        ]
                    }
                }
            }
        }
    ]
}
```

Reply - dados de amostra

```
{
    "Reply": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-15T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-22T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 2 text"
                    },
                    "PostedBy": {
```

```
                        "S": "User B"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-29T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-10-05T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 2 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        }
    ]
}
```

Criar exemplos de tabelas e fazer upload de dados no DynamoDB

Este apêndice fornece o código para criar as tabelas e adicionar dados programaticamente.

Tópicos

- Criar exemplos de tabelas e carregar dados usando o AWS SDK para Java
- Criar exemplos de tabelas e carregar dados usando o AWS SDK para .NET

Criar exemplos de tabelas e carregar dados usando o AWS SDK para Java

O seguinte exemplo de código Java cria tabelas e carrega dados nelas. Para obter instruções passo a passo para executar esse código usando o Eclipse, consulte Exemplos de código Java.

```
package com.amazonaws.codesamples;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class CreateTablesLoadData {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);
```

```
    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    static String productCatalogTableName = "ProductCatalog";
    static String forumTableName = "Forum";
    static String threadTableName = "Thread";
    static String replyTableName = "Reply";

    public static void main(String[] args) throws Exception {

        try {

            deleteTable(productCatalogTableName);
            deleteTable(forumTableName);
            deleteTable(threadTableName);
            deleteTable(replyTableName);

            // Parameter1: table name
            // Parameter2: reads per second
            // Parameter3: writes per second
            // Parameter4/5: partition key and data type
            // Parameter6/7: sort key and data type (if applicable)

            createTable(productCatalogTableName, 10L, 5L, "Id", "N");
            createTable(forumTableName, 10L, 5L, "Name", "S");
            createTable(threadTableName, 10L, 5L, "ForumName", "S", "Subject", "S");
            createTable(replyTableName, 10L, 5L, "Id", "S", "ReplyDateTime", "S");

            loadSampleProducts(productCatalogTableName);
            loadSampleForums(forumTableName);
            loadSampleThreads(threadTableName);
            loadSampleReplies(replyTableName);

        } catch (Exception e) {
            System.err.println("Program failed:");
            System.err.println(e.getMessage());
        }
        System.out.println("Success.");
    }

    private static void deleteTable(String tableName) {
        Table table = dynamoDB.getTable(tableName);
        try {
```

```
            System.out.println("Issuing DeleteTable request for " + tableName);
            table.delete();
            System.out.println("Waiting for " + tableName + " to be deleted...this may
take a while...");
            table.waitForDelete();

        } catch (Exception e) {
            System.err.println("DeleteTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType) {

        createTable(tableName, readCapacityUnits, writeCapacityUnits, partitionKeyName,
partitionKeyType, null, null);
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType, String sortKeyName,
String sortKeyType) {

        try {

            ArrayList<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
            keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition

                            // key

            ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
            attributeDefinitions
                    .add(new AttributeDefinition().withAttributeName(partitionKeyName)
                            .withAttributeType(partitionKeyType));

            if (sortKeyName != null) {
                keySchema.add(new
KeySchemaElement().withAttributeName(sortKeyName).withKeyType(KeyType.RANGE)); // Sort
```

```
                        // key
                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName(sortKeyName).withAttributeType(sortKeyType));
            }

            CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)
                    .withProvisionedThroughput(new
ProvisionedThroughput().withReadCapacityUnits(readCapacityUnits)
                            .withWriteCapacityUnits(writeCapacityUnits));

            // If this is the Reply table, define a local secondary index
            if (replyTableName.equals(tableName)) {

                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName("PostedBy").withAttributeType("S"));

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();
                localSecondaryIndexes.add(new
LocalSecondaryIndex().withIndexName("PostedBy-Index")
                        .withKeySchema(
                                new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH), //
Partition

                                // key
                                new
KeySchemaElement().withAttributeName("PostedBy").withKeyType(KeyType.RANGE)) // Sort

                        // key
                        .withProjection(new
Projection().withProjectionType(ProjectionType.KEYS_ONLY)));

                request.setLocalSecondaryIndexes(localSecondaryIndexes);
            }

            request.setAttributeDefinitions(attributeDefinitions);

            System.out.println("Issuing CreateTable request for " + tableName);
            Table table = dynamoDB.createTable(request);
```

```
            System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
            table.waitForActive();

        } catch (Exception e) {
            System.err.println("CreateTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleProducts(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Id", 101).withString("Title", "Book
101 Title")
                    .withString("ISBN", "111-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1"))).withNumber("Price", 2)
                    .withString("Dimensions", "8.5 x 11.0 x
0.5").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 102).withString("Title", "Book 102
Title")
                    .withString("ISBN", "222-2222222222")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1", "Author2")))
                    .withNumber("Price", 20).withString("Dimensions", "8.5 x 11.0 x
0.8").withNumber("PageCount", 600)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 103).withString("Title", "Book 103
Title")
                    .withString("ISBN", "333-3333333333")
```

```
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author1", "Author2")))
                    // Intentional. Later we'll run Scan to find price error. Find
                    // items > 1000 in price.
                    .withNumber("Price", 2000).withString("Dimensions", "8.5 x 11.0 x
 1.5").withNumber("PageCount", 600)
                    .withBoolean("InPublication", false).withString("ProductCategory",
 "Book");
            table.putItem(item);

            // Add bikes.

            item = new Item().withPrimaryKey("Id", 201).withString("Title", "18-
Bike-201")
                    // Size, followed by some title.
                    .withString("Description", "201
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Mountain A")
                    // Trek, Specialized.
                    .withNumber("Price", 100).withStringSet("Color", new
 HashSet<String>(Arrays.asList("Red", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 202).withString("Title", "21-
Bike-202")
                    .withString("Description", "202
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company A").withNumber("Price", 200)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Green",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 203).withString("Title", "19-
Bike-203")
                    .withString("Description", "203
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 300)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Green", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("Id", 204).withString("Title", "18-
Bike-204")
                    .withString("Description", "204
 Description").withString("BicycleType", "Mountain")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 400)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 205).withString("Title", "20-
Bike-205")
                    .withString("Description", "205
 Description").withString("BicycleType", "Hybrid")
                    .withString("Brand", "Brand-Company C").withNumber("Price", 500)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleForums(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Name", "Amazon DynamoDB")
                    .withString("Category", "Amazon Web
 Services").withNumber("Threads", 2).withNumber("Messages", 4)
                    .withNumber("Views", 1000);
            table.putItem(item);

            item = new Item().withPrimaryKey("Name", "Amazon
 S3").withString("Category", "Amazon Web Services")
                    .withNumber("Threads", 0);
            table.putItem(item);
```

```
        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleThreads(String tableName) {
        try {
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000); // 7
            // days
            // ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000); // 14
            // days
            // ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000); // 21
            // days
            // ago

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("ForumName", "Amazon DynamoDB")
                    .withString("Subject", "DynamoDB Thread 1").withString("Message",
"DynamoDB thread 1 message")
                    .withString("LastPostedBy", "User
A").withString("LastPostedDateTime", dateFormatter.format(date2))
                    .withNumber("Views", 0).withNumber("Replies",
0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"primarykey", "table")));
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("ForumName", "Amazon
DynamoDB").withString("Subject", "DynamoDB Thread 2")
                    .withString("Message", "DynamoDB thread 2
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date3)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"partitionkey", "sortkey")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
S3").withString("Subject", "S3 Thread 1")
                    .withString("Message", "S3 Thread 3
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date1)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new
HashSet<String>(Arrays.asList("largeobjects", "multipart upload")));
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleReplies(String tableName) {
        try {
            // 1 day ago
            long time0 = (new Date()).getTime() - (1 * 24 * 60 * 60 * 1000);
            // 7 days ago
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000);
            // 14 days ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000);
            // 21 days ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000);

            Date date0 = new Date();
            date0.setTime(time0);
```

```
            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            // Add threads.

            Item item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB
Thread 1")
                    .withString("ReplyDateTime", (dateFormatter.format(date3)))
                    .withString("Message", "DynamoDB Thread 1 Reply 1
text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 1")
                    .withString("ReplyDateTime", dateFormatter.format(date2))
                    .withString("Message", "DynamoDB Thread 1 Reply 2
text").withString("PostedBy", "User B");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date1))
                    .withString("Message", "DynamoDB Thread 2 Reply 1
text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date0))
                    .withString("Message", "DynamoDB Thread 2 Reply 2
text").withString("PostedBy", "User A");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
```

```
            System.err.println(e.getMessage());

        }
    }

}
```

Criar exemplos de tabelas e carregar dados usando o AWS SDK para .NET

O seguinte exemplo de código C# cria tabelas e carrega dados nelas. Para obter instruções passo a passo para executar esse código no Visual Studio, consulte Exemplos de código .NET.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class CreateTablesLoadData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                //DeleteAllTables(client);
                DeleteTable("ProductCatalog");
                DeleteTable("Forum");
                DeleteTable("Thread");
                DeleteTable("Reply");

                // Create tables (using the AWS SDK for .NET low-level API).
                CreateTableProductCatalog();
                CreateTableForum();
                CreateTableThread(); // ForumTitle, Subject */
                CreateTableReply();
```

```
            // Load data (using the .NET SDK document API)
            LoadSampleProducts();
            LoadSampleForums();
            LoadSampleThreads();
            LoadSampleReplies();
            Console.WriteLine("Sample complete!");
            Console.WriteLine("Press ENTER to continue");
            Console.ReadLine();
        }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void DeleteTable(string tableName)
    {
        try
        {
            var deleteTableResponse = client.DeleteTable(new DeleteTableRequest()
            {
                TableName = tableName
            });
            WaitTillTableDeleted(client, tableName, deleteTableResponse);
        }
        catch (ResourceNotFoundException)
        {
            // There is no such table.
        }
    }

    private static void CreateTableProductCatalog()
    {
        string tableName = "ProductCatalog";

        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Id",
                                AttributeType = "N"
                            }
                        },
```

```
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement
                                {
                                    AttributeName = "Id",
                                    KeyType = "HASH"
                                }
                            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void CreateTableForum()
        {
            string tableName = "Forum";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "Name",
                                    AttributeType = "S"
                                }
                            },
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement
                                {
                                    AttributeName = "Name", // forum Title
                                    KeyType = "HASH"
                                }
                            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
```

```
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableThread()
    {
        string tableName = "Thread";

        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "ForumName", // Hash attribute
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "Subject",
                                AttributeType = "S"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement
                            {
                                AttributeName = "ForumName", // Hash attribute
                                KeyType = "HASH"
                            },
                            new KeySchemaElement
                            {
                                AttributeName = "Subject", // Range attribute
                                KeyType = "RANGE"
                            }
                        },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
```

```
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableReply()
    {
        string tableName = "Reply";
        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Id",
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "ReplyDateTime",
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "PostedBy",
                                AttributeType = "S"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement()
                            {
                                AttributeName = "Id",
                                KeyType = "HASH"
                            },
                            new KeySchemaElement()
                            {
                                AttributeName = "ReplyDateTime",
                                KeyType = "RANGE"
                            }
                        },
            LocalSecondaryIndexes = new List<LocalSecondaryIndex>()
```

```
                                {
                                    new LocalSecondaryIndex()
                                    {
                                        IndexName = "PostedBy_index",


                                        KeySchema = new List<KeySchemaElement>() {
                                            new KeySchemaElement() {
                                                AttributeName = "Id", KeyType = "HASH"
                                            },
                                            new KeySchemaElement() {
                                                AttributeName = "PostedBy", KeyType =
"RANGE"
                                            }
                                        },
                                        Projection = new Projection() {
                                            ProjectionType = ProjectionType.KEYS_ONLY
                                        }
                                    }
                                },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void WaitTillTableCreated(AmazonDynamoDBClient client, string
tableName,
                                 CreateTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable.
            while (status != "ACTIVE")
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
```

```
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                // Try-catch to handle potential eventual-consistency issue.
                catch (ResourceNotFoundException)
                { }
            }
        }

        private static void WaitTillTableDeleted(AmazonDynamoDBClient client, string
tableName,
                             DeleteTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable
            try
            {
                while (status == "DELETING")
                {
                    System.Threading.Thread.Sleep(5000); // wait 5 seconds

                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
            }
```

```
            catch (ResourceNotFoundException)
            {
                // Table deleted.
            }
        }

        private static void LoadSampleProducts()
        {
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // ********** Add Books *********************
            var book1 = new Document();
            book1["Id"] = 101;
            book1["Title"] = "Book 101 Title";
            book1["ISBN"] = "111-1111111111";
            book1["Authors"] = new List<string> { "Author 1" };
            book1["Price"] = -2; // *** Intentional value. Later used to illustrate
scan.
            book1["Dimensions"] = "8.5 x 11.0 x 0.5";
            book1["PageCount"] = 500;
            book1["InPublication"] = true;
            book1["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book1);

            var book2 = new Document();

            book2["Id"] = 102;
            book2["Title"] = "Book 102 Title";
            book2["ISBN"] = "222-2222222222";
            book2["Authors"] = new List<string> { "Author 1", "Author 2" }; ;
            book2["Price"] = 20;
            book2["Dimensions"] = "8.5 x 11.0 x 0.8";
            book2["PageCount"] = 600;
            book2["InPublication"] = true;
            book2["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book2);

            var book3 = new Document();
            book3["Id"] = 103;
            book3["Title"] = "Book 103 Title";
            book3["ISBN"] = "333-3333333333";
            book3["Authors"] = new List<string> { "Author 1", "Author2", "Author
3" }; ;
            book3["Price"] = 2000;
            book3["Dimensions"] = "8.5 x 11.0 x 1.5";
```

```
            book3["PageCount"] = 700;
            book3["InPublication"] = false;
            book3["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book3);

            // ************ Add bikes. ******************
            var bicycle1 = new Document();
            bicycle1["Id"] = 201;
            bicycle1["Title"] = "18-Bike 201"; // size, followed by some title.
            bicycle1["Description"] = "201 description";
            bicycle1["BicycleType"] = "Road";
            bicycle1["Brand"] = "Brand-Company A"; // Trek, Specialized.
            bicycle1["Price"] = 100;
            bicycle1["Color"] = new List<string> { "Red", "Black" };
            bicycle1["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle1);

            var bicycle2 = new Document();
            bicycle2["Id"] = 202;
            bicycle2["Title"] = "21-Bike 202Brand-Company A";
            bicycle2["Description"] = "202 description";
            bicycle2["BicycleType"] = "Road";
            bicycle2["Brand"] = "";
            bicycle2["Price"] = 200;
            bicycle2["Color"] = new List<string> { "Green", "Black" };
            bicycle2["ProductCategory"] = "Bicycle";
            productCatalogTable.PutItem(bicycle2);

            var bicycle3 = new Document();
            bicycle3["Id"] = 203;
            bicycle3["Title"] = "19-Bike 203";
            bicycle3["Description"] = "203 description";
            bicycle3["BicycleType"] = "Road";
            bicycle3["Brand"] = "Brand-Company B";
            bicycle3["Price"] = 300;
            bicycle3["Color"] = new List<string> { "Red", "Green", "Black" };
            bicycle3["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle3);

            var bicycle4 = new Document();
            bicycle4["Id"] = 204;
            bicycle4["Title"] = "18-Bike 204";
            bicycle4["Description"] = "204 description";
            bicycle4["BicycleType"] = "Mountain";
```

```
            bicycle4["Brand"] = "Brand-Company B";
            bicycle4["Price"] = 400;
            bicycle4["Color"] = new List<string> { "Red" };
            bicycle4["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle4);

            var bicycle5 = new Document();
            bicycle5["Id"] = 205;
            bicycle5["Title"] = "20-Title 205";
            bicycle4["Description"] = "205 description";
            bicycle5["BicycleType"] = "Hybrid";
            bicycle5["Brand"] = "Brand-Company C";
            bicycle5["Price"] = 500;
            bicycle5["Color"] = new List<string> { "Red", "Black" };
            bicycle5["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle5);
        }

        private static void LoadSampleForums()
        {
            Table forumTable = Table.LoadTable(client, "Forum");

            var forum1 = new Document();
            forum1["Name"] = "Amazon DynamoDB"; // PK
            forum1["Category"] = "Amazon Web Services";
            forum1["Threads"] = 2;
            forum1["Messages"] = 4;
            forum1["Views"] = 1000;

            forumTable.PutItem(forum1);

            var forum2 = new Document();
            forum2["Name"] = "Amazon S3"; // PK
            forum2["Category"] = "Amazon Web Services";
            forum2["Threads"] = 1;

            forumTable.PutItem(forum2);
        }

        private static void LoadSampleThreads()
        {
            Table threadTable = Table.LoadTable(client, "Thread");

            // Thread 1.
```

```
            var thread1 = new Document();
            thread1["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread1["Subject"] = "DynamoDB Thread 1"; // Range attribute.
            thread1["Message"] = "DynamoDB thread 1 message text";
            thread1["LastPostedBy"] = "User A";
            thread1["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0));
            thread1["Views"] = 0;
            thread1["Replies"] = 0;
            thread1["Answered"] = false;
            thread1["Tags"] = new List<string> { "index", "primarykey", "table" };

            threadTable.PutItem(thread1);

            // Thread 2.
            var thread2 = new Document();
            thread2["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread2["Subject"] = "DynamoDB Thread 2"; // Range attribute.
            thread2["Message"] = "DynamoDB thread 2 message text";
            thread2["LastPostedBy"] = "User A";
            thread2["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0));
            thread2["Views"] = 0;
            thread2["Replies"] = 0;
            thread2["Answered"] = false;
            thread2["Tags"] = new List<string> { "index", "primarykey", "rangekey" };

            threadTable.PutItem(thread2);

            // Thread 3.
            var thread3 = new Document();
            thread3["ForumName"] = "Amazon S3"; // Hash attribute.
            thread3["Subject"] = "S3 Thread 1"; // Range attribute.
            thread3["Message"] = "S3 thread 3 message text";
            thread3["LastPostedBy"] = "User A";
            thread3["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7, 0,
0, 0));
            thread3["Views"] = 0;
            thread3["Replies"] = 0;
            thread3["Answered"] = false;
            thread3["Tags"] = new List<string> { "largeobjects", "multipart upload" };
            threadTable.PutItem(thread3);
        }
```

```
        private static void LoadSampleReplies()
        {
            Table replyTable = Table.LoadTable(client, "Reply");

            // Reply 1 - thread 1.
            var thread1Reply1 = new Document();
            thread1Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0)); // Range attribute.
            thread1Reply1["Message"] = "DynamoDB Thread 1 Reply 1 text";
            thread1Reply1["PostedBy"] = "User A";

            replyTable.PutItem(thread1Reply1);

            // Reply 2 - thread 1.
            var thread1reply2 = new Document();
            thread1reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0)); // Range attribute.
            thread1reply2["Message"] = "DynamoDB Thread 1 Reply 2 text";
            thread1reply2["PostedBy"] = "User B";

            replyTable.PutItem(thread1reply2);

            // Reply 3 - thread 1.
            var thread1Reply3 = new Document();
            thread1Reply3["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1Reply3["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
0, 0, 0)); // Range attribute.
            thread1Reply3["Message"] = "DynamoDB Thread 1 Reply 3 text";
            thread1Reply3["PostedBy"] = "User B";

            replyTable.PutItem(thread1Reply3);

            // Reply 1 - thread 2.
            var thread2Reply1 = new Document();
            thread2Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
attribute.
            thread2Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
0, 0, 0)); // Range attribute.
            thread2Reply1["Message"] = "DynamoDB Thread 2 Reply 1 text";
```

```
            thread2Reply1["PostedBy"] = "User A";


            replyTable.PutItem(thread2Reply1);

            // Reply 2 - thread 2.
            var thread2Reply2 = new Document();
            thread2Reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(1,
 0, 0, 0)); // Range attribute.
            thread2Reply2["Message"] = "DynamoDB Thread 2 Reply 2 text";
            thread2Reply2["PostedBy"] = "User A";

            replyTable.PutItem(thread2Reply2);
        }
    }
}
```

Exemplo de aplicação do DynamoDB usando o AWS SDK for Python (Boto): Jogo da velha

O Jogo da velha é uma aplicação Web de exemplo criada no Amazon DynamoDB. A aplicação usa o AWS SDK for Python (Boto) para fazer as chamadas do DynamoDB necessárias para armazenar os dados do jogo em uma tabela do DynamoDB e o framework da Web para Python chamado Flask para ilustrar o desenvolvimento completo da aplicação no DynamoDB, incluindo como modelar os dados. Ele também demonstra as melhores práticas quando se trata de modelagem de dados no DynamoDB, incluindo a tabela que você cria para a aplicação do jogo, a chave primária que você define, os índices adicionais necessários de acordo com as exigências da consulta e o uso de atributos de valor concatenados.

Você joga o aplicativo Jogo da velha na web da seguinte forma:

1. Você faz login na página inicial do aplicativo.
2. Em seguida, você convida outro usuário para jogar o jogo como o seu oponente.

   Até que outro usuário aceite o convite, o status do jogo permanece como `PENDING`. Depois que um oponente aceita o convite, o status do jogo muda para `IN_PROGRESS`.
3. O jogo começa depois que o oponente faz login e aceita o convite.

4. A aplicação armazena todas as movimentações e as informações de status do jogo em uma tabela do DynamoDB.
5. O jogo termina com uma vitória ou empate, o que define o status do jogo como `FINISHED`.

O exercício de criação do aplicativo de ponta a ponta é descrito em etapas:

- Etapa 1: implantar e testar localmente: nesta seção, você faz download, implanta e testa a aplicação em seu computador local. Você poderá criar as tabelas necessárias na versão para download do DynamoDB.
- Etapa 2: examinar o modelo de dados e os detalhes da implantação : esta seção primeiro descreve o modelo de dados em detalhes, incluindo os índices e o uso do atributo de valor concatenado. Em seguida, a seção explica como o aplicativo funciona.
- Etapa 3: implantar em produção usando o serviço DynamoDB: esta seção aborda as considerações de implantação em produção. Nesta etapa, você cria uma tabela usando o serviço Amazon DynamoDB e implanta a aplicação usando o AWS Elastic Beanstalk. Quando a aplicação está em produção, você também concede as permissões apropriadas para que a aplicação possa acessar a tabela do DynamoDB. As instruções desta seção orientam você durante a implantação da produção de ponta a ponta.
- Etapa 4: Limpar os recursos: esta seção destaca as áreas que não são cobertas por este exemplo. A seção também fornece as etapas necessárias para você remover os recursos da AWS que criou nas etapas anteriores a fim de evitar qualquer cobrança.

Etapa 1: implantar e testar localmente

Tópicos

- 1.1: baixar e instalar os pacotes obrigatórios
- 1.2: testar a aplicação de jogo

Nesta etapa, você faz download, implanta e testa o aplicativo Jogo da velha em seu computador local. Em vez de usar o serviço da Web Amazon DynamoDB, você fará download do DynamoDB para seu computador e criará a tabela necessária localmente.

1.1: baixar e instalar os pacotes obrigatórios

Para testar este aplicativo localmente, será necessário o seguinte:

- Python
- Flask (um microframework para Python)
- AWS SDK for Python (Boto)
- Execução do DynamoDB no computador
- Git

Para aproveitar essas ferramentas, faça o seguinte:

1. Instalar o Python. Para obter instruções detalhadas, acesse Fazer download do Python.

   O aplicativo Jogo da velha foi testando com a versão 2.7 do Python.
2. Instale o Flask e o AWS SDK for Python (Boto) usando o Python Package Installer (PIP):
   - Instale o PIP.

     Para obter instruções, consulte Instalar PIP. Na página de instalação, selecione o link get-pip.py e, em seguida, salve o arquivo. Em seguida, abra um terminal de comando como administrador e digite as informações a seguir no prompt de comando.

     ```
     python.exe get-pip.py
     ```

     No Linux, você não especifica a extensão `.exe`. Você só especifica `python get-pip.py`.
   - Usando o PIP, instale os pacotes Flask e Boto usando o código a seguir.

     ```
     pip install Flask
     pip install boto
     pip install configparser
     ```
3. Baixe o DynamoDB para o seu computador. Para obter instruções sobre como executá-lo, consulte Configurar o DynamoDB local (versão para download) .
4. Faça download do aplicativo Jogo da velha:
   a. Instale o Git. Para obter instruções, consulte downloads do git.
   b. Execute o código a seguir para fazer download da aplicação.

      ```
      git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
      ```

1.2: testar a aplicação de jogo

Para testar a aplicação Jogo da velha, é necessário executar o DynamoDB localmente no seu computador.

Para executar a aplicação Jogo da velha

1. Inicie o DynamoDB.
2. Inicie o servidor web do aplicativo Jogo da velha.

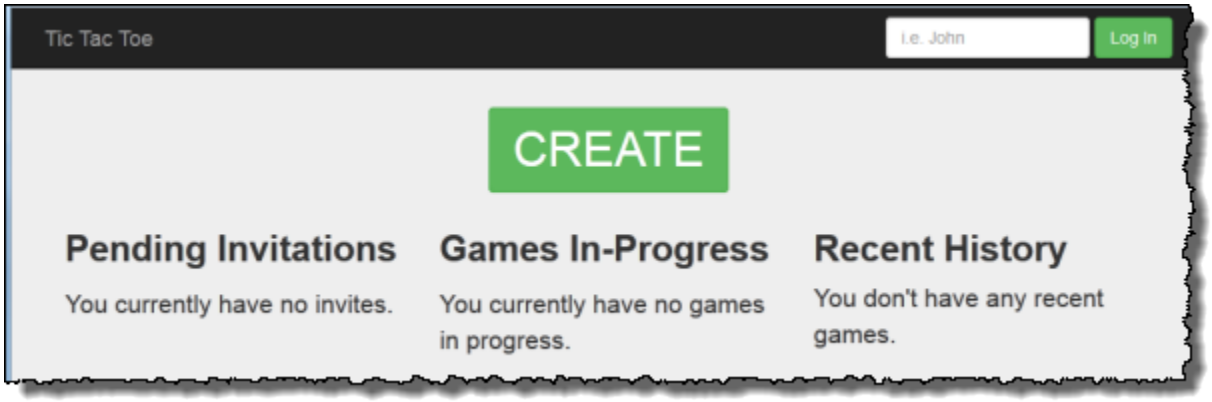
   Para isso, abra um terminal de comando, navegue para a pasta na qual você fez download do aplicativo jogo da velha e execute o aplicativo localmente usando o código a seguir.

   ```
   python.exe application.py --mode local --serverPort 5000 --port 8000
   ```

   No Linux, você não especifica a extensão `.exe`.
3. Abra seu navegador da web e digite as informações a seguir.

   ```
   http://localhost:5000/
   ```

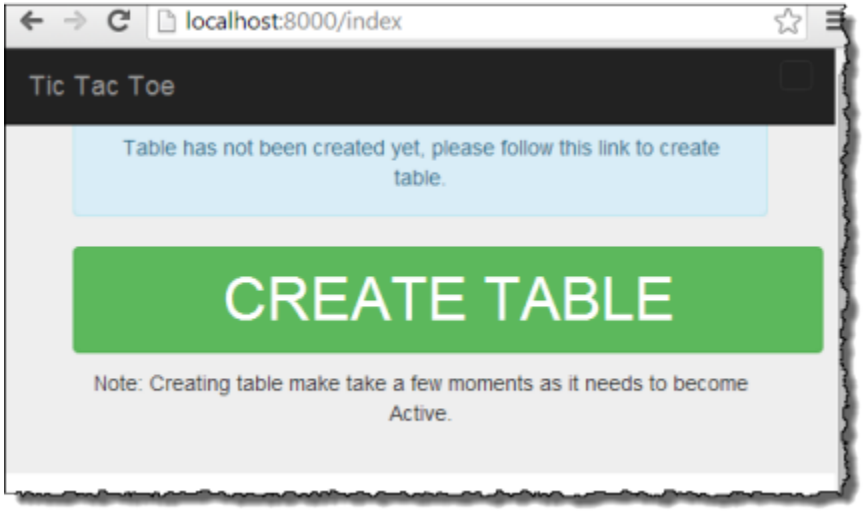
   O navegador mostra a página inicial.



4. Digite **`user1`** na caixa Log in (Login) para fazer login como user1.

> **Note**
>
> Este aplicativo de exemplo não realiza autenticação de usuário. O ID de usuário só é usado para identificar os jogadores. Se dois jogadores fizerem login com o mesmo alias, o aplicativo funciona como se você estivesse jogando em dois navegadores diferentes.

5. Se esta for a primeira vez que você está jogando, uma página solicitando a criação da tabela obrigatória (`Games`) no DynamoDB será mostrada. Selecione CREATE TABLE (Criar tabela).



6. Selecione CREATE (Criar) para criar o primeiro jogo da velha.
7. Digite **`user2`** na caixa Choose an Opponent (Escolher um oponente) e selecione Create Game! (Criar jogo!).

   Desse modo, o jogo é criado adicionando um item na tabela `Games`. Isso define o status do jogo como `PENDING`.

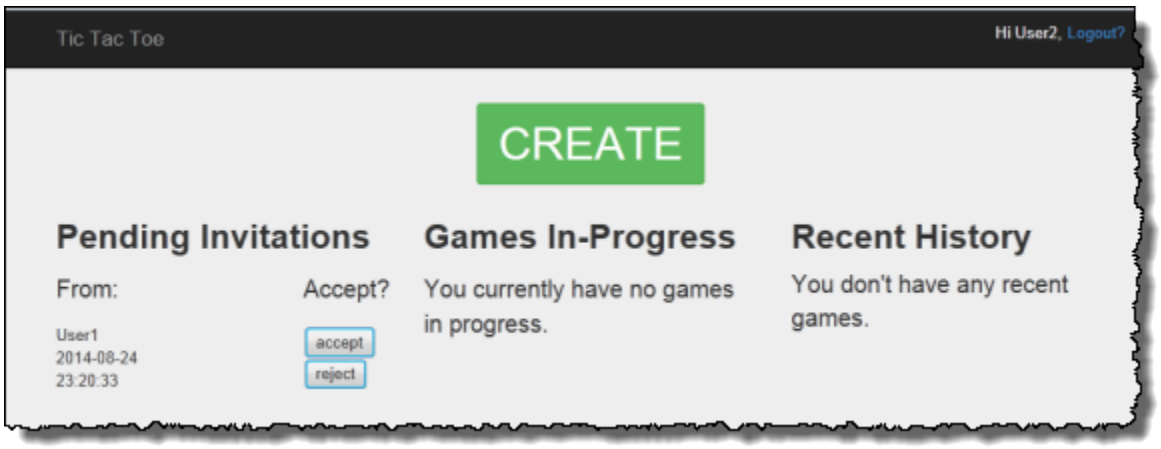
8. Abra outra janela do navegador e digite as informações a seguir.

```
http://localhost:5000/
```

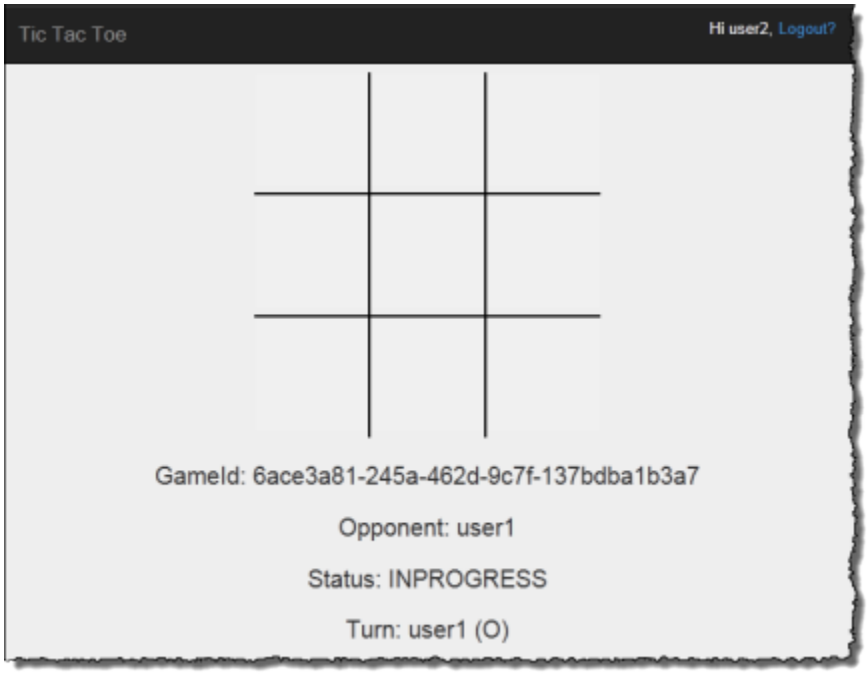
   O navegador passa informações por meio de cookies. Portanto, você deve usar o modo incognito ou navegação privada para que seus cookies não sejam estendidos.

9. Faça login como user2.

   É exibida uma página que mostra um convite pendente do user1.



10. Selecione accept (aceitar) para aceitar o convite.



    A página do jogo é exibida com uma grade de jogo da velha vazia. A página também mostra informações relevantes do jogo, como o ID do jogo, de quem é a vez e o status do jogo.

11. Jogue o jogo.

Para cada movimento do usuário, o serviço da Web envia uma solicitação para o DynamoDB atualizar condicionalmente o item do jogo na tabela `Games`. Por exemplo, as condições garantem que o movimento é válido, que o quadrado que o usuário escolheu está disponível e que era a vez do usuário que fez o movimento. Para um movimento válido, a operação de atualização adiciona um

novo atributo correspondente à seleção no quadro. A operação de atualização também define o valor do atributo existente para o usuário que pode fazer o próximo movimento.

Na página do jogo, a aplicação faz chamadas JavaScript assíncronas a cada segundo, por até 5 minutos, para verificar se o estado do jogo no DynamoDB foi alterado. Caso tenha sido, o aplicativo atualiza a página com novas informações. Depois de cinco minutos, o aplicativo para de fazer as solicitações e você precisa atualizar a página para obter informações atualizadas.

Etapa 2: examinar o modelo de dados e os detalhes da implantação

Tópicos

- 2.1: modelo de dados básico
- 2.2: aplicação em ação (demonstração do código)

2.1: modelo de dados básico

Esta aplicação de exemplo destaca os seguintes conceitos de modelo de dados do DynamoDB:

- Tabela: no DynamoDB, uma tabela é uma coleção de itens (ou seja, registros), e cada item é uma coleção de pares de nome-valor chamados atributos.

  Neste exemplo de Jogo da velha, o aplicativo armazena todos os dados do jogo em uma tabela, `Games`. O aplicativo cria um item na tabela por jogo e armazena todos os dados de jogos como atributos. Um jogo da velha pode ter até nove movimentações. Como as tabelas do DynamoDB não possuem um esquema em casos nos quais apenas a chave primária é o atributo obrigatório, a aplicação pode armazenar um número variável de atributos por item de jogo.

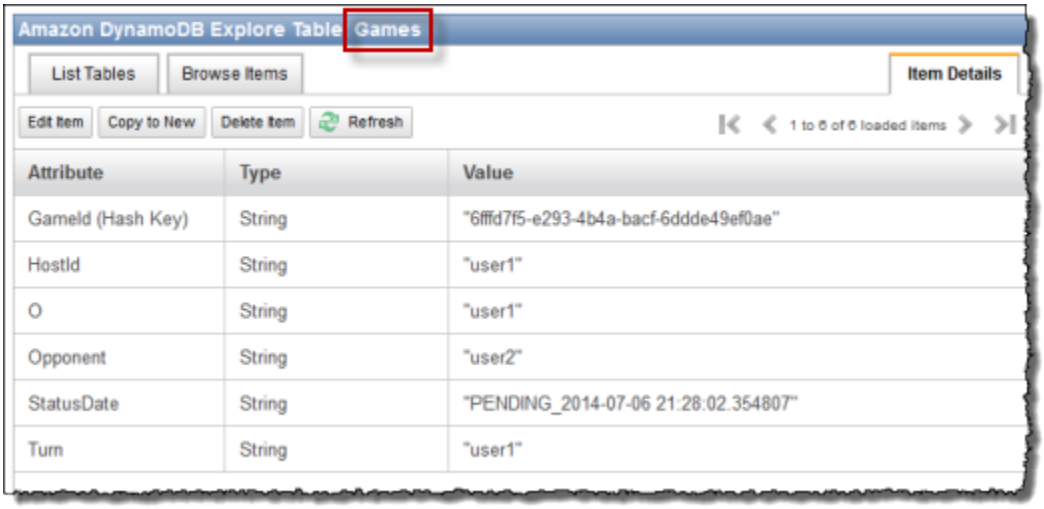
  A tabela `Games` possui uma chave primária simples composta de um atributo, `GameId`, do tipo string. O aplicativo atribui um ID exclusivo a cada jogo. Para obter mais informações sobre chaves primárias do DynamoDB, consulte Chave primária.

  Quando um usuário inicia um jogo da velha, convidando outro usuário para jogar, o aplicativo cria um novo item na tabela `Games` com atributos que armazenam metadados de jogos, como os seguintes:

  - `HostId`, o usuário que iniciou o jogo.
  - `Opponent`, o usuário que foi convidado para jogar.
  - O usuário que está na vez de jogar. O usuário que iniciou o jogo joga primeiro.

- O usuário que usa o símbolo O no quadro. O usuário que inicia os jogos usa o símbolo O.

Além disso, o aplicativo cria um atributo `StatusDate` concatenado, marcando o estado inicial do jogo como `PENDING`. A captura de tela a seguir mostra um item de exemplo como ele aparece no console do DynamoDB:



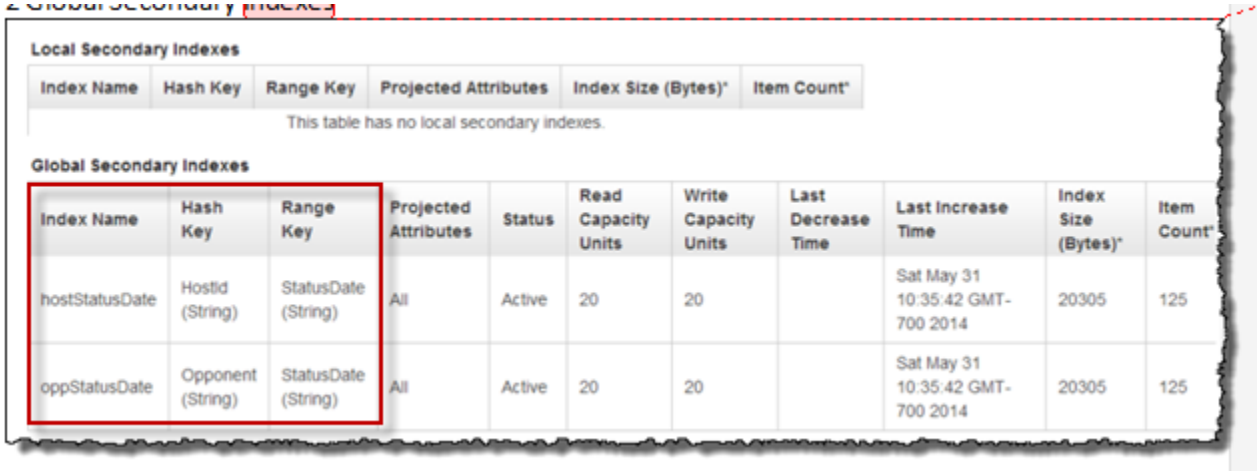
À medida que o jogo progride, o aplicativo adiciona um atributo à tabela para cada movimento do jogo. O nome do atributo é a posição no quadro, por exemplo `TopLeft` ou `BottomRight`. Por exemplo, um movimento pode ter um atributo `TopLeft` com o valor `0`, um atributo `TopRight` com o valor `0` e um atributo `BottomRight` com o valor `X`. O valor do atributo é `0` ou `X`, dependendo de qual usuário fez o movimento. Por exemplo, considere o quadro a seguir.

- Atributos de valor concatenados: o atributo `StatusDate` ilustra um atributo de valor concatenado. Em essa abordagem, em vez de criar atributos separados para armazenar o status do jogo

(PENDING, IN_PROGRESS e FINISHED) e a data (quando o último movimento foi feito), você pode combiná-los como um único atributo, por exemplo IN_PROGRESS_2014-04-30 10:20:32.

Em seguida, o aplicativo usa o atributo StatusDate na criação de índices secundários, especificando StatusDate como uma chave de classificação para o índice. A vantagem de usar o atributo de valor concatenado StatusDate é melhor demonstrada nos índices discutidos a seguir.

- Índices secundários globais: você pode usar a chave primária da tabela, GameId, para consultar a tabela com eficiência para encontrar um item do jogo. Para possibilitar a consulta de outros atributos que não sejam atributos da chave primária na tabela, o DynamoDB oferece suporte à criação de índices secundários. Neste aplicativo de exemplo, você cria os seguintes dois índices secundários:



Local Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Index Size (Bytes)*	Item Count*
This table has no local secondary indexes.					

Global Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Status	Read Capacity Units	Write Capacity Units	Last Decrease Time	Last Increase Time	Index Size (Bytes)*	Item Count*
hostStatusDate	HostId (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125
oppStatusDate	Opponent (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125

  - HostId-StatusDate-index. O índice tem HostId como chave de partição e StatusDate como chave de classificação. Você pode usar esse índice para consultar o HostId, por exemplo, para localizar jogos hospedados por um determinado usuário.
  - OpponentId-StatusDate-index. O índice tem OpponentId como chave de partição e StatusDate como chave de classificação. Você pode usar esse índice para consultar o Opponent, por exemplo, para localizar jogos nos quais um determinado usuário é o oponente.

  Esses índices são chamados de índices secundários globais porque a chave de partição nesses índices não é igual à chave de partição (GameId), usada na chave primária da tabela.

  Observe que ambos os índices especificam StatusDate como chave de classificação. Fazer isso permite o seguinte:

  - Você pode consultar usando o operador de comparação BEGINS_WITH. Por exemplo, você pode encontrar todos os jogos com o atributo IN_PROGRESS hospedados por um determinado

usuário. Neste caso, o operador `BEGINS_WITH` verifica o valor `StatusDate` que começa com `IN_PROGRESS`.

- O DynamoDB armazena os itens no índice em ordem classificada, por valor de chave de classificação. Portanto, se todos os prefixos de status forem os mesmos (por exemplo, `IN_PROGRESS`), o formato ISO usado para a parte de data terá itens classificados do mais antigo para o mais recente. Essa abordagem permite que determinadas consultas sejam executadas de forma eficiente, por exemplo, a seguinte:
  - Recupere até 10 dos jogos `IN_PROGRESS` mais recentes hospedados pelo usuário que está conectado. Para essa consulta, você especifica o índice `HostId-StatusDate-index`.
  - Recupere até 10 dos jogos `IN_PROGRESS` mais recentes nos quais o usuário conectado é o oponente. Para essa consulta, você especifica o índice `OpponentId-StatusDate-index`.

Para obter mais informações sobre índices secundários, consulte Melhorar o acesso aos dados com índices secundários no DynamoDB.

2.2: aplicação em ação (demonstração do código)

Este aplicativo tem duas páginas principais:

- Página inicial – esta página oferece ao usuário um login simples, um botão CRIAR para criar um novo jogo da velha, uma lista de jogos em andamento, o histórico de jogos e todos os convites de jogo pendentes ativos.

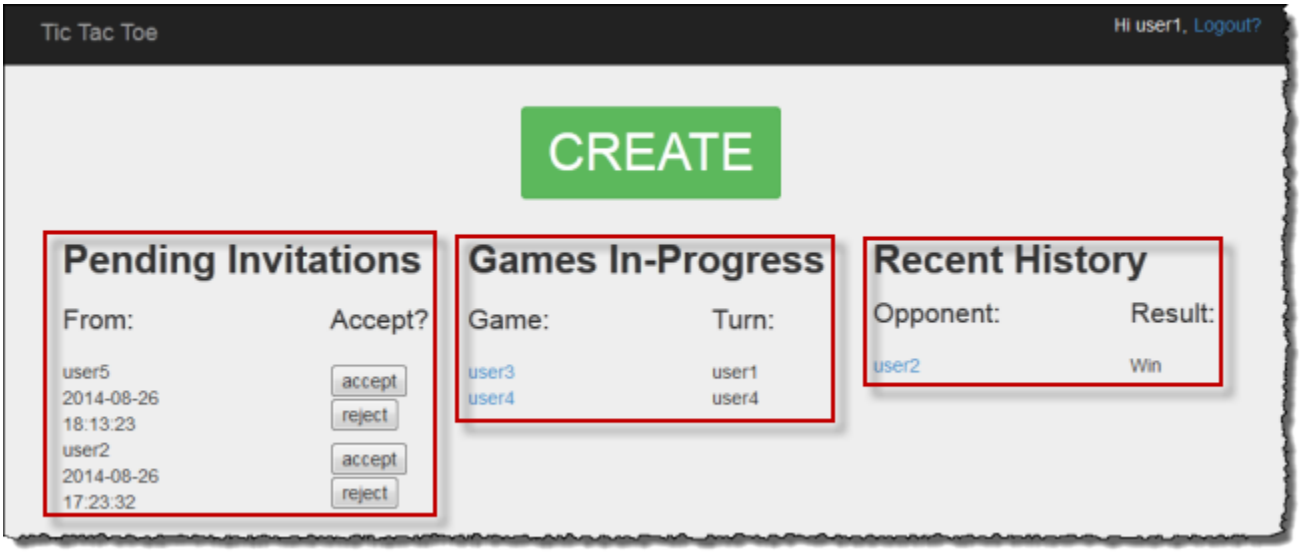
  A página inicial não é atualizada automaticamente; você deve atualizar a página para atualizar as listas.
- Página do jogo – Esta página mostra a grade do jogo da velha na qual os usuários jogam.

  O aplicativo atualiza a página de jogos automaticamente a cada segundo. O JavaScript no navegador chama o servidor web Python a cada segundo para consultar na tabela Games se os itens do jogo na tabela foram alterados. Se for o caso, o JavaScript aciona uma atualização da página para que o usuário veja o quadro atualizado.

Vamos ver em detalhes como o aplicativo funciona.

Home page (Página inicial)

Depois que o usuário fizer login, o aplicativo exibe as três listas de informações a seguir.



- Convites: esta lista mostra até 10 convites mais recentes de outros usuários que estão aguardando aceitação pelo usuário que está conectado. Na captura de tela anterior, o user1 tem convites de user5 e de user2 pendentes.
- Jogos em andamento: esta lista mostra até 10 jogos mais recentes que estão em andamento. Esses são os jogos que o usuário está ativamente jogando, que têm o status `IN_PROGRESS`. Na captura de tela, o user1 está jogando ativamente um jogo da velha com user3 e user4.
- Histórico recente: esta lista mostra até 10 jogos mais recentes que o usuário terminou, os quais tem o status `FINISHED`. No jogo apresentado na captura de tela, o user1 jogou anteriormente com o user2. Para cada jogo concluído, a lista mostra o resultado do jogo.

No código, a função `index` (em `application.py`) faz as seguintes três chamadas para recuperar informações de status do jogo:

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Cada uma dessas chamadas retorna uma lista de itens do DynamoDB que são encapsulados por objetos `Game`. É fácil extrair dados desses objetos na exibição. A função de índice passa essas listas de objetos para a exibição para renderizar o HTML.

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
```

```
                        inprogress=inProgressGames,
                        finished=finishedGames)
```

A aplicação Jogo da velha define a classe `Game` principalmente para armazenar os dados do jogo recuperados do DynamoDB. Essas funções retornam listas de objetos `Game` que permitem que você isole o resto da aplicação do código relacionado a itens do Amazon DynamoDB. Portanto, essas funções ajudam a separar o código do seu aplicativo dos detalhes da camada de armazenamento de dados.

O padrão de arquitetura descrito aqui também é chamado de padrão de interface do usuário MVC (controlador de visualização de modelo). Neste caso, as instâncias do objeto `Game` (representando os dados) são o modelo, e a página HTML é a exibição. O controlador é dividido em dois arquivos. O arquivo `application.py` tem o controlador para o framework Flask, e a lógica de negócios é isolada no arquivo `gameController.py`. Ou seja, a aplicação armazena tudo o que tem a ver com o DynamoDB SDK em seu próprio arquivo separado na pasta `dynamodb`.

Vamos analisar as três funções e como elas consultam a tabela Games usando índices secundários globais para recuperar dados relevantes.

Usar getGameInvites para obter a lista de convites de jogo pendentes

A função `getGameInvites` recupera a lista dos 10 convites pendentes mais recentes. Esses jogos foram criados pelos usuários, mas os oponentes não aceitaram os convites de jogo. Para esses jogos, o status permanece `PENDING` até que o oponente aceite o convite. Se o oponente recusar o convite, o aplicativo removerá o item correspondente da tabela.

A função especifica a consulta da seguinte forma:

- Ela especifica o índice `OpponentId-StatusDate-index` para ser usado com os seguintes valores de chave de índice e operadores de comparação:
  - A chave de partição é `OpponentId` e usa a chave de índice *`user ID`*.
  - A chave de classificação é `StatusDate` e usa o operador de comparação e o valor de chave de índice `beginswith="PENDING_"`.

  Você pode usar o índice `OpponentId-StatusDate-index` para recuperar jogos para os quais o usuário conectado é convidado – ou seja, nos quais o usuário conectado é o oponente.
- A consulta limita o resultado a 10 itens.

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                                Opponent__eq=user,
                                                StatusDate__beginswith="PENDING_",
                                                index="OpponentId-StatusDate-index",
                                                limit=10)
```

No índice, para cada `OpponentId` (a chave de partição), o DynamoDB mantém itens classificados por `StatusDate` (a chave de classificação). Portanto, os jogos que a consulta retorna serão os 10 jogos mais recentes.

Usar getGamesWithStatus para obter a lista de jogos com um status específico

Depois que um oponente aceita um convite de jogo, o status do jogo muda para `IN_PROGRESS`. Depois que o jogo for concluído, o status mudará para `FINISHED`.

As consultas para encontrar jogos que estão em andamento ou concluídos são as mesmas, exceto para o valor de status diferente. Portanto, o aplicativo define a função `getGamesWithStatus`, que usa o valor de status como um parâmetro.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

A seção a seguir aborda os jogos em andamento, mas a mesma descrição também se aplica a jogos concluídos.

Uma lista de jogos em andamento para um determinado usuário inclui o seguinte:

- Jogos em andamento hospedados pelo usuário
- Jogos em andamento nos quais o usuário é o oponente

A função `getGamesWithStatus` executa as duas consultas seguintes, cada vez usando o índice secundário apropriado.

- A função consulta a tabela `Games` usando o índice `HostId-StatusDate-index`. Para o índice, a consulta especifica valores de chave primária – tanto os valores de chave de partição (`HostId`) quanto os de chave de classificação (`StatusDate`), juntamente com operadores de comparação.

  ```
  hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
  ```

```
                                                StatusDate__beginswith=status,
                                                index="HostId-StatusDate-index",
                                                limit=10)
```

Observe a sintaxe do Python para operadores de comparação:

- `HostId__eq=user` especifica o operador de comparação de igualdade.
- `StatusDate__beginswith=status` especifica o operador de comparação `BEGINS_WITH`.

- A função consulta a tabela `Games` usando o índice `OpponentId-StatusDate-index`.

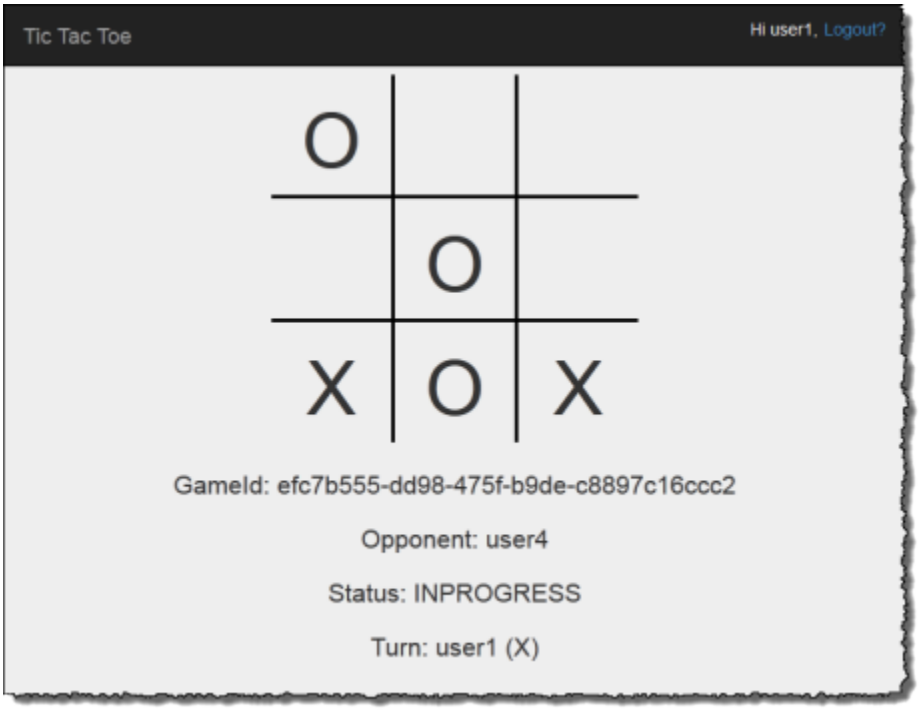
```
oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                   StatusDate__beginswith=status,
                                                   index="OpponentId-StatusDate-index",
                                                   limit=10)
```

- Em seguida, a função combina as duas listas, classifica, e para os primeiros itens de 0 a 10, cria uma lista dos objetos `Game` e retorna a lista para a função de chamada (ou seja, o índice).

```
games = self.mergeQueries(hostGamesInProgress,
                          oppGamesInProgress)
return games
```

Página de jogos

A página de jogos é onde o usuário joga os jogos da velha. Ela mostra a grade do jogo junto com as informações relevantes do jogo. A captura de tela a seguir mostra um jogo de exemplo em andamento:



O aplicativo exibe a página de jogos nas seguintes situações:

- O usuário cria um jogo convidando outro usuário para jogar.

  Neste caso, a página mostra o usuário como host e o status do jogo como `PENDING`, aguardando o oponente aceitar.
- O usuário aceita um dos convites pendentes na página inicial.

  Neste caso, a página mostra o usuário como o oponente e o status do jogo como `IN_PROGRESS`.

Uma seleção do usuário no quadro gera uma solicitação `POST` de formato para o aplicativo. Ou seja, o Flask chama a função `selectSquare` (em `application.py`) com os dados no formato HTML. Essa função, por sua vez, chama a função `updateBoardAndTurn` (em `gameController.py`) para atualizar o item de jogo da seguinte forma:

- Ela adiciona um novo atributo específico ao movimento.
- Ela atualiza o valor do atributo `Turn` para o usuário cuja vez é a próxima.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

A função retorna verdadeiro se a atualização do item foi bem-sucedida; caso contrário, retorna falso. Observe o seguinte sobre a função `updateBoardAndTurn`:

- A função chama a função `update_item` do SDK for Python para fazer um conjunto finito de atualizações em um item existente. A função é mapeada na operação `UpdateItem` no DynamoDB. Para obter mais informações, consulte UpdateItem.

> **Note**
>
> A diferença entre as operações `UpdateItem` e `PutItem` é que `PutItem` substitui o item inteiro. Para obter mais informações, consulte PutItem.

Para a chamada `update_item`, o código identifica o seguinte:

- A chave primária da tabela `Games` (ou seja, `ItemId`).

```
key = { "GameId" : { "S" : gameId } }
```

- O novo atributo a ser adicionado, específico para o movimento do usuário atual, e seu valor (por exemplo, `TopLeft="X"`).

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- Condições que devem ser verdadeiras para a atualização acontecer:
  - O jogo deve estar em andamento. Ou seja, o valor do atributo `StatusDate` deve começar com `IN_PROGRESS`.
  - A vez atual deve ser de um usuário válido, conforme especificado pelo atributo `Turn`.
  - O quadrado que o usuário escolheu deve estar disponível. Ou seja, o atributo correspondente ao quadrado não deve existir.

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
    "Turn" : {"Value" : {"S" : current_player}},
```

```
    position : {"Exists" : False}}
```

Agora, a função chama `update_item` para atualizar o item.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

Depois que a função retorna, as chamadas da função `selectSquare` são redirecionadas conforme mostrado no exemplo a seguir.

```
redirect("/game="+gameId)
```

Essa chamada faz com que o navegador seja atualizado. Como parte dessa atualização, o aplicativo verifica se o jogo terminou em uma vitória ou empate. Em caso afirmativo, o aplicativo atualizará o item de jogo adequadamente.

Etapa 3: implantar em produção usando o serviço DynamoDB

Tópicos

- 3.1: criar um perfil do IAM para oAmazon EC2
- 3.2: criar a tabela de jogos no Amazon DynamoDB
- 3.3: empacotar e implantar o código da aplicação Jogo da velha
- 3.4: configurar o ambiente do AWS Elastic Beanstalk

Em seções anteriores, você implantou e testou a aplicação Jogo da velha localmente no seu computador usando o DynamoDB local. Agora, você implanta o aplicativo em produção da seguinte forma:

- Implante o aplicativo usando o AWS Elastic Beanstalk, um serviço fácil de usar para implantação e escalabilidade de aplicativos web e web services. Para obter mais informações, consulte Implantar uma aplicação Flask no AWS Elastic Beanstalk.

  O Elastic Beanstalk inicia uma ou mais instâncias do Amazon Elastic Compute Cloud (Amazon EC2), as quais você configura por meio do Elastic Beanstalk, nas quais sua aplicação Jogo da velha será executada.

- Usando o serviço Amazon DynamoDB, crie uma tabela `Games` que resida na AWS em vez de localmente em seu computador.

Além disso, você também tem que configurar as permissões. Todos os recursos da AWS que você cria, como a tabela `Games` no DynamoDB, são privados por padrão. Somente o proprietário do recurso, que é a conta da AWS que criou a tabela `Games`, pode acessar essa tabela. Assim, por padrão, seu aplicativo Jogo da velha não pode atualizar a tabela `Games`.

Para conceder permissões necessárias, crie uma função do AWS Identity and Access Management (IAM) e conceda a essa função permissões para acessar a tabela `Games`. Sua instância do Amazon EC2; primeiro assume essa função. Em resposta, a AWS retorna credenciais de segurança temporárias que a instância do Amazon EC2 pode usar para atualizar a tabela `Games` em nome da aplicação Jogo da velha. Ao configurar sua aplicação Elastic Beanstalk, você especifica a função do IAM que a instância ou as instâncias do Amazon EC2 podem assumir. Para obter mais informações sobre os perfis do IAM, consulte Funções do IAM para Amazon EC2 no Guia do usuário do Amazon EC2.

> **Note**
>
> Antes de criar instâncias do Amazon EC2 para a aplicação Jogo da velha, você deve primeiro decidir a região da AWS onde deseja que o Elastic Beanstalk crie as instâncias. Depois de criar a aplicação Elastic Beanstalk, você deve fornecer os mesmos nomes de região e endpoint em um arquivo de configuração. A aplicação Jogo da velha usa informações desse arquivo para criar a tabela `Games` e enviar as solicitações subsequentes em uma região da AWS específica. Tanto a tabela `Games` do DynamoDB quanto as instâncias do Amazon EC2 iniciadas pelo Elastic Beanstalk devem estar na mesma região. Para obter uma lista das regiões disponíveis, acesse Amazon DynamoDB no Referência geral da Amazon Web Services.

Resumindo, você faz o seguinte para implantar o aplicativo Jogo da velha em produção:

1. Crie uma função do IAM usando o serviço IAM. Você poderá anexar uma política a essa função, concedendo permissões para as ações do DynamoDB acessarem a tabela `Games`.
2. Empacote o código do aplicativo Jogo da velha e um arquivo de configuração, e crie um arquivo `.zip`. Use o arquivo `.zip` para fornecer o código da aplicação Jogo da velha para o Elastic Beanstalk inserir em seus servidores. Para obter mais informações sobre a criação de um pacote,

acesse Criar um pacote de origem da aplicação no Guia do desenvolvedor do AWS Elastic Beanstalk.

No arquivo de configuração (`beanstalk.config`), você fornece informações sobre a região e o endpoint da AWS. A aplicação Jogo da velha usa essas informações para determinar com qual região do DynamoDB ela se comunicará.

3. Configure o ambiente do Elastic Beanstalk. O Elastic Beanstalk inicia uma ou mais instâncias do Amazon EC2 e implanta o pacote da sua aplicação nessas instâncias. Depois que o ambiente do Elastic Beanstalk estiver pronto, você fornecerá o nome do arquivo de configuração, adicionando a variável de ambiente `CONFIG_FILE`.
4. Crie uma tabela do DynamoDB. Usando o serviço Amazon DynamoDB, crie a tabela `Games` na AWS em vez de localmente em seu computador. Lembre-se: essa tabela tem uma chave primária simples que consiste na chave de partição `GameId` do tipo string.
5. Teste o jogo em produção.

3.1: criar um perfil do IAM para oAmazon EC2

Criar uma função do IAM do tipo Amazon EC2 permitirá que a instância do Amazon EC2 que está executando sua aplicação Jogo da velha assuma a função correta e faça solicitações da aplicação para acessar a tabela `Games`. Ao criar a função, selecione a opção Custom Policy (Política personalizada), copie e cole a política a seguir.

JSON

```
{
   "Version":"2012-10-17"      ,
   "Statement":[
      {
         "Action":[
            "dynamodb:ListTables"
         ],
         "Effect":"Allow",
         "Resource":"*"
      },
      {
         "Action":[
            "dynamodb:*"
         ],
```

```
            "Effect":"Allow",
            "Resource":[
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games",
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games/index/*"
            ]
         }
      ]
}
```

Para obter instruções adicionais, consulte Criação de uma função para um serviço da AWS (AWS Management Console) no Guia do usuário do IAM.

3.2: criar a tabela de jogos no Amazon DynamoDB

A tabela `Games` no DynamoDB armazena os dados do jogo. Se a tabela não existir, o aplicativo criará a tabela para você. Nesse caso, deixe o aplicativo criar a tabela `Games`.

3.3: empacotar e implantar o código da aplicação Jogo da velha

Se você seguiu as etapas deste exemplo, então, já fez download do aplicativo Jogo da velha. Caso contrário, faça download e extraia todos os arquivos para uma pasta no seu computador local. Para instruções, consulte Etapa 1: implantar e testar localmente.

Após extrair todos os arquivos, você terá uma pasta `code`. Para fazer o handoff dessa pasta para o Elastic Beanstalk, você empacotará o conteúdo dessa pasta como um arquivo `.zip`. Primeiramente, você precisa adicionar um arquivo de configuração a essa pasta. Sua aplicação usa as informações da região e do endpoint para criar uma tabela do DynamoDB na região especificada e fará solicitações de operação de tabela subsequentes usando o endpoint especificado.

1. Alterne para a pasta na qual você fez download do aplicativo Jogo da velha.
2. Na pasta raiz do aplicativo, crie um arquivo de texto chamado `beanstalk.config` com o conteúdo a seguir.

   ```
   [dynamodb]
   region=<AWS region>
   endpoint=<DynamoDB endpoint>
   ```

   Por exemplo, você pode usar o conteúdo a seguir.

   ```
   [dynamodb]
   ```

```
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
```

Para obter uma lista de regiões disponíveis, acesse Amazon DynamoDB na Referência geral da Amazon Web Services.

> **Important**
>
> A região especificada no arquivo de configuração é o local onde a aplicação Jogo da velha cria a tabela `Games` no DynamoDB. Você deve criar a aplicação Elastic Beanstalk discutida na próxima seção na mesma região.

> **Note**
>
> Ao criar sua aplicação Elastic Beanstalk, você pede para iniciar um ambiente no qual pode escolher o tipo de ambiente. Para testar o aplicativo de exemplo Jogo da velha, você pode escolher o tipo de ambiente Single Instance (Instância única), ignorar o seguinte, e ir para a próxima etapa.
> No entanto, o tipo de ambiente Load balancing, autoscaling (Balanceamento de carga, escalabilidade automática) oferece um ambiente de alta disponibilidade e escalável, algo que você deve considerar ao criar e implantar outros aplicativos. Se você escolher esse tipo de ambiente, também será necessário gerar um UUID e adicioná-lo ao arquivo de configuração, como mostrado a seguir.
>
> ```
> [dynamodb]
> region=us-west-2
> endpoint=dynamodb.us-west-2.amazonaws.com
> [flask]
> secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
> ```
>
> No comunicação entre cliente e servidor, quando o servidor envia a resposta, por segurança, o servidor envia um cookie assinado que o cliente envia de volta para o servidor na próxima solicitação. Quando há apenas um servidor, o servidor pode gerar localmente uma chave de criptografia quando ele for iniciado. Quando há vários servidores, todos eles precisam saber a mesma chave de criptografia; caso contrário, eles não poderão ler os cookies definidos pelos servidores do mesmo nível. Ao adicionar

`secret_key` ao arquivo de configuração, todos os servidores são informados para usar essa chave de criptografia.

3. Compacte o conteúdo da pasta raiz do aplicativo (que inclui o arquivo `beanstalk.config`), por exemplo, `TicTacToe.zip`.
4. Faça upload do arquivo `.zip` para um bucket do Amazon Simple Storage Service (Amazon S3) Na próxima seção, você fornece este arquivo `.zip` para o Elastic Beanstalk para fazer upload no servidor ou servidores.

   Para obter instruções sobre como fazer upload para um bucket do Amazon S3, consulte Criar um bucket e Adicionar um objeto a um bucket no Guia do usuário do Amazon Simple Storage Service.

3.4: configurar o ambiente do AWS Elastic Beanstalk

Nesta etapa, você cria uma aplicação Elastic Beanstalk, que é um conjunto de componentes, incluindo ambientes. Para este exemplo, você inicia uma instância do Amazon EC2 para implantar e executar sua aplicação Jogo da velha.

1. Insira o seguinte URL personalizado para configurar um console do Elastic Beanstalk para configurar o ambiente.

   ```
   https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
   newApplication
   ?applicationName=TicTacToeyour-name
   &solutionStackName=Python
   &sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
   &environmentType=SingleInstance
   &instanceType=t1.micro
   ```

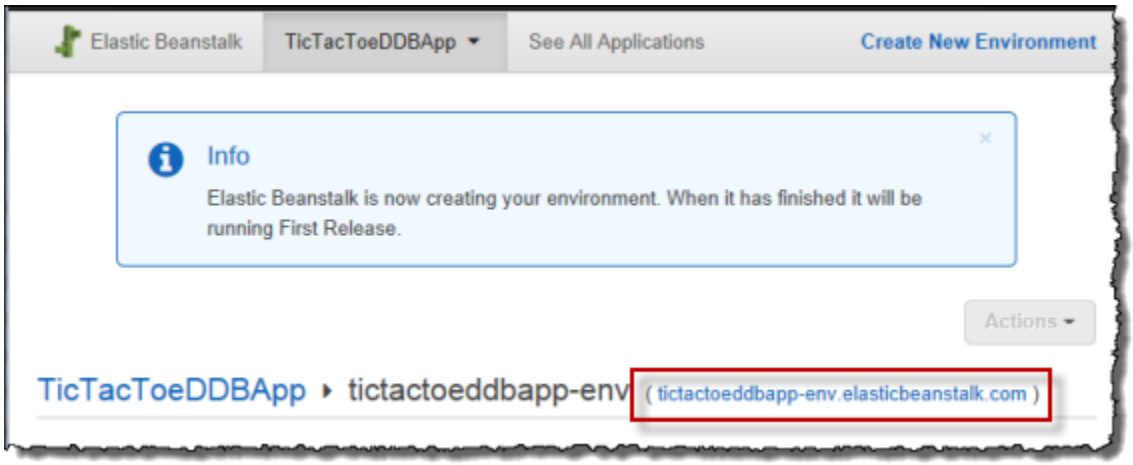
   Para obter mais informações sobre URLs personalizados, acesse Construção de um URL de início imediato no Guia do desenvolvedor do AWS Elastic Beanstalk. Para obter o URL, observe o seguinte:

   - Você precisará fornecer um nome de região da AWS (o mesmo que o fornecido no arquivo de configuração), um nome de bucket do Amazon S3 e o nome do objeto.
   - Para testes, o URL solicita o tipo de ambiente SingleInstance e `t1.micro` como o tipo de instância.

- O nome do aplicativo precisa ser exclusivo. Assim, no URL anterior, sugerimos que você insira seu nome ao `applicationName`.

Essa ação abre o console do Elastic Beanstalk. Em alguns casos, talvez seja necessário fazer login.

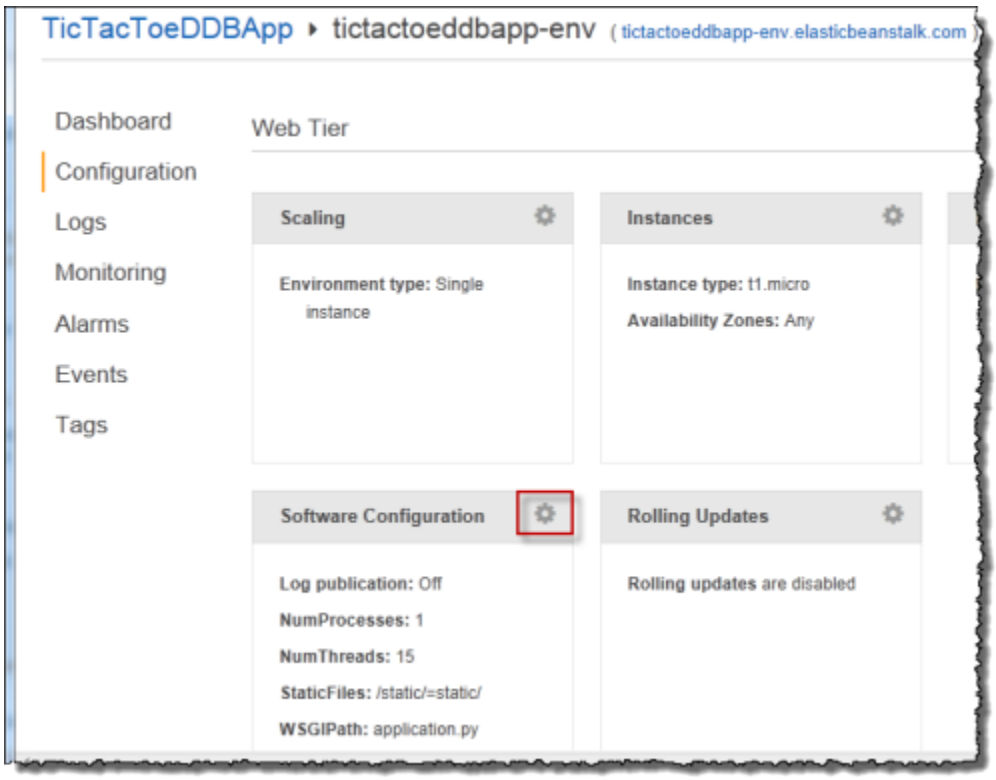
2. No console do Elastic Beanstalk, escolha Revisar e iniciar e, em seguida, selecione Iniciar.
3. Anote o URL para referência futura. Este URL abre a página inicial do seu aplicativo Jogo da velha.



4. Configure o aplicativo Jogo da velha para que ele saiba a localização do arquivo de configuração.

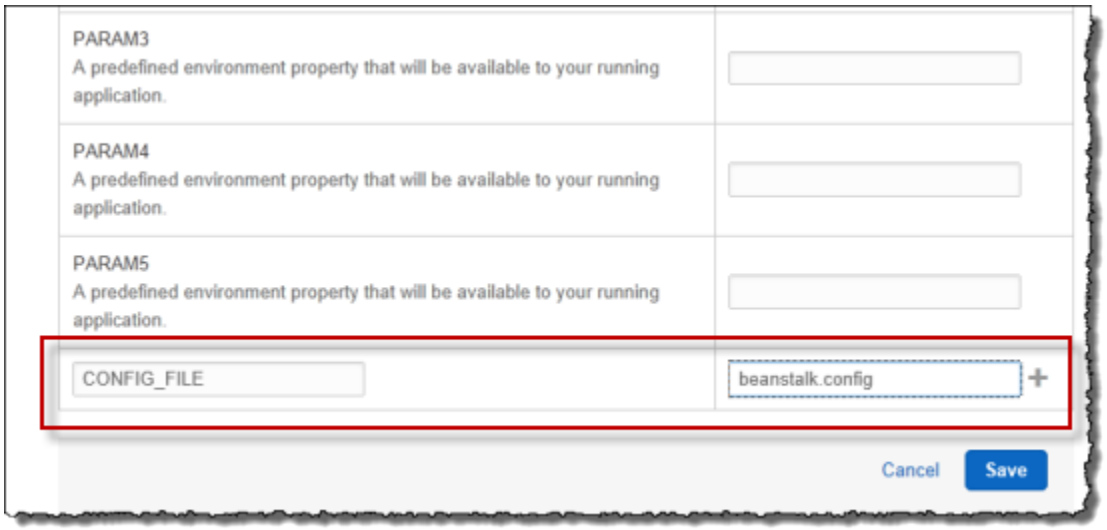
   Após o Elastic Beanstalk criar o aplicativo, escolha Configuração.

   a. Selecione o ícone de engrenagem ao lado de Software Configuration (Configuração de software), conforme mostrado na captura de tela a seguir.



b. No final da seção Environment Properties (Propriedades do ambiente), digite **`CONFIG_FILE`** e seu valor **`beanstalk.config`** e depois selecione Save (Salvar).

Pode levar alguns minutos para a atualização deste ambiente ser concluída.

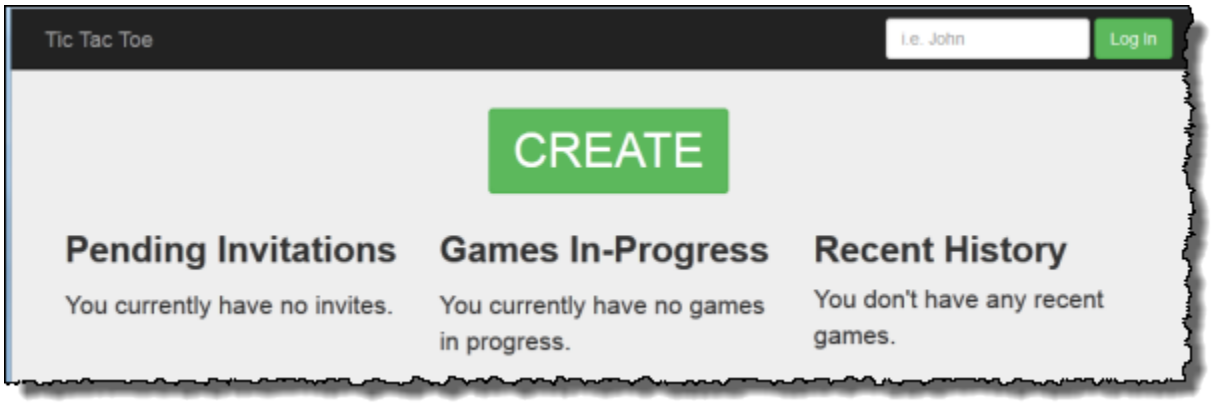


Depois que a atualização for concluída, você poderá jogar o jogo.

5. No navegador, digite o URL que você copiou na etapa anterior, como mostrado no exemplo a seguir.

```
http://<pen-name>.elasticbeanstalk.com
```

Essa ação abre a página inicial do aplicativo.



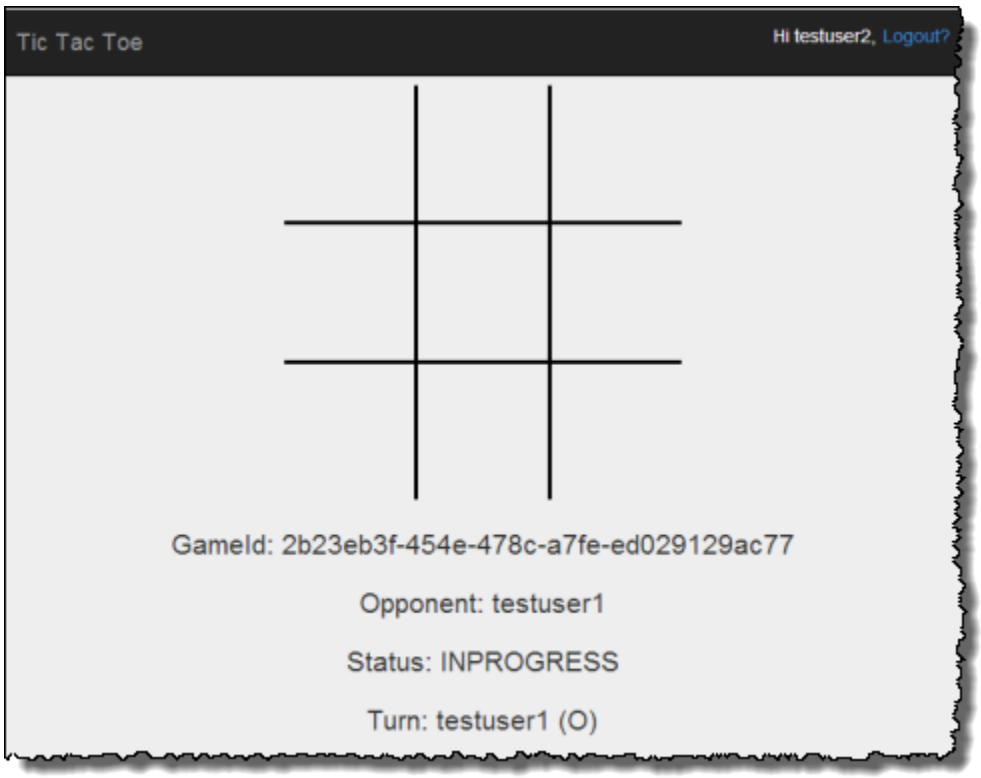
6. Faça login como testuser1 e selecione CREATE (Criar) para iniciar um novo jogo da velha.
7. Digite **`testuser2`** na caixa Choose an Opponent (Escolher um oponente).

8. Abra outra janela do navegador.

   Certifique-se de limpar todos os cookies em sua janela do navegador, de modo que você não se conecte como o mesmo usuário.
9. Digite o mesmo URL para abrir a página inicial do aplicativo, como mostrado no exemplo a seguir:

   ```
   http://<env-name>.elasticbeanstalk.com
   ```
10. Faça login como testuser2.
11. Para o convite de testuser1 na lista de convites pendentes, selecione accept (aceitar).

12. Agora a página do jogo aparece.



testuser1 e testuser2 pode jogar o jogo. Para cada movimento, o aplicativo salva a movimentação no item correspondente na tabela `Games`.

Etapa 4: Limpar os recursos

Agora você concluiu a implantação e o teste do aplicativo Jogo da velha. A aplicação aborda o desenvolvimento de aplicações web de ponta a ponta no Amazon DynamoDB, com exceção da autenticação de usuários. O aplicativo usa as informações de login na página inicial apenas para adicionar o nome do jogador ao criar um jogo. Em um aplicativo de produção, você adicionaria o código necessário para realizar o login e a autenticação do usuário.

Se você concluiu o teste, poderá remover os recursos que criou para testar o aplicativo Jogo da velha para evitar ser cobrado.

Como remover recursos criados

1. Remova a tabela `Games` criada no DynamoDB.
2. Encerre o ambiente do Elastic Beanstalk para liberar as instâncias do Amazon EC2.
3. Exclua a função do IAM que você criou.
4. Remova o objeto que você criou no Amazon S3.

Palavras reservadas no DynamoDB

As seguintes palavras-chave são reservadas para uso pelo DynamoDB. Não use nenhuma delas como nomes de atributo em expressões. A lista a seguir não diferencia maiúsculas de minúsculas.

Se precisar escrever uma expressão que contenha um nome de atributo em conflito com uma palavra reservada do DynamoDB, você poderá definir um nome de atributo de expressão para usar no lugar dessa palavra reservada. Para ter mais informações, consulte Nomes (aliases) de atributo de expressão no DynamoDB.

```
ABORT
ABSOLUTE
ACTION
ADD
AFTER
AGENT
AGGREGATE
ALL
ALLOCATE
ALTER
```

```
ANALYZE
AND
ANY
ARCHIVE
ARE
ARRAY
AS
ASC
ASCII
ASENSITIVE
ASSERTION
ASYMMETRIC
AT
ATOMIC
ATTACH
ATTRIBUTE
AUTH
AUTHORIZATION
AUTHORIZE
AUTO
AVG
BACK
BACKUP
BASE
BATCH
BEFORE
BEGIN
BETWEEN
BIGINT
BINARY
BIT
BLOB
BLOCK
BOOLEAN
BOTH
BREADTH
BUCKET
BULK
BY
BYTE
CALL
CALLED
CALLING
CAPACITY
```

```
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHECK
CLASS
CLOB
CLOSE
CLUSTER
CLUSTERED
CLUSTERING
CLUSTERS
COALESCE
COLLATE
COLLATION
COLLECTION
COLUMN
COLUMNS
COMBINE
COMMENT
COMMIT
COMPACT
COMPILE
COMPRESS
CONDITION
CONFLICT
CONNECT
CONNECTION
CONSISTENCY
CONSISTENT
CONSTRAINT
CONSTRAINTS
CONSTRUCTOR
CONSUMED
CONTINUE
CONVERT
COPY
CORRESPONDING
COUNT
COUNTER
CREATE
```

```
CROSS
CUBE
CURRENT
CURSOR
CYCLE
DATA
DATABASE
DATE
DATETIME
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DEFINE
DEFINED
DEFINITION
DELETE
DELIMITED
DEPTH
DEREF
DESC
DESCRIBE
DESCRIPTOR
DETACH
DETERMINISTIC
DIAGNOSTICS
DIRECTORIES
DISABLE
DISCONNECT
DISTINCT
DISTRIBUTE
DO
DOMAIN
DOUBLE
DROP
DUMP
DURATION
DYNAMIC
EACH
ELEMENT
```

```
ELSE
ELSEIF
EMPTY
ENABLE
END
EQUAL
EQUALS
ERROR
ESCAPE
ESCAPED
EVAL
EVALUATE
EXCEEDED
EXCEPT
EXCEPTION
EXCEPTIONS
EXCLUSIVE
EXEC
EXECUTE
EXISTS
EXIT
EXPLAIN
EXPLODE
EXPORT
EXPRESSION
EXTENDED
EXTERNAL
EXTRACT
FAIL
FALSE
FAMILY
FETCH
FIELDS
FILE
FILTER
FILTERING
FINAL
FINISH
FIRST
FIXED
FLATTERN
FLOAT
FOR
FORCE
```

```
FOREIGN
FORMAT
FORWARD
FOUND
FREE
FROM
FULL
FUNCTION
FUNCTIONS
GENERAL
GENERATE
GET
GLOB
GLOBAL
GO
GOTO
GRANT
GREATER
GROUP
GROUPING
HANDLER
HASH
HAVE
HAVING
HEAP
HIDDEN
HOLD
HOUR
IDENTIFIED
IDENTITY
IF
IGNORE
IMMEDIATE
IMPORT
IN
INCLUDING
INCLUSIVE
INCREMENT
INCREMENTAL
INDEX
INDEXED
INDEXES
INDICATOR
INFINITE
```

```
INITIALLY
INLINE
INNER
INNTER
INOUT
INPUT
INSENSITIVE
INSERT
INSTEAD
INT
INTEGER
INTERSECT
INTERVAL
INTO
INVALIDATE
IS
ISOLATION
ITEM
ITEMS
ITERATE
JOIN
KEY
KEYS
LAG
LANGUAGE
LARGE
LAST
LATERAL
LEAD
LEADING
LEAVE
LEFT
LENGTH
LESS
LEVEL
LIKE
LIMIT
LIMITED
LINES
LIST
LOAD
LOCAL
LOCALTIME
LOCALTIMESTAMP
```

```
LOCATION
LOCATOR
LOCK
LOCKS
LOG
LOGED
LONG
LOOP
LOWER
MAP
MATCH
MATERIALIZED
MAX
MAXLEN
MEMBER
MERGE
METHOD
METRICS
MIN
MINUS
MINUTE
MISSING
MOD
MODE
MODIFIES
MODIFY
MODULE
MONTH
MULTI
MULTISET
NAME
NAMES
NATIONAL
NATURAL
NCHAR
NCLOB
NEW
NEXT
NO
NONE
NOT
NULL
NULLIF
NUMBER
```

```
NUMERIC
OBJECT
OF
OFFLINE
OFFSET
OLD
ON
ONLINE
ONLY
OPAQUE
OPEN
OPERATOR
OPTION
OR
ORDER
ORDINALITY
OTHER
OTHERS
OUT
OUTER
OUTPUT
OVER
OVERLAPS
OVERRIDE
OWNER
PAD
PARALLEL
PARAMETER
PARAMETERS
PARTIAL
PARTITION
PARTITIONED
PARTITIONS
PATH
PERCENT
PERCENTILE
PERMISSION
PERMISSIONS
PIPE
PIPELINED
PLAN
POOL
POSITION
PRECISION
```

```
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVATE
PRIVILEGES
PROCEDURE
PROCESSED
PROJECT
PROJECTION
PROPERTY
PROVISIONING
PUBLIC
PUT
QUERY
QUIT
QUORUM
RAISE
RANDOM
RANGE
RANK
RAW
READ
READS
REAL
REBUILD
RECORD
RECURSIVE
REDUCE
REF
REFERENCE
REFERENCES
REFERENCING
REGEXP
REGION
REINDEX
RELATIVE
RELEASE
REMAINDER
RENAME
REPEAT
REPLACE
REQUEST
RESET
```

```
RESIGNAL
RESOURCE
RESPONSE
RESTORE
RESTRICT
RESULT
RETURN
RETURNING
RETURNS
REVERSE
REVOKE
RIGHT
ROLE
ROLES
ROLLBACK
ROLLUP
ROUTINE
ROW
ROWS
RULE
RULES
SAMPLE
SATISFIES
SAVE
SAVEPOINT
SCAN
SCHEMA
SCOPE
SCROLL
SEARCH
SECOND
SECTION
SEGMENT
SEGMENTS
SELECT
SELF
SEMI
SENSITIVE
SEPARATE
SEQUENCE
SERIALIZABLE
SESSION
SET
SETS
```

```
SHARD
SHARE
SHARED
SHORT
SHOW
SIGNAL
SIMILAR
SIZE
SKEWED
SMALLINT
SNAPSHOT
SOME
SOURCE
SPACE
SPACES
SPARSE
SPECIFIC
SPECIFICTYPE
SPLIT
SQL
SQLCODE
SQLERROR
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATIC
STATUS
STORAGE
STORE
STORED
STREAM
STRING
STRUCT
STYLE
SUB
SUBMULTISET
SUBPARTITION
SUBSTRING
SUBTYPE
SUM
SUPER
SYMMETRIC
```

```
SYNONYM
SYSTEM
TABLE
TABLESAMPLE
TEMP
TEMPORARY
TERMINATED
TEXT
THAN
THEN
THROUGHPUT
TIME
TIMESTAMP
TIMEZONE
TINYINT
TO
TOKEN
TOTAL
TOUCH
TRAILING
TRANSACTION
TRANSFORM
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
TRUNCATE
TTL
TUPLE
TYPE
UNDER
UNDO
UNION
UNIQUE
UNIT
UNKNOWN
UNLOGGED
UNNEST
UNPROCESSED
UNSIGNED
UNTIL
UPDATE
```

```
UPPER
URL
USAGE
USE
USER
USERS
USING
UUID
VACUUM
VALUE
VALUED
VALUES
VARCHAR
VARIABLE
VARIANCE
VARINT
VARYING
VIEW
VIEWS
VIRTUAL
VOID
WAIT
WHEN
WHENEVER
WHERE
WHILE
WINDOW
WITH
WITHIN
WITHOUT
WORK
WRAPPED
WRITE
YEAR
ZONE
```

Exemplos do AWS SDK for Java 1.x

Esta seção contém código de exemplo para aplicações do DAX usando o SDK for Java 1.x.

Tópicos

- Uso do DAX com o AWS SDK for Java 1.x

- Modificar uma aplicação do SDK for Java 1.x existente para usar o DAX
- Consultar índices secundários globais com o SDK for Java 1.x

Uso do DAX com o AWS SDK for Java 1.x

Siga este procedimento para executar o exemplo de Java para o Amazon DynamoDB Accelerator (DAX) na instância do Amazon EC2.

> **Note**
>
> Estas instruções destinam-se a aplicações que usam o AWS SDK for Java 1.x. Para aplicações que usam o AWS SDK for Java 2.x, consulte Java e DAX.

Como executar o exemplo de Java para DAX

1. Instale o Java Development Kit (JDK).

   ```
   sudo yum install -y java-devel
   ```

2. Faça download do AWS SDK para Java (arquivo `.zip`) e extraia-o.

   ```
   wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

   unzip aws-java-sdk.zip
   ```

3. Baixe a versão mais recente do cliente Java do DAX (arquivo `.jar`).

   ```
   wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
   latest.jar
   ```

   > **Note**
   >
   > O cliente do SDK for Java do DAX está disponível no Apache Maven. Para ter mais informações, consulte Usar o cliente como dependência do Apache Maven.

4. Defina a variável `CLASSPATH`. Neste exemplo, substitua *`sdkVersion`* pelo número da versão atual do AWS SDK para Java (por exemplo, `1.11.112`).

```
export SDKVERSION=sdkVersion

export CLASSPATH=$(pwd)/TryDax/java:$(pwd)/DaxJavaClient-latest.jar:$(pwd)/
aws-java-sdk-$SDKVERSION/lib/aws-java-sdk-$SDKVERSION.jar:$(pwd)/aws-java-sdk-
$SDKVERSION/third-party/lib/*
```

5. Baixe o código-fonte do programa de exemplo (arquivo `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Quando o download for concluído, extraia os arquivos de origem.

```
unzip TryDax.zip
```

6. Navegue até o diretório de código Java e compile o código conforme descrito a seguir.

```
cd TryDax/java/
javac TryDax*.java
```

7. Execute o programa.

```
java TryDax
```

Você deve ver saída semelhante ao seguinte:

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
Writing 10 items for partition key: 4
Writing 10 items for partition key: 5
Writing 10 items for partition key: 6
Writing 10 items for partition key: 7
Writing 10 items for partition key: 8
Writing 10 items for partition key: 9
```

```
Writing 10 items for partition key: 10

Running GetItem, Scan, and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1 and sort keys 1-10
 Total time: 136.681 ms - Avg time: 13.668 ms
 Total time: 122.632 ms - Avg time: 12.263 ms
 Total time: 167.762 ms - Avg time: 16.776 ms
 Total time: 108.130 ms - Avg time: 10.813 ms
 Total time: 137.890 ms - Avg time: 13.789 ms
Query test - partition key 5 and sort keys between 2 and 9
 Total time: 13.560 ms - Avg time: 2.712 ms
 Total time: 11.339 ms - Avg time: 2.268 ms
 Total time: 7.809 ms - Avg time: 1.562 ms
 Total time: 10.736 ms - Avg time: 2.147 ms
 Total time: 12.122 ms - Avg time: 2.424 ms
Scan test - all items in the table
 Total time: 58.952 ms - Avg time: 11.790 ms
 Total time: 25.507 ms - Avg time: 5.101 ms
 Total time: 37.660 ms - Avg time: 7.532 ms
 Total time: 26.781 ms - Avg time: 5.356 ms
 Total time: 46.076 ms - Avg time: 9.215 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Anote as informações de tempo: o número de milissegundos necessários para os testes `GetItem`, `Query` e `Scan`.

8. Na etapa anterior, você executou o programa no endpoint do DynamoDB. Agora, execute o programa novamente, mas, desta vez, as operações `GetItem`, `Query` e `Scan` são processadas pelo cluster do DAX.

   Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

   - Usando o console do DynamoDB: escolha seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Usando a AWS CLI: insira o comando a seguir.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

O endpoint do cluster é mostrado na saída, como no exemplo a seguir.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Agora execute o programa novamente, mas, desta vez, especifique o endpoint do cluster como um parâmetro de linha de comando.

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Observe o restante da saída e anote as informações de tempo. Os tempos decorridos para `GetItem`, `Query` e `Scan` devem ser significativamente mais baixos com o DAX do que com o DynamoDB.

Para obter mais informações sobre esse programa, consulte as seguintes seções:

- TryDax.java
- TryDaxHelper.java
- TryDaxTests.java

Usar o cliente como dependência do Apache Maven

Siga estas etapas para usar o cliente do SDK do DAX para Java em seu aplicativo como uma dependência.

Como usar o cliente como uma dependência do Maven

1. Faça download do Apache Maven e instale-o. Para obter mais informações, consulte Download do Apache Maven e Instalação do Apache Maven.

2. Adicione a dependência do cliente Maven ao arquivo Project Object Model (POM) da aplicação. Neste exemplo, substitua `x.x.x.x` pelo número da versão real do cliente (por exemplo, `1.0.200704.0`).

```
<!--Dependency:-->
<dependencies>
    <dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>amazon-dax-client</artifactId>
     <version>x.x.x.x</version>
    </dependency>
</dependencies>
```

TryDax.java

O arquivo `TryDax.java` contém o método `main`. Se você executar o programa sem parâmetros de linha de comando, ele criará um cliente do Amazon DynamoDB e usará esse cliente para todas as operações de API. Se você especificar um endpoint de cluster do DynamoDB Accelerator (DAX) na linha de comando, o programa também criará um cliente do DAX e o usará para operações `GetItem`, `Query` e `Scan`.

É possível modificar o programa de várias maneiras:

- Use o cliente do DAX em vez do cliente do DynamoDB. Para ter mais informações, consulte Java e DAX.
- Escolha um nome diferente para a tabela de teste.
- Modifique o número de itens gravados, alterando os parâmetros `helper.writeData`. O segundo parâmetro é o número de chaves de partição, e o terceiro parâmetro é o número de chaves de classificação. Por padrão, o programa usa 1–10 para valores de chave de partição e 1–10 para valores de chave de classificação, totalizando 100 itens gravados na tabela. Para ter mais informações, consulte TryDaxHelper.java.
- Modifique o número de testes de `GetItem`, `Query` e `Scan` e modifique seus parâmetros.
- Assinale como comentários as linhas contendo `helper.createTable` e `helper.deleteTable` (se não quiser criar e excluir a tabela de cada vez que executar o programa).

> **Note**
>
> Para executar o programa, é possível configurar o Maven para usar o cliente do SDK for Java do DAX e o AWS SDK para Java como dependências. Para ter mais informações, consulte Usar o cliente como dependência do Apache Maven.
> Opcionalmente, você pode fazer download e incluir o cliente Java do DAX e o AWS SDK para Java em seu classpath. Consulte Java e DAX para obter um exemplo de configuração da variável `CLASSPATH`.

```
public class TryDax {

    public static void main(String[] args) throws Exception {

        TryDaxHelper helper = new TryDaxHelper();
        TryDaxTests tests = new TryDaxTests();

        DynamoDB ddbClient = helper.getDynamoDBClient();
        DynamoDB daxClient = null;
        if (args.length >= 1) {
            daxClient = helper.getDaxClient(args[0]);
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        helper.createTable(tableName, ddbClient);
        System.out.println("Populating table...");
        helper.writeData(tableName, ddbClient, 10, 10);

        DynamoDB testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem, Scan, and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");
```

```
        // GetItem
        tests.getItemTest(tableName, testClient, 1, 10, 5);

        // Query
        tests.queryTest(tableName, testClient, 5, 2, 9, 5);

        // Scan
        tests.scanTest(tableName, testClient, 5);

        helper.deleteTable(tableName, ddbClient);
    }

}
```

TryDaxHelper.java

O arquivo `TryDaxHelper.java` contém métodos utilitários.

Os métodos `getDynamoDBClient` e `getDaxClient` fornecem clientes do Amazon DynamoDB e DynamoDB Accelerator (DAX). Para operações do plano de controle (`CreateTable`, `DeleteTable`) e operações de gravação, o programa usa o cliente do DynamoDB. Se você especificar um endpoint de cluster do DAX, o programa principal criará um cliente do DAX para realizar operações de leitura (`GetItem`, `Query`, `Scan`).

Os outros métodos de `TryDaxHelper` (`createTable`, `writeData`, `deleteTable`) são para configurar e destruir a tabela do DynamoDB e seus dados.

É possível modificar o programa de várias maneiras:

- Use configurações de throughput provisionado diferentes para a tabela.
- Modifique o tamanho de cada item gravado (consulte a variável `stringSize` no método `writeData`).
- Modifique o número de testes de `GetItem`, `Query` e `Scan` e seus parâmetros.
- Assinale como comentários as linhas contendo `helper.CreateTable` e `helper.DeleteTable` (se não quiser criar e excluir a tabela de cada vez que executar o programa).

> **Note**
>
> Para executar o programa, é possível configurar o Maven para usar o cliente do SDK for Java do DAX e o AWS SDK para Java como dependências. Para ter mais informações, consulte Usar o cliente como dependência do Apache Maven.
> Ou você pode fazer download e incluir o cliente Java do DAX e o AWS SDK para Java em seu classpath. Consulte Java e DAX para obter um exemplo de configuração da variável `CLASSPATH`.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.util.EC2MetadataUtils;

public class TryDaxHelper {

    private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

    DynamoDB getDynamoDBClient() {
        System.out.println("Creating a DynamoDB client");
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
                .withRegion(region)
                .build();
        return new DynamoDB(client);
    }

    DynamoDB getDaxClient(String daxEndpoint) {
        System.out.println("Creating a DAX client with cluster endpoint " +
 daxEndpoint);
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
```

```
        AmazonDynamoDB client = daxClientBuilder.build();
        return new DynamoDB(client);
    }

    void createTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("Attempting to create table; please wait...");

            table = client.createTable(tableName,
                    Arrays.asList(
                            new KeySchemaElement("pk", KeyType.HASH), // Partition key
                            new KeySchemaElement("sk", KeyType.RANGE)), // Sort key
                    Arrays.asList(
                            new AttributeDefinition("pk", ScalarAttributeType.N),
                            new AttributeDefinition("sk", ScalarAttributeType.N)),
                    new ProvisionedThroughput(10L, 10L));
            table.waitForActive();
            System.out.println("Successfully created table.  Table status: " +
                    table.getDescription().getTableStatus());

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    void writeData(String tableName, DynamoDB client, int pkmax, int skmax) {
        Table table = client.getTable(tableName);
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (Integer ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (Integer isk = 1; isk <= skmax; isk++) {
                    table.putItem(new Item()
```

```
                            .withPrimaryKey("pk", ipk, "sk", isk)
                            .withString("someData", someData));
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    void deleteTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            table.delete();
            table.waitForDelete();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

}
```

TryDaxTests.java

O arquivo `TryDaxTests.java` contém métodos que executam operações de leitura em uma tabela de teste no Amazon DynamoDB. Esses métodos não consideram como os dados serão acessados (usando o cliente do DynamoDB ou o cliente do DAX) e, portanto, não é necessário modificar a lógica da aplicação.

É possível modificar o programa de várias maneiras:

- Modifique o método `queryTest` para que ele use um `KeyConditionExpression` diferente.
- Adicione um `ScanFilter` ao método `scanTest`, para que apenas alguns dos itens sejam retornados para você.

> **Note**
>
> Para executar o programa, é possível configurar o Maven para usar o cliente do SDK for Java do DAX e o AWS SDK para Java como dependências. Para ter mais informações, consulte Usar o cliente como dependência do Apache Maven.
> Ou você pode fazer download e incluir o cliente Java do DAX e o AWS SDK para Java em seu classpath. Consulte Java e DAX para obter um exemplo de configuração da variável `CLASSPATH`.

```
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;

public class TryDaxTests {

    void getItemTest(String tableName, DynamoDB client, int pk, int sk, int iterations)
 {
        long startTime, endTime;
        System.out.println("GetItem test - partition key " + pk + " and sort keys 1-" +
 sk);
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (Integer ipk = 1; ipk <= pk; ipk++) {
                    for (Integer isk = 1; isk <= sk; isk++) {
                        table.getItem("pk", ipk, "sk", isk);
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
```

```
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    void queryTest(String tableName, DynamoDB client, int pk, int sk1, int sk2, int
iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key " + pk + " and sort keys between
" + sk1 + " and " + sk2);
        Table table = client.getTable(tableName);

        HashMap<String, Object> valueMap = new HashMap<String, Object>();
        valueMap.put(":pkval", pk);
        valueMap.put(":skval1", sk1);
        valueMap.put(":skval2", sk2);

        QuerySpec spec = new QuerySpec()
                .withKeyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                .withValueMap(valueMap);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<QueryOutcome> items = table.query(spec);

            try {
                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to query table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    void scanTest(String tableName, DynamoDB client, int iterations) {
        long startTime, endTime;
        System.out.println("Scan test - all items in the table");
        Table table = client.getTable(tableName);
```

```
        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<ScanOutcome> items = table.scan();
            try {

                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to scan table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    public void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

Modificar uma aplicação do SDK for Java 1.x existente para usar o DAX

Se você já tem uma aplicação Java que usa o Amazon DynamoDB, será necessário modificá-la para que ela possa acessar seu cluster do DynamoDB Accelerator (DAX). Não é necessário reescrever toda a aplicação porque o cliente Java do DAX é muito semelhante ao cliente de nível inferior do DynamoDB incluído no AWS SDK para Java.

> **Note**
>
> Estas instruções destinam-se a aplicações que usam o AWS SDK for Java 1.x. Para aplicações que usam o AWS SDK for Java 2.x, consulte Como modificar uma aplicação existente para usar o DAX.

Suponha que você tenha uma tabela do DynamoDB chamada `Music`. A chave de partição dessa tabela é `Artist`, e sua chave de classificação é `SongTitle`. O seguinte programa lê um item diretamente da tabela `Music`.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

        // Create a DynamoDB client
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

        HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
        key.put("Artist", new AttributeValue().withS("No One You Know"));
        key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

        GetItemRequest request = new GetItemRequest()
            .withTableName("Music").withKey(key);

        try {
            System.out.println("Attempting to read the item...");
            GetItemResult result = client.getItem(request);
            System.out.println("GetItem succeeded: " + result);

        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }
    }
}
```

Para modificar o programa, substitua o cliente do DynamoDB por um cliente do DAX.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
```

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

    //Create a DAX client

    AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
    daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
    AmazonDynamoDB client = daxClientBuilder.build();


       /*
       ** ...
       ** Remaining code omitted (it is identical)
       ** ...
       */

    }
}
```

Usar a API de documentos do DynamoDB

O AWS SDK para Java fornece uma interface de documentos para o DynamoDB. Essa API de documentos atua como um wrapper em torno do cliente de baixo nível do DynamoDB. Para obter mais informações, consulte Interfaces de documentos.

A interface de documentos também pode ser usada com o cliente de baixo nível do DAX, conforme mostrado no exemplo a seguir.

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class GetMusicItemWithDocumentApi {
```

```
    public static void main(String[] args) throws Exception {

        //Create a DAX client

        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
        AmazonDynamoDB client = daxClientBuilder.build();

        // Document client wrapper
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");

        try {
            System.out.println("Attempting to read the item...");
            GetItemOutcome outcome = table.tgetItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Scared of My Shadow");
            System.out.println(outcome.getItem());
            System.out.println("GetItem succeeded: " + outcome);
        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }

    }
}
```

Cliente assíncrono do DAX

O `AmazonDaxClient` é síncrono. Para uma operação da API do DAX de longa execução, como uma operação `Scan` de uma tabela grande, isso poderá bloquear a execução do programa até que a operação seja concluída. Se o seu programa precisa realizar outros trabalhos enquanto uma operação de API do DAX está em andamento, use `ClusterDaxAsyncClient` em vez disso.

O programa a seguir mostra como usar `ClusterDaxAsyncClient`, junto com Java `Future`, para implementar uma solução sem bloqueio.

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

import com.amazon.dax.client.dynamodbv2.ClientConfig;
```

```
import com.amazon.dax.client.dynamodbv2.ClusterDaxAsyncClient;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.handlers.AsyncHandler;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBAsync;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class DaxAsyncClientDemo {
 public static void main(String[] args) throws Exception {

  ClientConfig daxConfig = new ClientConfig().withCredentialsProvider(new
 ProfileCredentialsProvider())
    .withEndpoints("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111");

  AmazonDynamoDBAsync client = new ClusterDaxAsyncClient(daxConfig);

  HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
  key.put("Artist", new AttributeValue().withS("No One You Know"));
  key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

  GetItemRequest request = new GetItemRequest()
    .withTableName("Music").withKey(key);

  // Java Futures
  Future<GetItemResult> call = client.getItemAsync(request);
  while (!call.isDone()) {
   // Do other processing while you're waiting for the response
   System.out.println("Doing something else for a few seconds...");
   Thread.sleep(3000);
  }
  // The results should be ready by now

  try {
   call.get();

  } catch (ExecutionException ee) {
   // Futures always wrap errors as an ExecutionException.
   // The *real* exception is stored as the cause of the
   // ExecutionException
   Throwable exception = ee.getCause();
   System.out.println("Error getting item: " + exception.getMessage());
  }
```

```
  // Async callbacks
  call = client.getItemAsync(request, new AsyncHandler<GetItemRequest, GetItemResult>()
 {

   @Override
   public void onSuccess(GetItemRequest request, GetItemResult getItemResult) {
    System.out.println("Result: " + getItemResult);
   }

   @Override
   public void onError(Exception e) {
    System.out.println("Unable to read item");
    System.err.println(e.getMessage());
    // Callers can also test if exception is an instance of
    // AmazonServiceException or AmazonClientException and cast
    // it to get additional information
   }

  });
  call.get();

 }
}
```

Consultar índices secundários globais com o SDK for Java 1.x

Você pode usar o Amazon DynamoDB Accelerator (DAX) para consultar índices secundários globais usando interfaces programáticas do DynamoDB.

O exemplo a seguir demonstra como usar o DAX para consultar o índice secundário global `CreateDateIndex` que é criado em Exemplo: índices secundários globais usando a API de documento do AWS SDK para Java.

A classe `DAXClient` instancia os objetos de cliente que são necessários para interagir com as interfaces de programação do DynamoDB.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import software.amazon.dynamodb.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
import software.amazon.dynamodb.AmazonDynamoDB;
```

```
public class DaxClient {

 private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

 DynamoDB getDaxDocClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDB(client);
 }

 DynamoDBMapper getDaxMapperClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDBMapper(client);
 }
}
```

É possível consultar um índice secundário global das seguintes formas:

- Use o método `queryIndex` na classe `QueryIndexDax` definida no seguinte exemplo. O `QueryIndexDax` considera como um parâmetro o objeto de cliente que é retornado pelo método `getDaxDocClient` na classe `DaxClient`.
- Se você estiver usando a interface de persistência de objetos, use o método `queryIndexMapper` na classe `QueryIndexDax` definida no seguinte exemplo. O `queryIndexMapper` considera como um parâmetro o objeto de cliente que é retornado pelo método `getDaxMapperClient` definido na classe `DaxClient`.

```
import java.util.Iterator;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import java.util.List;
import software.amazon.dynamodb.datamodeling.DynamoDBQueryExpression;
import software.amazon.dynamodb.model.AttributeValue;
import java.util.HashMap;
```

```
import software.amazon.dynamodb.document.Item;
import software.amazon.dynamodb.document.utils.ValueMap;
import software.amazon.dynamodb.document.spec.QuerySpec;
import software.amazon.dynamodb.document.QueryOutcome;
import software.amazon.dynamodb.document.ItemCollection;
import software.amazon.dynamodb.document.Index;
import software.amazon.dynamodb.document.Table;
import software.amazon.dynamodb.document.DynamoDB;

public class QueryIndexDax {

 //This is used to query Index using the low-level interface.
 public static void queryIndex(DynamoDB client, String tableName, String indexName) {
  Table table = client.getTable(tableName);

  System.out.println("\n*******************************************************
\n");
  System.out.print("Querying index " + indexName + "...");

  Index index = table.getIndex(indexName);

  ItemCollection<QueryOutcome> items = null;

  QuerySpec querySpec = new QuerySpec();

  if (indexName == "CreateDateIndex") {
   System.out.println("Issues filed on 2013-11-01");
   querySpec.withKeyConditionExpression("CreateDate = :v_date and
 begins_with(IssueId, :v_issue)")
     .withValueMap(new ValueMap().withString(":v_date",
 "2013-11-01").withString(":v_issue", "A-"));
   items = index.query(querySpec);
  } else {
   System.out.println("\nNo valid index name provided");
   return;
  }

  Iterator<Item> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
   System.out.println(iterator.next().toJSONPretty());
  }
```

```
 }

 //This is used to query Index using the high-level mapper interface.
 public static void queryIndexMapper(DynamoDBMapper mapper, String tableName, String
 indexName) {
  HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
  eav.put(":v_date", new AttributeValue().withS("2013-11-01"));
  eav.put(":v_issue", new AttributeValue().withS("A-"));
  DynamoDBQueryExpression<CreateDate> queryExpression = new
 DynamoDBQueryExpression<CreateDate>()
     .withIndexName("CreateDateIndex").withConsistentRead(false)
     .withKeyConditionExpression("CreateDate = :v_date and
 begins_with(IssueId, :v_issue)")
     .withExpressionAttributeValues(eav);

  List<CreateDate> items = mapper.query(CreateDate.class, queryExpression);
  Iterator<CreateDate> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
   CreateDate iterObj = iterator.next();
   System.out.println(iterObj.getCreateDate());
   System.out.println(iterObj.getIssueId());
  }
 }
}
```

A definição de classe abaixo representa a tabela de problemas e é usada no método `queryIndexMapper`.

```
import software.amazon.dynamodb.datamodeling.DynamoDBTable;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexRangeKey;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;

@DynamoDBTable(tableName = "Issues")
public class CreateDate {
 private String createDate;
 @DynamoDBHashKey(attributeName = "IssueId")
 private String issueId;
```

```
 @DynamoDBIndexHashKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "CreateDate")
 public String getCreateDate() {
  return createDate;
 }

 public void setCreateDate(String createDate) {
  this.createDate = createDate;
 }

 @DynamoDBIndexRangeKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "IssueId")
 public String getIssueId() {
  return issueId;
 }

 public void setIssueId(String issueId) {
  this.issueId = issueId;
 }
}
```

Exemplos do SDK da AWS para Go 1.x

Esta seção contém exemplo de código para aplicações do DAX usando o Go 1.x.

Tópicos

- DAX SDK for Go

DAX SDK for Go

Siga este procedimento para executar o exemplo do SDK for Go para Amazon DynamoDB Accelerator (DAX) em sua instância do Amazon EC2.

Para executar o exemplo do SDK for Go para DAX

1. Configure o SDK for Go em sua instância do Amazon EC2:

   a. Instale a linguagem de programação Go (`Golang`).

   ```
   sudo yum install -y golang
   ```

b. Verifique se o Golang está instalado e funcionando corretamente.

```
go version
```

Uma mensagem semelhante a esta deve ser mostrada.

```
go version go1.15.5 linux/amd64
```

As instruções restantes dependem do suporte do módulo, o qual se tornou o padrão com o Go versão 1.13.

2. Instale o aplicativo de exemplo do Golang.

```
go get github.com/aws-samples/aws-dax-go-sample
```

3. Execute os seguintes programas do Golang. O primeiro programa cria uma tabela do DynamoDB chamada `TryDaxGoTable`. O segundo programa grava dados na tabela.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command put-item
```

4. Execute os seguintes programas do Golang.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command scan
```

Anote as informações de tempo: o número de milissegundos necessários para os testes `GetItem`, `Query` e `Scan`.

5. Na etapa anterior, você executou os programas no endpoint do DynamoDB. Agora, execute os programas novamente, mas, desta vez, as operações `GetItem`, `Query` e `Scan` são processadas pelo cluster do DAX.

   Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

   - Usando o console do DynamoDB: escolha seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Usando a AWS CLI: insira o comando a seguir.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     O endpoint do cluster é mostrado na saída, como no exemplo a seguir.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Agora, execute os programas novamente, mas, desta vez, especifique o endpoint do cluster como um parâmetro de linha de comando.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

Observe o restante da saída e anote as informações de tempo. Os tempos decorridos para `GetItem`, `Query` e `Scan` devem ser significativamente mais baixos com o DAX do que com o DynamoDB.

6. Execute o seguinte programa Golang para excluir a `TryDaxGoTable`.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command delete-table
```

Exemplos do SDK da AWS para Node.js 2.x

Esta seção contém código de exemplo para aplicações do DAX usando o SDK da AWS para Node.js 2.x.

Tópicos

- Node.js e DAX

Node.js e DAX

Siga estas etapas para executar a aplicação de exemplo em Node.js na instância do Amazon EC2.

Para executar o exemplo em Node.js para DAX

1. Configure o Node.js na instância do Amazon EC2 da seguinte forma:

   a. Instale o gerenciador de versão de nó (`nvm`).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
   ```

   b. Use o nvm para instalar o Node.js.

   ```
   nvm install 12.16.3
   ```

   c. Verifique se o Node.js está instalado e funcionando corretamente.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Isso deve exibir a seguinte mensagem.

```
Running Node.js v12.16.3
```

2. Instale o cliente Node.js do DAX usando o gerenciador de pacotes de nó (`npm`).

```
npm install amazon-dax-client
```

3. Baixe o código-fonte do programa de exemplo (arquivo `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Quando o download for concluído, extraia os arquivos de origem.

```
unzip TryDax.zip
```

4. Execute os seguintes programas Node.js. O primeiro programa cria uma tabela do Amazon DynamoDB chamada `TryDaxTable`. O segundo programa grava dados na tabela.

```
node 01-create-table.js
node 02-write-data.js
```

5. Execute os seguintes programas Node.js.

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

Anote as informações de tempo: o número de milissegundos necessários para os testes `GetItem`, `Query` e `Scan`.

6. Na etapa anterior, você executou os programas no endpoint do DynamoDB. Execute os programas novamente, mas, desta vez, as operações, `GetItem`, `Query` e `Scan` são processadas pelo cluster do DAX.

Para determinar o endpoint do cluster do DAX, escolha uma das seguintes opções:

- Usando o console do DynamoDB: escolha seu cluster do DAX. O endpoint do cluster é mostrado no console, como no exemplo a seguir.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Usando a AWS CLI: insira o comando a seguir.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

O endpoint do cluster é mostrado na saída, como no exemplo a seguir.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Agora, execute os programas novamente, mas, desta vez, especifique o endpoint do cluster como um parâmetro de linha de comando.

```
node 03-getitem-test.js dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
node 04-query-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
node 05-scan-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Observe o restante da saída e anote as informações de tempo. Os tempos decorridos para `GetItem`, `Query` e `Scan` devem ser significativamente mais baixos com o DAX do que com o DynamoDB.

7. Execute o seguinte programa Node.js para excluir a `TryDaxTable`.

```
node 06-delete-table
```

Para obter mais informações sobre esses programas, consulte as seguintes seções:

- 01-create-table.js
- 02-write-data.js
- 03-getitem-test.js
- 04-query-test.js
- 05-scan-test.js
- 06-delete-table.js

01-create-table.js

O programa `01-create-table.js` cria uma tabela (`TryDaxTable`). Os programas Node.js restantes nesta seção dependem desta tabela.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
  KeySchema: [
    { AttributeName: "pk", KeyType: "HASH" }, //Partition key
    { AttributeName: "sk", KeyType: "RANGE" }, //Sort key
  ],
  AttributeDefinitions: [
    { AttributeName: "pk", AttributeType: "N" },
    { AttributeName: "sk", AttributeType: "N" },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 10,
    WriteCapacityUnits: 10,
  },
};

dynamodb.createTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to create table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Created table. Table description JSON:",
```

```
      JSON.stringify(data, null, 2)
    );
  }
});
```

02-write-data.js

O programa `02-write-data.js` grava dados de teste em `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();

var tableName = "TryDaxTable";

var someData = "X".repeat(1000);
var pkmax = 10;
var skmax = 10;

for (var ipk = 1; ipk <= pkmax; ipk++) {
  for (var isk = 1; isk <= skmax; isk++) {
    var params = {
      TableName: tableName,
      Item: {
        pk: ipk,
        sk: isk,
        someData: someData,
      },
    };

    //
    //put item

    ddbClient.put(params, function (err, data) {
      if (err) {
```

```
        console.error("Unable to write data: ", JSON.stringify(err, null, 2));
      } else {
        console.log("PutItem succeeded");
      }
    });
  }
}
```

03-getitem-test.js

O programa `03-getitem-test.js` executa operações `GetItem` em `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 1;
var sk = 10;
var iterations = 5;

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();
```

```
  for (var ipk = 1; ipk <= pk; ipk++) {
    for (var isk = 1; isk <= sk; isk++) {
      var params = {
        TableName: tableName,
        Key: {
          pk: ipk,
          sk: isk,
        },
      };

      client.get(params, function (err, data) {
        if (err) {
          console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
        } else {
          // GetItem succeeded
        }
      });
    }
  }

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

04-query-test.js

O programa `04-query-test.js` executa operações `Query` em `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";
```

```
AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;

var params = {
  TableName: tableName,
  KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
  ExpressionAttributeValues: {
    ":pkval": pk,
    ":skval1": sk1,
    ":skval2": sk2,
  },
};

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  client.query(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Query succeeded
```

```
    }
  });

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

05-scan-test.js

O programa `05-scan-test.js` executa operações `Scan` em `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var iterations = 5;
```

```
var params = {
  TableName: tableName,
};
var startTime = new Date().getTime();
for (var i = 0; i < iterations; i++) {
  client.scan(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Scan succeeded
    }
  });
}

var endTime = new Date().getTime();
console.log(
  "\tTotal time: ",
  endTime - startTime,
  "ms - Avg time: ",
  (endTime - startTime) / iterations,
  "ms"
);
```

06-delete-table.js

O programa `06-delete-table.js` exclui `TryDaxTable`. Execute esse programa depois de concluir o teste.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client
```

```
var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
};

dynamodb.deleteTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to delete table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Deleted table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

Histórico de documentos do DynamoDB

A tabela a seguir descreve as alterações importantes em cada versão do Guia do desenvolvedor do DynamoDB de 3 de julho de 2018 em diante. Para receber notificações sobre atualizações nessa documentação, assine o RSS feed (no canto superior esquerdo da página).

Alteração	Descrição	Data
O DynamoDB aprimorou seus recursos de diagnóstico de controle de utilização com um novo modo do CloudWatch Contributor Insights e adicionou detalhes de exceções de controle de utilização e métricas direcionadas do CloudWatch	O DynamoDB oferece diagnósticos aprimorados de controle de utilização por meio de um novo modo apenas de `Throttled keys` no CloudWatch Contributor Insights que rastreia especificamente chaves de partição com utilização mais limitada e de exceções de controle de utilização mais detalhadas que incluem campos `ThrottlingReason` estruturados, fornecendo códigos de motivo e ARNs de recursos específicos para ajudar a identificar exatamente qual recurso é limitado e por quê, bem como 16 novas métricas direcionadas do CloudWatch que correspondem a diferentes cenários de controle de utilização, para oferecer um monitoramento preciso de eventos específicos de controle de utilização. Para ter mais informações, consulte	14 de agosto de 2025

	Problemas de controle de utilização no DynamoDB.	
Maior limite para alternar entre os modos de capacidade	Aumento do limite para alternar uma tabela do modo de capacidade provisionada para o modo de capacidad e sob demanda de uma vez a cada 24 horas para quatro vezes a cada 24 horas. Essa mudança oferece maior flexibilidade para workloads que exigem o carregame nto de grandes volumes de dados várias vezes por dia ou precisam atender a aumentos inesperados de tráfego. Para ter mais informações, consulte Considerações ao alternar os modos de capacidade no DynamoDB.	12 de agosto de 2025
Lançado o DynamoDB Local versão 3.0, que migra do SDK da AWS para Java V1 para V2	Essa atualização remove a dependência da V1, melhora o desempenho e se alinha à estrutura do pacote SDK da AWS V2.	17 de julho de 2025

Adicionado suporte para descoberta otimizada de fragmentos no DynamoDB Streams	O novo mecanismo de descoberta de fragmento s reduz o tempo inicial de descoberta, melhora o desempenho do processam ento em estado estável, fornece visibilidade do status do processamento e mantém a latência p100 consistente em operações de grande escala. Para ter mais informações, consulte AWS PrivateLink for Amazon DynamoDB Streams.	17 de julho de 2025
Adicionado o suporte para consistência forte multirreg ional (MRSC) para tabelas globais do DynamoDB	O DynamoDB agora oferece suporte a consistência forte multirregional (MRSC) para tabelas globais com região testemunha opcional. O MRSC permite leituras altamente consistentes em todas as regiões com um objetivo de ponto de recuperaç ão (RPO) de zero durante falhas regionais. As tabelas globais do MRSC devem ser implantadas em exatament e três regiões, com três réplicas ou duas réplicas e uma testemunha. Para obter mais informações e regiões disponíveis, consulte Consistência forte multirreg ional (MRSC).	30 de junho de 2025

Adaptador da Kinesis Client Library (KCL) 2.x do Amazon DynamoDB Streams	O DynamoDB agora oferece suporte a um adaptador atualizado do DynamoDB Streams compatível com a KCL 2.x e o AWS SDK v2. Para ter mais informaçõ es, consulte Migrating from KCL1.x to KCL 3.x.	11 de junho de 2025
Suporte adicionado para executar o DynamoDB localmente no AWS CloudShel l	Agora você pode executar o DynamoDB local e diretamen te no seu navegador por meio do AWS Management Console sem precisar baixar ou instalar software. Para ter mais informações, consulte Implantar o DynamoDB localmente no computador.	19 de maio de 2025
Suporte adicionado para tipos de instância r7i no DAX	Opções de tipo de instância expandida para o DAX para oferecer maior flexibilidade em desempenho e capacidad e. Para obter mais detalhes sobre os novos tipos de instância r7i e os respectivos recursos, consulte Criptografia em repouso do DAX.	29 de abril de 2025

Nova política de acesso total `AmazonDynamoDBFullAccess_v2` com escopo reduzido substitui `AmazonDynamoDBFullAccess` .	Em 25 de abril de 2025, o DynamoDB descontinuou a política `AmazonDynamoDBFullAccess` e a substituiu por uma política de escopo reduzido chamada `AmazonDynamoDBFullAccess_v2` . Para ter mais informações, consulte AWS managed policy: AmazonDynamoDBFullAccess_v2.	28 de abril de 2025
O DynamoDB adiciona suporte do AWS PrivateLink para o DynamoDB Streams	A AWS adicionou compatibilidade com o DynamoDB Streams, permitindo que os clientes invoquem as APIs do DynamoDB Streams diretamente na Amazon Virtual Private Cloud (VPC) usando conectividade de rede privada. Esse recurso permite acesso simplificado à rede privada sem atravessar a internet pública, atendendo aos requisitos de conformidade e segurança das workloads do DynamoDB. Para ter mais informações, consulte AWS PrivateLink for Amazon DynamoDB Streams.	26 de março de 2025
O DAX SDK para JS 3.x e Go 2.x já estão disponíveis	O DynamoDB Accelerator (DAX) SDK para JS 3.x agora está disponível e é compatível com o AWS SDK para Java 3.x.	17 de março de 2025

Lançamento do modo de capacidade sob demanda do NoSQL Workbench 3.13.5 para configurações padrão de tabela	Quando você cria uma tabela com configurações padrão, o DynamoDB cria uma tabela que usa o modo de capacidade sob demanda em vez do modo de capacidade provisionada.	24 de fevereiro de 2025
Nova prática recomendada para configurar clientes do DAX	Publicação de uma nova orientação prescritiva de práticas recomendadas do DAX para configurar clientes do DAX. Para ter mais informações, consulte Configurar clientes do DAX.	17 de fevereiro de 2025
Novas práticas recomendadas para operações de dados em massa	Publicação de novos tópicos de práticas recomendadas para trabalhar com modelos de dados complexos no DynamoDB. Consulte mais informações em Best practices for using bulk data operations in DynamoDB.	9 de janeiro de 2025
Amazon DynamoDB agora é compatível com a recuperação para um ponto no tempo (PITR) configurável	O DynamoDB agora é compatível com períodos de recuperação configuráveis para PITR. Agora é possível definir o período de PITR para cada tabela entre 1 e 35 dias. Consulte mais informações em Backups para um ponto no tempo do DynamoDB.	7 de janeiro de 2025

Novo tópico publicado sobre a integração do Amazon Managed Streaming for Apache Kafka com o Amazon DynamoDB	Saiba como o Amazon Managed Streaming for Apache Kafka se integra ao Amazon DynamoDB lendo dados de tópicos do Apache Kafka e armazenando-os no DynamoDB. Para ter mais informações, consulte Integrar o DynamoDB com o Amazon Managed Streaming for Apache Kafka.	26 de dezembro de 2024
DynamoDB apresenta suporte à integração ETL zero com o SageMaker AI Lakehouse	O DynamoDB apresenta uma integração ETL zero que automatiza a extração e o carregamento de dados do Amazon DynamoDB no data lake do cliente. Consulte mais informações em Integração ETL zero do DynamoDB com o Amazon SageMaker Lakehouse.	3 de dezembro de 2024
As tabelas globais do DynamoDB agora aceitam consistência forte multirregional	Com a consistência forte multirregional, é possível criar aplicações multirregionais altamente disponíveis com um objetivo de ponto de recuperação (RPO) de zero, alcançando o mais alto nível de resiliência. Para ter mais informações, consulte Global tables with multi-Region strong consistency.	3 de dezembro de 2024

Atualização da política gerenciada do DynamoDB	Adição de duas novas permissões à política gerenciada `AmazonDynamoDBReadOnlyAccess` : `dynamodb:GetAbacStatus` e `dynamodb:UpdateAbacStatus` . Essas permissões autorizam que você visualize o status do ABAC e habilite o ABAC para a Conta da AWS na região atual. Para ter mais informações, consulte Política gerenciada da AWS: AmazonDynamoDBReadOnlyAccess.	18 de novembro de 2024
O DynamoDB apresenta compatibilidade com o controle de acesso por atributo (ABAC)	O ABAC é uma estratégia de autorização que define permissões de acesso com base nas tags anexadas a usuários, perfis e recursos da AWS. O ABAC usa condições baseadas em tags em suas políticas do AWS Identity and Access Management (IAM) ou em outras políticas para permitir ou negar ações específicas em suas tabelas ou índices quando as tags das entidades principais do IAM correspondem às tags das tabelas. Para ter mais informações, consulte Usar o controle de acesso por atributo com o DynamoDB.	18 de novembro de 2024

O DynamoDB apresenta throughput a quente para tabelas sob demanda e provisionadas	O DynamoDB agora comporta throughput a quente. O throughput a quente oferece visibilidade do número de operações de leitura e de gravação que sua tabela do DynamoDB pode comportar instantaneamente, bem como a capacidade de preaquece r as tabelas do DynamoDB. Para ter mais informações, consulte Throughput a quente do DynamoDB.	13 de novembro de 2024
Capacidade de consumir registros do Amazon DynamoDB Streams com o Apache Flink	Agora você pode consumir registros do Amazon DynamoDB Streams com o Apache Flink e utilizar o Amazon Managed Service for Apache Flink para criar e gerenciar rapidamente aplicações de processamento de fluxos de ponta a ponta. Para ter mais informações, consulte DynamoDB Streams e Apache Flink.	12 de novembro de 2024

Novos tópicos sobre faturamento publicados para tabelas e backups globais	Dois novos tópicos publicados sobre faturamento para tabelas globais e para backups. Para obter mais informações, consulte Conceitos básicos sobre faturamento do Amazon DynamoDB para tabelas globais e Conceitos básicos sobre faturamento do Amazon DynamoDB para backups.	16 de outubro de 2024
Integração ETL zero do Amazon DynamoDB com o Amazon Redshift	A integração ETL zero do Amazon DynamoDB com o Amazon Redshift fornece um pipeline de ETL sem código e totalmente gerenciado com replicação do DynamoDB para o Amazon Redshift. Para ter mais informações, consulte Integração ETL zero do Amazon DynamoDB com o Amazon Redshift.	15 de outubro de 2024
Atualização somente da documentação para adicionar um tópico sobre o uso de IA generativa com o DynamoDB	Um novo tópico publicado com informações sobre o uso de IA generativa com o DynamoDB, incluindo exemplos de casos de uso de IA generativa para o DynamoDB. Para obter mais informações, consulte Usar IA generativa com o DynamoDB.	11 de outubro de 2024

O SDK agora oferece suporte a endpoints baseados em contas da AWS	Foi adicionada a documentação para endpoints baseados em contas e a configuração do `ACCOUNT_ID_ENDPOINT_MODE` para clientes do SDK. Para obter mais informações, consulte SDK support of AWS account-based endpoints.	3 de setembro de 2024
Experiência inicial redefinida	A experiência inicial foi redefinida para consolidar as informações e ajudar você a começar com uma integração mais rápida. Para obter mais informações, consulte Conceitos básicos do DynamoDB.	1.º de agosto de 2024
Expansão do DAX para novas regiões: Espanha e Suécia	O DAX agora está disponível nas regiões Espanha e Suécia. Para obter mais informações, consulte Componentes de cluster do DAX.	30 de julho de 2024
A documentação de backup e de restauração do DynamoDB foi reestruturada e consolidada.	O Guia do desenvolvedor do DynamoDB tem uma nova estrutura para backup e restauração. Para ter mais informações, consulte Usar backup e restauração do DynamoDB.	2 de julho de 2024

Atualização do tópico “O que é o Amazon DynamoDB?”	Publicação de uma versão revisada e atualizada do tópico “O que é o Amazon DynamoDB?”. Para obter mais informações, consulte O que é o Amazon DynamoDB?	21 de junho de 2024
Integrar o DynamoDB Streams ao EventBridge	Publicação de um novo tópico sobre a integração do DynamoDB Streams com o EventBridge. Para obter mais informações, consulte Integrating with EventBridge.	21 de junho de 2024
Recomendações do DAX	Publicação de um novo tópico de práticas recomendadas que fornece informações abrangentes para usar o DynamoDB Accelerator de forma eficaz. Esse tópico aborda a otimização do desempenho, o gerenciamento dos custos e as práticas recomendadas operacionais. Para obter mais informações, consulte Recomendações do DAX.	3 de junho de 2024
Migrar uma tabela do DynamoDB de uma conta para outra	Adição de um novo tópico sobre a migração de tabelas do DynamoDB de uma conta para outra. Para obter mais informações, consulte Migrar uma tabela do DynamoDB de uma conta para outra.	29 de maio de 2024

A documentação de monitoramento e de registro em log do DynamoDB foi reestruturada e consolidada	Uma nova estrutura para monitoramento e registro em log no DynamoDB inclui três capítulos concisos sobre métricas, operações de registro em log e insights de colaboradores.	3 de maio de 2024

A documentação do modo de capacidade do DynamoDB foi reestruturada e consolidada	O guia do DynamoDB agora inclui um novo capítulo, que contém todas as informaçõ es sobre os modos de capacidade do DynamoDB: sob demanda e provision ado. Com essa atualizaç ão, o tópico Considera tions when changing read/ write Capacity Mode foi movido para o capítulo “Best practices”. Agora esse tópico é denominado Considera tions when switching capacity modes e inclui informações elaboradas sobre as práticas recomendadas ao alternar entre os modos de capacidad e. Além disso, o guia agora apresenta um novo capítulo, que inclui todas as informaçõ es sobre leituras e gravações do DynamoDB e consumos de unidades de capacidad e para operações de leitura e de gravação. Para ter mais informações, consulte DynamoDB throughput capacity, Considerations when switching capacity modes e DynamoDB reads and writes.	1º. de maio de 2024

Número máximo de solicitações sob demanda	Agora é possível especificar o número máximo de solicitações sob demanda que uma tabela individual, um índice ou ambos podem realizar. Especificar o throughput máximo sob demanda ajudará a manter o uso e os custos por tabela limitados e a impedir o aumento inadvertido nos recursos consumidos. Para ter mais informações, consulte Maximum throughput for on-demand tables.	1º. de maio de 2024
Melhorias no criador de operações do NoSQL Workbench	O NoSQL Workbench agora inclui suporte nativo para o modo escuro. Operações aprimoradas de tabelas e de itens no criador de operações . As informações de solicitações do criador de operações e de resultados de itens estão disponíveis no formato JSON. Para ter mais informações, consulte Criador de operações do NoSQL Workbench.	24 de abril de 2024

Políticas baseadas em recursos para recursos do Amazon DynamoDB	O DynamoDB agora é compatível com políticas baseadas em recursos para tabelas, índices e fluxos. Com as políticas baseadas em recursos, é possível definir as permissões de acesso especificando quem tem acesso a cada recurso e as ações que podem ser realizadas em cada recurso. Para ter mais informações, consulte Using resource-based policies for DynamoDB.	20 de março de 2024
Atualização da política gerenciada do DynamoDB	Foi adicionada uma nova permissão `dynamodb:GetResourcePolicy` à política gerenciada `AmazonDynamoDBReadOnlyAccess` . Essa permissão concede acesso para ler políticas baseadas em recursos vinculadas aos recursos do DynamoDB. Para ter mais informações, consulte Política gerenciada da AWS: AmazonDynamoDBReadOnlyAccess.	20 de março de 2024

AWS PrivateLink para Amazon DynamoDB	O Amazon DynamoDB agora comporta o AWS PrivateLink. Com o AWS PrivateLink, é possível simplificar a conectividade de rede privada entre nuvens privadas virtuais (VPCs), o DynamoDB e seus data centers on-premises usando endpoints da VPC de interface e endereços IP privados. Para ter mais informações, consulte AWS PrivateLink para DynamoDB.	19 de março de 2024
Guia de programação com JavaScript	O Amazon DynamoDB apresenta um guia de programação para o AWS SDK para JavaScript. Saiba mais sobre o AWS SDK para JavaScript, camadas de abstração, configuração de conexão, tratamento de erros, definição de políticas de repetição, gerenciamento de keep-alive e muito mais. Para ter mais informações, consulte Programar com JavaScript.	6 de março de 2024

Guia de programação com AWS SDK for Java 2.x	Foi criado um guia de programação que aborda detalhadamente interfaces de alto nível, baixo nível e de documentos, clientes HTTP e as respectivas configurações e tratamento de erros, bem como as configurações mais comuns nas quais você deve pensar ao usar o SDK para Java 2.x. Para ter mais informações, consulte Programar o Amazon DynamoDB com AWS SDK for Java 2.x.	5 de março de 2024
Clonar tabelas com NoSQL Workbench	Permitem que os desenvolvedores usem o NoSQL Workbench para copiar ou clonar tabelas entre ambientes e regiões de desenvolvimento (DynamoDB Local e DynamoDB web). Para ter mais informações, consulte Clonar tabelas com NoSQL Workbench.	26 de fevereiro de 2024
Guia de programação com Python	Foi criado um guia que aborda detalhadamente as bibliotecas de alto e baixo nível, bem como as configurações mais comuns que devem ser consideradas ao usar o SDK do Python. Para ter mais informações, consulte Programar com Python.	5 de janeiro de 2024

O tópico Vida útil (TTL) foi reformulado	A seção TTL do guia foi completamente reformulada. O novo guia ajuda você a começar a usar o TTL fornecendo trechos de código prontos para uso ao longo do caminho. Os trechos de código atuais fornecidos estão em Python e Javascript. Para ter mais informações, consulte TTL.	20 de dezembro de 2023
Práticas recomendadas para entender os relatórios de uso e faturamento da AWS	Foi adicionada uma nova seção que esclarece vários tipos de uso e as cobranças desses tipos de uso no DynamoDB. Para ter mais informações, consulte Relatórios de uso e faturamento.	15 de dezembro de 2023
Integração ETL zero do Amazon DynamoDB com o Amazon OpenSearch Service	O Amazon DynamoDB agora aceita a integração ETL zero com o Amazon OpenSearch Service, o que permite realizar uma pesquisa nos dados do DynamoDB replicando-os e transformando-os automaticamente sem código ou infraestrutura personalizados. Para ter mais informações, consulte Integração ETL zero do Amazon DynamoDB com Amazon OpenSearch Service.	28 de novembro de 2023

Migrar um banco de dados relacional para o DynamoDB	Foi criado um guia de migração para ajudar os usuários a entender como migrar de um banco de dados relacional para o DynamoDB.	27 de novembro de 2023
Gere dados de amostra com o NoSQL Workbench	O NoSQL Workbench para Amazon DynamoDB agora comporta a criação de modelos de dados diretamente de modelos de dados de amostra para ajudar você a projetar esquemas de dados para suas workloads. Você pode usar esse recurso para se familiarizar com as práticas recomendadas de modelagem de dados NoSQL ao criar aplicações no DynamoDB.	28 de setembro de 2023
Exportação incremental para o S3	Agora você pode exportar dados que foram inseridos, atualizados ou excluídos, em pequenos incrementos. Com a Exportação incremental, você pode exportar dados alterados que variam de alguns megabytes a terabytes com apenas alguns cliques no Console de Gerenciamento da AWS, em uma chamada de API ou na AWS Command Line Interface.	26 de setembro de 2023

Modelagem de dados para o DynamoDB	Agora você pode aprender mais sobre modelagem de dados com exemplos do DynamoDB que se concentram em casos de uso específicos, bem como os respectivos padrões de acesso, e obter orientação passo a passo para realizar esses padrões de acesso.	14 de julho de 2023
Seção de resolução de problemas	Agora é possível encontrar conteúdo de solução de problemas de latência e controle de utilização que podem ocorrer em suas tabelas do DynamoDB.	13 de março de 2023
Proteção contra exclusão para o Amazon DynamoDB	A proteção contra exclusão agora está disponível para tabelas do Amazon DynamoDB em todas as regiões da AWS. Agora o DynamoDB permite que você proteja suas tabelas contra exclusão acidental ao realizar operações regulares de gerenciamento de tabelas.	8 de março de 2023

Suporte do AWS CloudFormation para KDSD em tabelas globais	O Amazon Kinesis Data Streams para DynamoDB agora é compatível com o AWS CloudFormation para tabelas globais do DynamoDB, o que significa que você pode habilitar a transmissão para um Amazon Kinesis Data Streams nas tabelas do DynamoDB com modelos do CloudFormation.	15 de fevereiro de 2023
O DynamoDB local permite 100 ações por transação	Agora você pode realizar até 100 ações em uma única transação no DynamoDB local.	9 de fevereiro de 2023
Usar Lentes do Well-Architected para DynamoDB para otimizar uma workload do DynamoDB	Agora é possível usar o Lentes do Well-Architected para DynamoDB, um conjunto de princípios de design e orientações que você pode utilizar para projetar workloads bem arquitetadas do DynamoDB.	3 de fevereiro de 2023
Disponibilidade de PartiQL em GovCloud	PartiQL: uma linguagem de consultas compatível com SQL para o Amazon DynamoDB agora é compatível em AWS GovCloud (Leste dos EUA) e AWS GovCloud (Oeste dos EUA).	21 de dezembro de 2022

Conjunto de instalação único para NoSQL Workbench e DynamoDB local	O NoSQL Workbench para DynamoDB agora inclui um processo de instalação guiado do DynamoDB local para simplificar a configuração do ambiente de desenvolvimento do DynamoDB local.	6 de dezembro de 2022
Importação em massa do S3	Agora o Amazon DynamoDB comporta importações em massa do Amazon S3, facilitando a migração e o carregamento de dados em novas tabelas do DynamoDB.	18 de agosto de 2022
Integração aprimorada com o Service Quotas	Agora, o Service Quotas permite que você gerencie proativamente suas cotas de conta e tabela. Você pode visualizar os valores atuais, definir alarmes para quando a utilização de uma cota exceder um limite configurável e muito mais.	15 de junho de 2022
O NoSQL Workbench agora é compatível com tabelas e GSI	Agora, você pode usar o NoSQL Workbench para operações do ambiente de gerenciamento de tabela e índice secundário global (GSI), como CreateTable, UpdateTable e DeleteTable.	2 de junho de 2022

Classe de tabela Standard-Infrequent Acess já disponível na China	Classe de tabela Standard-Infrequent Access do Amazon DynamoDB já disponível nas regiões da China. Reduza seus custos do DynamoDB em até 60%, usando essa nova classe de tabela para tabelas que armazenam dados acessados com pouca frequência.	18 de abril de 2022
Aumento nas cotas de serviço padrão e nas operações de gerenciamento de tabelas	O DynamoDB aumentou a cota padrão para o número de tabelas por conta e região de 256 para 2,5 mil tabelas, bem como o número de operações de gerenciamento de tabelas simultâneas de 50 para 500.	9 de março de 2022
Limitação opcional de itens com o PartiQL for DynamoDB	O DynamoDB pode limitar o número de itens processados no PartiQL para operações do DynamoDB como um parâmetro opcional em cada solicitação.	8 de março de 2022
O AWS Backup agora está disponível nas regiões da China (Pequim e Ningxia).	O AWS Backup agora se integra ao DynamoDB nas regiões da China (Pequim e Ningxia). Você pode atender aos requisitos de conformidade e continuidade de negócios com maior facilidade e por meio de recursos aprimorados de backup no AWS Backup, como backups entre contas e entre regiões.	26 de janeiro de 2022

Informações de capacidade de throughput via chamadas de API PartiQL	O DynamoDB pode retornar a capacidade de throughput consumida por chamadas de API do PartiQL para ajudar você a otimizar suas consultas e custos de throughput.	18 de janeiro de 2022
Integração do AWS Backup	O DynamoDB agora ajuda você a atender aos requisitos de conformidade e continuidade de negócios com maior facilidade por meio de recursos aprimorados de backup no AWS Backup, como backups entre contas e entre regiões.	24 de novembro de 2021
Conjuntos de dados de importação/exportação do NoSQL Workbench em CSV	O NoSQL Workbench for Amazon DynamoDB agora permite importar e preencher automaticamente dados de exemplo para ajudar você a criar e visualizar seus modelos de dados.	11 de outubro de 2021
Filtrar e recuperar a atividade do plano de dados do Amazon DynamoDB Streams com o AWS CloudTrail	O Amazon DynamoDB agora oferece um controle mais granular do registro em log de auditoria ao permitir que vocêfiltre a atividade da API do plano de dados do Streams no AWS CloudTrail.	22 de setembro de 2021

Console atualizado	O console do DynamoDB agora é o console padrão para ajudar você a gerenciar dados com maior facilidade, simplificar tarefas comuns e oferecer acesso mais rápido a capacidades e recursos.	25 de agosto de 2021
O DAX SDK para Java 2.x já está disponível	O DynamoDB Accelerator (DAX) SDK para Java 2.x agora disponível e é compatível com o AWS SDK para Java 2.x. Você pode se beneficiar dos recursos mais recentes, incluindo E/S/S sem bloqueio.	29 de julho de 2021
Atualizações do recurso NoSQL Workbench, incluindo operações do ambiente de gerenciamento	O NoSQL Workbench for Amazon DynamoDB agora ajuda você a executar operações frequentes com maior facilidade para modificar e acessar dados de tabela.	28 de julho de 2021
As tabelas globais do DynamoDB agora estão disponíveis na região Ásia-Pacífico	As tabelas globais do DynamoDB agora estão disponíveis na região Ásia-Pacífico (Osaka). Replique automaticamente suas tabelas do DynamoDB nas regiões da AWS escolhidas por você.	28 de julho de 2021
O DAX agora está disponível na China	O DynamoDB Accelerator (DAX) agora está disponíve l na região China (Pequim), operada pela Sinnet.	28 de julho de 2021

Criptografia em trânsito do DAX	O DynamoDB Accelerator (DAX) agora comporta criptografia em trânsito de dados entre as aplicações e clusters do DAX e entre os nós dentro de um cluster do DAX.	24 de julho de 2021
Integração com o CloudFormation e o CloudTrail	Integração com o AWS CloudFormation e aprimoramentos de segurança com o registro em log do plano de dados do CloudFormation.	18 de junho de 2021
O CloudFormation agora é compatível com tabelas globais	As tabelas globais do Amazon DynamoDB agora são compatíveis com o AWS CloudFormation, o que significa que você pode criar tabelas globais e gerenciar suas configurações com modelos do CloudFormation.	14 de maio de 2021
Compatibilidade local do Amazon DynamoDB for Java 2.x	Agora você pode usar o AWS SDK para Java 2.x com o DynamoDB local, a versão para download do Amazon DynamoDB. Com o DynamoDB local, você pode desenvolver e testar aplicações usando uma versão do DynamoDB em execução em seu ambiente de desenvolvimento local sem incorrer em custos adicionais.	3 de maio de 2021

O NoSQL Workbench agora é compatível com o AWS CloudFormation	O NoSQL Workbench for Amazon DynamoDB agora é compatível com o AWS CloudFormation, para que você possa gerenciar e modificar modelos de dados do DynamoDB com modelos do CloudFormation. Além disso, agora você pode definir as configurações de capacidade da tabela no NoSQL Workbench.	22 de abril de 2021
O DynamoDB e o AWS Amplify agora comportam integração	O AWS Amplify agora orquestra várias atualizações de índice secundário global do DynamoDB em uma única implantação.	20 de abril de 2021
AWS CloudTrail para registrar APIs do plano de dados do Amazon DynamoDB Streams	Agora você pode usar o AWS CloudTrail para registrar a atividade da API do plano de dados do Amazon DynamoDB Streams e monitorar e investigar alterações em nível de item em suas tabelas do DynamoDB.	20 de abril de 2021

O Amazon Kinesis Data Streams for Amazon DynamoDB agora é compatível com o AWS CloudFormation	O Amazon Kinesis Data Streams for Amazon DynamoDB é compatível com o AWS CloudFormation, o que significa que você pode habilitar a transmissão para um fluxo de dados do Amazon Kinesis em suas tabelas do DynamoDB com modelos do CloudFormation. Ao transmitir suas alterações de dados do DynamoDB para um fluxo de dados do Kinesis, você pode criar aplicações avançadas de transmissão com os serviços do Amazon Kinesis.	12 de abril de 2021
O Amazon Keyspaces agora oferece endpoints compatíveis com o FIPS 140-2	oferta Amazon Keyspaces (for Apache Cassandra) agora oferece endpoints compatíveis com o Federal Information Processing Standards (FIPS) 140-2 para ajudar você a executar workloads altamente regulamentadas com maior facilidade. O FIPS 140-2 é um padrão do governo canadense que especifica os requisitos de segurança para módulos criptográficos que protegem informações sigilosas.	8 de abril de 2021

Instâncias T3 do Amazon EC2 para DAX	O DAX agora oferece suporte ao tipos de instância T3 do Amazon EC2, os quais fornecem um nível de referência de performance da CPU com a capacidad e de intermitência acima da linha de referência quando necessário.	15 de fevereiro de 2021
Compatibilidade do NoSQL Workbench for Amazon DynamoDB com PartiQL	Agora, você pode usar o NoSQL Workbench for DynamoDB para criar instruções PartiQL for DynamoDB.	4 de dezembro de 2020
PartiQL for DynamoDB	Agora você pode usar a PartiQL for DynamoDB, uma linguagem de consultas compatível com SQL, para interagir com tabelas do DynamoDB e executar consultas ad hoc usando o AWS Management Console, a AWS Command Line Interface e as APIs do DynamoDB for PartiQL.	23 de novembro de 2020
Amazon Kinesis Data Streams for Amazon DynamoDB	Agora você pode usar o Amazon Kinesis Data Streams para Amazon DynamoDB com suas tabelas do DynamoDB para capturar alterações em nível de item e replicá-las em um fluxo de dados do Kinesis.	23 de novembro de 2020

Exportação de tabelas do DynamoDB	Agora você pode exportar suas tabelas do DynamoDB para Amazon S3 para poder realizar análises e consultas complexas em seus dados com serviços como Athena, AWS Glue e Lake Formation.	9 de novembro de 2020
Compatibilidade para valores vazios	O DynamoDB agora oferece suporte a valores vazios para atributos String e Binary não chaves em tabelas do DynamoDB. O suporte a valores vazios oferece maior flexibilidade para usar atributos para um conjunto mais amplo de casos de uso sem precisar transform ar esses atributos antes de enviá-los para o DynamoDB. Os tipos de dados List, Map e Set também oferecem suporte a valores String e Binary vazios.	18 de maio de 2020
Compatibilidade do NoSQL Workbench for Amazon DynamoDB para Linux	O NoSQL Workbench para Amazon DynamoDB agora é compatível com as distribui ções do Linux Ubuntu, Fedora e Debian.	4 de maio de 2020

CloudWatch Contributor Insights for DynamoDB: GA	O CloudWatch Contributor Insights for DynamoDB está disponível para o público em geral. O CloudWatch Contributor Insights for DynamoDB é uma ferramenta de diagnóstico que fornece uma visão rápida das tendências de tráfego da tabela do DynamoDB e ajuda a identificar as chaves acessadas com maior frequência da tabela (também conhecidas como teclas de atalho).	2 de abril de 2020
Atualizar as tabelas globais	Agora você pode atualizar suas tabelas globais da versão 2017.11.29 para a versão mais recente das tabelas globais (2019.11.21) com apenas alguns cliques no console do DynamoDB. Ao atualizar a versão das suas tabelas globais, você pode aumentar a disponibilidade das suas tabelas do DynamoDB facilmente estendendo suas tabelas existentes para regiões da AWS adicionais, sem precisar reconstruir tabelas.	16 de março de 2020

NoSQL Workbench for Amazon DynamoDB: GA	O NoSQL Workbench for Amazon DynamoDB está disponível para o público em geral. Use o NoSQL Workbench para projetar, criar, consultar e gerenciar tabelas do DynamoDB.	2 de março de 2020
Métricas de cluster de cache do DAX	Suporte do DAX a novas métricas do CloudWatch, as quais permitem entender melhor a performance do cluster do DAX.	6 de fevereiro de 2020
CloudWatch Contributor Insights for DynamoDB: demonstração	O CloudWatch Contributor Insights for DynamoDB é uma ferramenta de diagnóstico que fornece uma visão rápida das tendências de tráfego da tabela do DynamoDB e ajuda a identificar as chaves acessadas com maior frequência da tabela (também conhecidas como teclas de atalho).	26 de novembro de 2019

Compatibilidade com capacidade adaptativa para workloads desbalanceadas	A capacidade adaptativa do Amazon DynamoDB agora lida com workloads desbalanceadas ao isolar melhor e de forma automática os itens acessados com frequência. Se sua aplicação direciona tráfego desproporcionalmente alto para um ou mais itens, o DynamoDB rebalanceará as partições de modo que os itens acessados com frequência não residam na mesma partição.	26 de novembro de 2019
Compatibilidade com as chaves gerenciadas pelo cliente	O DynamoDB agora é compatível com as chaves gerenciadas pelo cliente, o que significa que você pode ter controle total sobre como criptografar e gerenciar a segurança de seus dados do DynamoDB.	25 de novembro de 2019
Compatibilidade do NoSQL Workbench com DynamoDB Local (versão para download)	Agora o NoSQL Workbench comporta a conexão com o DynamoDB local (versão para download) para desenvolver, criar, consultar e gerenciar tabelas do DynamoDB.	8 de novembro de 2019

NoSQL Workbench: demonstração	Essa é a versão inicial do NoSQL Workbench for DynamoDB. Use o NoSQL Workbench para projetar, criar, consultar e gerenciar tabelas do DynamoDB. Para obter mais informações, consulte NoSQL Workbench para Amazon DynamoDB (demonstração).	16 de setembro de 2019
O DAX agora comporta operações transacionais usando Python e .NET.	O DAX oferece suporte às APIs `TransactWriteItems` e `TransactGetItems` para aplicações escritas em Go, Java, .NET, Node.js e Python. Para obter mais informações, consulte Aceleração na memória com DAX.	14 de fevereiro de 2019
Atualizações do Amazon DynamoDB local (versão para download)	O DynamoDB local (versão para download) agora comporta APIs transacionais, capacidade de leitura/gravação sob demanda, relatórios de capacidade para operações de leitura e gravação e 20 índices secundários globais. Para obter mais informações, consulte Diferenças entre o DynamoDB para download e serviço da Web DynamoDB.	4 de fevereiro de 2019

Amazon DynamoDB sob demanda	DynamoDB sob demanda é uma opção de faturamento flexível capaz de servir centenas de solicitações por segundo sem planejamento de capacidade. O DynamoDB sob demanda oferece definição de preço “pagamento por solicitação” para solicitações de leitura e gravação, para que você pague apenas pelo o que usar. Para ter mais informações, consulte DynamoDB throughput capacity.	28 de novembro de 2018
Amazon DynamoDB Transactions	As transações do DynamoDB realizam alterações de tudo ou nada em vários itens dentro e entre tabelas, fornecendo atomicidade, consistências, isolamento e durabilidade (ACID) no DynamoDB. Para obter mais informações, consulte Amazon DynamoDB Transactions.	27 de novembro de 2018

O Amazon DynamoDB criptografa todos os dados ociosos do cliente	A criptografia de DynamoDB em repouso fornece uma camada adicional de proteção de dados mantendo seus dados seguros na tabela criptografada, incluindo sua chave principal, índices secundários local e global, fluxos, tabelas globais, backups e clusters de DAX sempre que dados forem armazenados em mídia durável. Para obter mais informações, consulte Criptografia em repouso do Amazon DynamoDB.	15 de novembro de 2018
Use o Amazon DynamoDB local com maior facilidade com a nova imagem do Docker	Agora, com a nova imagem do Docker do DynamoDB local, ficou mais fácil usar o DynamoDB local, a versão para download do DynamoDB, para obter ajuda em desenvolvimento e teste de aplicações do DynamoDB. Para obter mais informações, consulte DynamoDB (Versão baixável) e Docker.	22 de agosto de 2018

O DynamoDB Accelerator (DAX) agora comporta criptografia em repouso	O DynamoDB Accelerator (DAX) agora oferece suporte a criptografia em repouso para novos clusters DAX para ajudar a acelerar leituras das tabelas do Amazon DynamoDB em aplicações com segurança frágil que estão sujeitas a conformidade e requisitos regulatórios rigorosos. Para obter mais informações, consulte Criptografia de DAX em repouso.	9 de agosto de 2018
A recuperação em um ponto anterior no tempo (PITR) do DynamoDB agora comporta restauração de tabelas excluídas	Se você excluir uma tabela com a recuperação point-in-time habilitada, um backup do sistema será criado automaticamente e será mantido por 35 dias (sem custo adicional). Para obter mais informações, consulte Antes de começar a usar a recuperação point-in-time.	7 de agosto de 2018
Atualizações agora disponíveis em RSS	Agora, é possível assinar o feed RSS (no canto superior esquerdo desta página) para receber notificações sobre atualizações do Guia do desenvolvedor do Amazon DynamoDB.	3 de julho de 2018

Atualizações anteriores

A tabela a seguir descreve as alterações importantes em cada versão do Guia do desenvolvedor do DynamoDB antes de 3 de julho de 2018.

Alteração	Descrição	Alterado em
Suporte à linguagem Go para o DAX	Agora, você pode habilitar a performance de leitura em microssegundos para as tabelas do Amazon DynamoDB em suas aplicações escritas na linguagem de programação Go usando o novo DynamoDB Accelerator (DAX) SDK for Go. Para obter mais informações, consulte DAX SDK for Go.	26 de junho de 2018
DynamoDB anuncia o SLA	O DynamoDB lançou um Acordo de Nível de Serviço (SLA) de disponibilidade pública. Para obter mas informações, consulte o Acordo de nível de serviço do Amazon DynamoDB.	19 de junho de 2018
Backups contínuos do DynamoDB e recuperação em um ponto anterior no tempo (PITR)	A recuperação point-in-time ajuda a proteger as tabelas do Amazon DynamoDB de operações acidentais de gravação ou exclusão. Com a recuperação point-in-time, você não precisa se preocupar com a criação, a manutenção ou a programação de backups sob demanda.	25 de abril de 2018

Alteração	Descrição	Alterado em
	Por exemplo, suponhamos que um script de teste seja gravado acidentalmente em uma tabela do DynamoDB de produção. Com a recuperaç ão point-in-time, você pode recuperar a tabela para qualquer ponto durante os últimos 35 dias. O DynamoDB mantém backups increment ais da tabela. Para obter mais informações, consulte Backups para um ponto no tempo do DynamoDB.	
Criptografia em repouso para DynamoDB	A criptografia em repouso do DynamoDB, disponível para novas tabelas do DynamoDB, ajuda você proteger os dados da sua aplicação em tabelas do Amazon DynamoDB usando as chaves de criptogra fia gerenciadas pela AWS armazenadas no AWS Key Management Service. Para obter mais informações, consulte Criptografia em repouso do DynamoDB.	8 de fevereiro de 2018

Alteração	Descrição	Alterado em
Backup e restauração do DynamoDB	O backup sob demanda permite que você crie backups completos dos dados das tabelas do DynamoDB para arquivamento de dados, o que ajudará você a atender aos requisitos regulamen tares de sua corporação e do governo. Você pode fazer backup de tabelas de alguns megabytes para centenas de terabytes de dados, sem impactar o desempenho e a disponibilidade dos aplicativ os de produção. Para obter mais informações, consulte Backup e restauração para o DynamoDB.	29 de novembro de 2017

Alteração	Descrição	Alterado em
Tabelas globais do DynamoDB	As tabelas globais aproveitam a presença global do DynamoDB para oferecer a você um banco de dados totalmente gerenciado multiativo e com várias regiões que fornece performance rápida e local de leitura e gravação para aplicações globais com escalabilidade muito alta. As tabelas globais replicam as tabelas do Amazon DynamoDB automaticamente nas regiões da AWS escolhidas por você. Para obter mais informações, consulte Tabelas globais: replicação multiativa e multirregional.	29 de novembro de 2017
Suporte de Node.js para DAX	Os desenvolvedores Node.js podem utilizar o Amazon DynamoDB Accelerator (DAX), usando o cliente do DAX para Node.js. Para obter mais informações, consulte Aceleração em memória com o DynamoDB Accelerator (DAX).	5 de outubro de 2017

Alteração	Descrição	Alterado em
Endpoints da VPC para DynamoDB	Os endpoints do DynamoDB permitem que instâncias do Amazon EC2 na sua Amazon VPC acessem o DynamoDB sem exposição à Internet pública. O tráfego de rede entre sua VPC e o DynamoDB não sai da rede da Amazon. Para obter mais informações, consulte Usar endpoints da Amazon VPC para acessar o DynamoDB.	16 de agosto de 2017

Alteração	Descrição	Alterado em
Auto Scaling para DynamoDB	O Auto Scaling do DynamoDB elimina a necessidade de definir ou ajustar manualmente as configurações de throughput provisionado. Em vez disso, o Auto Scaling do DynamoDB ajusta dinamicamente a capacidade de leitura e gravação em resposta a padrões de tráfego reais. Isso permite que uma tabela ou um índice secundário global aumente a capacidade provisionada de leitura e gravação para processar aumentos repentinos no tráfego, sem limitações. Quando a workload diminui, o Auto Scaling do DynamoDB diminui a capacidade provisionada. Para obter mais informações, consulte Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB.	14 de junho de 2017

Alteração	Descrição	Alterado em
DynamoDB Accelerator (DAX)	O DynamoDB Accelerator (DAX) é um cache de memória totalmente gerenciado e altamente disponível para o DynamoDB que proporciona melhorias até 10 vezes maiores na performance, de milissegundos para microssegundos, mesmo com milhões de solicitações por segundo. Para obter mais informações, consulte Aceleração em memória com o DynamoDB Accelerator (DAX).	19 de abril de 2017
O DynamoDB agora oferece suporte à expiração automática de itens com vida útil (TTL)	A configuração de vida útil (TTL) do Amazon DynamoDB permite excluir automaticamente itens expirados de suas tabelas, sem custo adicional. Para obter mais informações, consulte Usar a vida útil (TTL) no DynamoDB.	27 de fevereiro de 2017
O DynamoDB agora oferece suporte para Tags de alocação de custos	Você pode adicionar tags às suas tabelas do Amazon DynamoDB para melhorar a categorização de uso e ter relatórios de custo mais granulares. Para obter mais informações, consulte Adicionar etiquetas e rótulos a recursos no DynamoDB.	19 de janeiro de 2017

Alteração	Descrição	Alterado em
Nova API `DescribeLimits` do DynamoDB	A API `DescribeLimits` retorna os limites de capacidade provisionada atuais para a sua conta da AWS em uma região, tanto para a região como um todo quanto para qualquer tabela do DynamoDB criada nela. Ele permite que determinar quais são os seus limites de nível de conta atuais, para que você possa compará-los com a capacidade provisionada que está usando no momento e tenha tempo de sobra para solicitar um aumento antes de atingir um limite. Para obter mais informações, consulte Cotas no Amazon DynamoDB e DescribeLimits na Referência da API do Amazon DynamoDB.	1 de março de 2016

Alteração	Descrição	Alterado em
Atualização do console do DynamoDB e novas terminologias para atributos de chave primária	O console de gerenciamento do DynamoDB foi reprojetado para ser mais intuitivo e fácil de usar. Como parte dessa atualização, estamos introduzindo novas terminologias para os atributos de chave primária: • Chave de partição – também conhecida como um atributo de hash. • Chave de classificação – também conhecida como um atributo de intervalo. Apenas os nomes foram alterados; a funcionalidade permanece a mesma. Ao criar uma tabela ou um índice secundário, você pode escolher uma chave primária simples (apenas a chave de partição) ou uma chave primária composta (chave de partição e chave de classificação). A documentação do DynamoDB foi atualizada para refletir essas alterações.	12 de novembro de 2015

Alteração	Descrição	Alterado em
Amazon DynamoDB Storage Backend for Titan	O DynamoDB Storage Backend for Titan é um backend de armazenamento para o banco de dados de grafos do Titan implement ado em cima do Amazon DynamoDB. Quando o DynamoDB Storage Backend for Titan é usado, seus dados se beneficiam da proteção do DynamoDB, o qual é executado nos datacente rs de alta disponibilidade da Amazon. O plug-in está disponível para o Titan versão 0.4.4 (principalmente para compatibilidade com aplicativ os existentes) e para o Titan versão 0.5.4 (recomend ado para novos aplicativos). Como outros back-ends de armazenamento para Titan, esse plug-in oferece suporte à pilha Tinkerpop (versões 2.4 e 2.5), incluindo a API Blueprints e o shell Gremlin.	20 de agosto de 2015

Alteração	Descrição	Alterado em
DynamoDB Streams, replicação entre regiões e verificação com leituras fortemente consistentes	O DynamoDB Streams captura uma sequência em ordem temporal de modificações em nível de item em qualquer tabela do DynamoDB e armazena essas informações em um log por até 24 horas. Os aplicativos podem acessar esse log e visualizar os itens de dados à medida que eles aparecem antes e depois que foram modificados, em tempo quase real. Para obter mais informações, consulte Capturar dados de alterações para o DynamoDB Streams e a Referência da API do DynamoDB Streams. A replicação entre regiões do DynamoDB é uma solução no lado do cliente para manter cópias idênticas de tabelas do DynamoDB em diferentes regiões da AWS, quase em tempo real. Você pode usar a replicação entre regiões para fazer backup de tabelas do DynamoDB ou fornecer acesso de baixa latência aos dados da região em que os usuários estão distribuídos geograficamente.	16 de julho de 2015

Alteração	Descrição	Alterado em
	A operação `Scan` do DynamoDB usa leituras finais consistentes por padrão. Você pode usar leituras fortemente consistentes em vez disso, definindo o parâmetro `ConsistentRead` como true. Para obter mais informações, consulte Consistência de leitura para verificação e Scan na Referência da API do Amazon DynamoDB.	
Suporte da AWS CloudTrail ao Amazon DynamoDB	O DynamoDB agora está integrado ao CloudTrail. O CloudTrail captura chamadas de API feitas do console do DynamoDB ou da DynamoDB API e as rastreia em arquivos de log. Para obter mais informações, consulte Registrar em log as operações do DynamoDB usando o AWS CloudTrail e o Guia do usuário do AWS CloudTrail.	28 de maio de 2015

Alteração	Descrição	Alterado em
Suporte aprimorado para expressões de consulta	Esta versão adiciona um novo parâmetro `KeyConditionExpression` à API `Query`. Uma `Query` lê itens de uma tabela ou um de índice usando valores de chave primária. O parâmetro `KeyConditionExpression` é uma string que identifica nomes de chaves primárias e as condições a serem aplicada aos valores de chaves; o `Query` recupera somente os itens que atendem à expressão . A sintaxe de `KeyConditionExpression` é semelhante à de outros parâmetros de expressão no DynamoDB e permite que você defina variáveis de substituição de nomes e valores dentro da expressão. Para obter mais informações, consulte Consultar tabelas no DynamoDB.	27 de abril de 2015

Alteração	Descrição	Alterado em
Novas funções de comparação para gravações condicionais	No DynamoDB, o parâmetro `ConditionExpression` determina se uma operação `PutItem`, `UpdateItem` ou `DeleteItem` é bem-sucedida: o item apenas será gravado se a condição for avaliada como true. Esta versão adiciona duas novas funções, `attribute_type` e `size`, para uso com `ConditionExpression`. Essas funções permitem que você realize gravações condicionais com base no tipo de dados ou no tamanho de um atributo em uma tabela. Para obter mais informações, consulte Exemplo de expressão de condição do DynamoDB na CLI.	27 de abril de 2015

Alteração	Descrição	Alterado em
API Scan para índices secundários	No DynamoDB, uma operação `Scan` lê todos os itens em uma tabela, aplica critérios de filtragem definidos pelo usuário e retorna os itens de dados selecionados à aplicação. Essa mesma capacidade agora também está disponível para índices secundários. Para verificar um índice secundário local ou um índice secundário global, você especifica o nome do índice e o nome de sua tabela principal . Por padrão, uma `Scan` de índice retorna todos os dados do índice. Você pode usar uma expressão de filtro para restringir os resultados que são retornados ao aplicativo. Para obter mais informações, consulte Verificar tabelas no DynamoDB.	10 de fevereiro de 2015

Alteração	Descrição	Alterado em
Operações online para índices secundários globais	A indexação online permite que você adicione ou remova índices secundários globais em tabelas existentes. Com a indexação online, não é necessário definir todos os índices de uma tabela quando você cria uma tabela. Em vez disso, é possível adicionar um novo índice a qualquer momento. Da mesma forma, se você decidir que não precisa mais de um índice, pode removê-lo a qualquer momento. Operações de indexação online são não bloqueantes e, portanto, a tabela permanece disponíve l para atividades de leitura e gravação enquanto índices estão sendo adicionados ou removidos. Para obter mais informações, consulte Gerenciar índices secundários globais no DynamoDB.	27 de janeiro de 2015

Alteração	Descrição	Alterado em
Suporte para modelo de documento com JSON	O DynamoDB permite armazenar e recuperar documentos com suporte completo para modelos de documento. Novos tipos de dados são totalmente compatíveis com o padrão JSON e permitem que você aninhe elementos de documento uns dentro dos outros. É possível usar operadores de desreferência de caminho de documento para ler e gravar elementos individuais, sem precisar recuperar o documento inteiro. Essa versão também apresenta novos parâmetros de expressão para especific ar projeções, condições e ações de atualização ao ler ou gravar itens de dados. Para saber mais sobre o suporte a modelos de documentos com o JSON, consulte Tipos de dadoseUsar expressões no DynamoDB.	7 de outubro de 2014

Alteração	Descrição	Alterado em
Dimensionamento flexível	Para tabelas e índices secundários globais, você pode aumentar a capacidad e de throughput provision ada de leitura e gravação em qualquer quantidade, desde que permaneça nos seus limites por tabela e por conta. Para obter mais informações, consulte Cotas no Amazon DynamoDB.	7 de outubro de 2014
Tamanhos maiores de itens	O tamanho máximo de itens no DynamoDB aumentou de 64 KB para 400 KB. Para obter mais informações, consulte Cotas no Amazon DynamoDB.	7 de outubro de 2014

Alteração	Descrição	Alterado em
Expressões condicionais aprimoradas	O DynamoDB expande os operadores disponíveis para expressões condicionais, dando a você flexibilidade adicional para inserções, atualizações e exclusões condicionais. Os operadores recém-disponibilizados permitem que você verifique se um atributo existe ou não, é maior ou menor que um determinado valor, está entre dois valores, começa com certos caracteres e muito mais. O DynamoDB também fornece um operador OR opcional para avaliar várias condições. Por padrão, várias condições em uma expressão são unidas por AND e, portanto, a expressão apenas será verdadeira se todas as suas condições também forem verdadeiras. Se você especificar OR em vez disso, a expressão será verdadeira se uma ou mais condições forem verdadeiras. Para obter mais informações, consulte Trabalhar com itens e atributos no DynamoDB.	24 de abril de 2014

Alteração	Descrição	Alterado em
Filtro de consulta	A API `Query` do DynamoDB oferece suporte a uma nova opção `QueryFilter` . Por padrão, um `Query` localiza itens que correspondem a um valor de chave de partição específico e uma condição de chave de classificação opcional. Um filtro `Query` aplica expressões condicion ais a outros atributos não chave. Se um filtro `Query` estiver presente, os itens que não corresponderem às condições de filtro serão descartados antes que os resultados de `Query` sejam retornados para o aplicativo. Para obter mais informações, consulte Consultar tabelas no DynamoDB.	24 de abril de 2014

Alteração	Descrição	Alterado em
Exportação e importação de dados usando o AWS Management Console	O console do DynamoDB foi aprimorado para simplificar exportações e importações de dados em tabelas do DynamoDB. Com apenas alguns cliques, você pode configurar um AWS Data Pipeline para orquestrar o fluxo de trabalho e um cluster do Amazon Elastic MapReduce para copiar dados de tabelas do DynamoDB para um bucket do Amazon S3 ou vice-versa. É possível executar uma exportação ou importação somente uma vez, ou configurar um trabalho de exportação diário. Você pode até mesmo executar exportações e importações entre regiões copiando os dados do DynamoDB de uma tabela em uma região da AWS para uma tabela em outra região da AWS.	6 de março de 2014

Alteração	Descrição	Alterado em
Documentação de APIs de nível superior reorganizada	Agora é mais fácil localizar informações sobre as seguintes APIs: • Java: DynamoDBMappper • .NET: Modelo de documento e modelo de persistência de objetos Essas APIs de nível mais alto agora estão documentadas aqui: Interfaces de programação de nível superior para o DynamoDB.	20 de janeiro de 2014

Alteração	Descrição	Alterado em
Índices secundários globais	O DynamoDB adiciona suporte a índices secundários globais. Como no caso de um índice secundário local, você define um índice secundário global, usando uma chave alternativa de uma tabela e emitindo solicitações Query no índice. Ao contrário de um índice secundário local, a chave de partição do índice secundário global não precisa ser igual à da tabela. Ela pode ser qualquer atributo escalar da tabela. A chave de classificação é opcional e também pode ser qualquer atributo de tabela escalar. Um índice secundário global também tem suas próprias configurações de throughput provisionado, que são separadas daquelas da tabela principal. Para obter mais informações, consulte Melhorar o acesso aos dados com índices secundários no DynamoDB e Como usar índices secundários globais no DynamoDB.	12 de dezembro de 2013

Alteração	Descrição	Alterado em
Controle de acesso refinado	O DynamoDB adiciona suporte para controle de acesso refinado. Esse recurso permite que os clientes especifiquem quais entidades (usuários, grupos ou funções) podem acessar itens e atributos individuais em uma tabela ou índice secundário do DynamoDB. Os aplicativos também podem tirar proveito da federação de identidades da Web para descarregar a tarefa de autenticação de usuários para um provedor de identidade de terceiros, como o Facebook, o Google ou o Login with Amazon. Dessa forma, as aplicações (incluindo aplicações móveis) podem lidar com um grande número de usuários e, ao mesmo tempo, garantir que ninguém possa acessar itens de dados do DynamoDB sem autorização. Para obter mais informações, consulte Uso de condições de política do IAM para controle de acesso refinado.	29 de outubro de 2013

Alteração	Descrição	Alterado em
4 KB de tamanho da unidade de capacidade de leitura	O tamanho da unidade de capacidade para leituras aumentou de 1 KB para 4 KB. Esse aumento pode reduzir o número de unidades de capacidade de leitura provisionadas necessárias para muitos aplicativos. Por exemplo, antes desta versão, a leitura de um item de 10 KB consumiria 10 unidades de capacidade de leitura. Agora, a mesma leitura de 10 KB consumiria apenas 3 unidades (10 KB/4 KB, arredondados para o próximo limite de 4 KB). Para obter mais informações, consulte Capacidade de throughput do DynamoDB.	14 de maio de 2013

Alteração	Descrição	Alterado em
Verificações paralelas	O DynamoDB adiciona suporte a operações Scan paralelas. Agora, os aplicativos podem dividir uma tabela em segmentos lógicos e verificar todos esses segmentos simultaneamente. Esse recurso reduz o tempo necessário para uma a conclusão de uma verificação e utiliza totalmente a capacidade de leitura provisionada da tabela. Para obter mais informações, consulte Verificar tabelas no DynamoDB.	14 de maio de 2013
Índices secundários locais	O DynamoDB adiciona suporte a índices secundários locais. Você pode definir índices de tipo de chave em atributos não chave e depois usar esses índices em solicitações de Consulta. Com os índices secundários locais, as aplicações podem recuperar itens de dados eficientemente entre várias dimensões. Para obter mais informações, consulte Índices secundários locais no DynamoDB.	18 de abril de 2013

Alteração	Descrição	Alterado em
Nova versão de API	Com este lançamento, o DynamoDB apresenta uma nova versão da API (2012-08-10). A versão anterior da API (2011-12-05) ainda tem suporte para compatibilidade com aplicativos existente s. Novos aplicativos devem usar a nova API de versão 2012-08-10. Recomenda mos migrar suas aplicações existentes para a API versão 2012-08-10, pois os novos recursos do DynamoDB (como índices secundários) não serão portados para compatibilidade com a versão anterior da API. Para obter mais informações sobre a API versão 2012-08-10, consulte a Referência da API do Amazon DynamoDB.	18 de abril de 2013

Alteração	Descrição	Alterado em
Suporte para variáveis de política do IAM	Agora, a linguagem de política de acesso do IAM oferece suporte para variáveis . Quando uma política é avaliada, todas as variáveis de política são substituídas por valores fornecidos por informações baseadas no contexto da sessão do usuário autenticado. Você pode usar variáveis de política para definir políticas de uso geral sem, explicitamente, listar todos os componentes da política. Para obter mais informações sobre variáveis de políticas, vá para Variáveis de políticas no guia Uso do AWS Identity and Access Management (IAM). Para obter exemplos de variáveis de políticas no DynamoDB, consulte Gerenciamento de identidade e acesso no Amazon DynamoDB.	4 de abril de 2013

Alteração	Descrição	Alterado em
Exemplos de código PHP atualizados para o AWS SDK para PHP versão 2	A versão 2 do AWS SDK para PHP está disponível. Os exemplos de código PHP no Guia do desenvolvedor do Amazon DynamoDB foram atualizados para usar esse novo SDK. Para obter mais informações sobre a versão 2 do SDK, consulte AWS SDK para PHP.	23 de janeiro de 2013
Novo endpoint	O DynamoDB é expandido para a região da AWS GovCloud (Oeste dos EUA). Para obter a lista atual de endpoints de serviços e protocolos, consulte Regiões e endpoints.	3 de dezembro de 2012
Novo endpoint	O DynamoDB é expandido para a região América do Sul (São Paulo). Para obter a lista atual de endpoints com suporte, consulte Regiões e endpoints.	3 de dezembro de 2012
Novo endpoint	O DynamoDB é expandido para a região Ásia-Pací fico (Sydney). Para obter a lista atual de endpoints com suporte, consulte Regiões e endpoints.	13 de novembro de 2012

Alteração	Descrição	Alterado em
O DynamoDB implementa suporte a somas de verificação CRC32, oferece suporte para obtenções de lote fortemente consistentes e remove restrições em atualizações de tabelas simultâneas.	• O DynamoDB calcula uma soma de verificação CRC32 da carga útil HTTP e retorna essa soma em um novo cabeçalho, `x-amz-crc32` . Para obter mais informações, consulte API de baixo nível do DynamoDB. • Por padrão, as operações de leitura realizadas pela API `BatchGetItem` são eventualmente consistentes. Um novo parâmetro `ConsistentRead` em `BatchGetItem` permite que você escolha a consistência de leitura forte para qualquer tabela na solicitação. Para obter mais informações, consulte Descrição. • Essa versão remove certas restrições ao atualizar muitas tabelas simultaneamente. O número total de tabelas que podem ser atualizadas ao mesmo tempo ainda é 10. Mo entanto, essas tabelas agora podem ser qualquer combinação de status `CREATING`, `UPDATING` ou `DELETING` Além disso, não existe mais uma quantidad	2 de novembro de 2012

Alteração	Descrição	Alterado em
	e mínima para aumentar ou reduzir os valores de ReadCapacityUnits ou WriteCapacityUnits para uma tabela. Para obter mais informações, consulte Cotas no Amazon DynamoDB.	
Documentação de práticas recomendadas	O Guia do desenvolvedor do Amazon DynamoDB identifica as práticas recomendadas para trabalhar com tabelas e itens, juntamente com recomendações para operações de consulta e verificação.	28 de setembro de 2012

Alteração	Descrição	Alterado em
Suporte para tipos de dados binário	Além dos tipos Número e String, o DynamoDB agora oferece suporte ao tipo de dados Binary. Antes desta versão, para armazenar dados binários, era necessário converter dados binários no formato de string e os armazenar no DynamoDB. Além do trabalho de conversão necessário no lado do cliente, a conversão muitas vezes aumentava o tamanho do item de dados, exigindo mais armazenamento e possivelmente capacidad e adicional de throughput provisionado. Agora, com atributos do tipo Binário, você pode armazenar quaisquer dados binários, por exemplo, dados compactados, dados criptogra fados e imagens. Para obter mais informações, consulte Tipos de dados. Para obter exemplos funcionais de como manipular dados do tipo Binary usando os AWS SDKs, consulte as seções a seguir: • Exemplo: tratar atributos do tipo binário usando a API de	21 de agosto de 2012

Alteração	Descrição	Alterado em
	documento do AWS SDK para Java • Exemplo: tratar atributos do tipo binário usando a API de baixo nível do AWS SDK para .NET Para obter suporte para o tipo de dados Binary adicionado nos AWS SDKs, você precisará fazer download dos SDKs mais recentes. Talvez também seja necessário atualizar todas as aplicações existentes. Para obter informações sobre como baixar os AWS SDKs, consulte Exemplos de código .NET.	
Itens de tabela do DynamoDB podem ser atualizados e copiados usando o console do DynamoDB	Agora, os usuários do DynamoDB podem atualizar e copiar itens de tabela usando o console do DynamoDB além de serem capazes de adicionar e excluir itens. Essa nova funcionalidade simplifica o processo de fazer alterações em itens individuais usando o Console.	14 de agosto de 2012

Alteração	Descrição	Alterado em
O DynamoDB reduz os requisitos mínimos de throughput da tabela	O DynamoDB agora oferece suporte a um número menor de requisitos mínimos de throughput da tabela, ou seja, 1 unidade de capacidade de gravação e 1 unidade de capacidade de leitura. Para obter mais informações, consulte o tópico Cotas no Amazon DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.	9 de agosto de 2012
Suporte para o Signature Version 4	O DynamoDB agora oferece suporte ao Signature Version 4 para autenticar solicitações.	5 de julho de 2012
Suporte ao explorador de tabelas no console do DynamoDB	O console do DynamoDB agora oferece suporte a um explorador de tabelas que permite navegar e consultar dados em tabelas. Você também pode inserir novos itens ou excluir itens existentes. As seções `SampleData` e Utilizar o console foram atualizadas para esses recursos.	22 de maio de 2012

Alteração	Descrição	Alterado em
New endpoints (Novos endpoints)	A disponibilidade do DynamoDB é expandida com novos endpoints nas regiões Oeste dos EUA (Norte da Califórnia), Oeste dos EUA (Oregon) e Ásia-Pacífico (Singapura). Para obter a lista atual de endpoints com suporte, acesse Regiões e endpoints.	24 de abril de 2012
Suporte à API BatchWriteItem	O DynamoDB agora oferece suporte a uma API de gravação em lote que permite que você insira e exclua vários itens de uma ou mais tabelas em uma única chamada a API. Para obter mais informações sobre a API de gravação em lote do DynamoDB, consulte BatchWriteItem. Para obter informações sobre como trabalhar com itens e usar o recurso de gravação em lote com AWS SDKs, consulte Trabalhar com itens e atributos no DynamoDB e Exemplos de código .NET.	19 de abril de 2012
Mais códigos de erro documentados	Para obter mais informações, consulte Tratamento de erros com o DynamoDB.	5 de abril de 2012

Alteração	Descrição	Alterado em
Novo endpoint	O DynamoDB é expandido para a região Ásia-Pacífico (Tóquio). Para obter a lista atual de endpoints com suporte, consulte Regiões e endpoints.	29 de fevereiro de 2012
Métrica `ReturnedItemCount` adicionada	Uma nova métrica `ReturnedItemCount` fornece o número de itens retornados na resposta de uma operação Query ou Scan para o DynamoDB e está disponível para monitoramento por meio do CloudWatch.	24 de fevereiro de 2012
Exemplos adicionados para incrementar valores	O DynamoDB oferece suporte a operações para incrementar e diminuir valores numéricos existentes. Exemplos mostram como somar a valores existentes nas seções "Como atualizar um item" em: Trabalhar com itens: Java. Trabalhar com itens: .NET.	25 de janeiro de 2012
Versão inicial do produto	O DynamoDB é apresentado como um novo serviço em versão Beta.	18 de janeiro de 2012

Recursos herdados do DynamoDB

Os tópicos a seguir são recursos herdados que o DynamoDB ainda comporta. Nenhum desenvolvimento ativo é feito nesses recursos.

Tópicos

- Global Tables versão 2017.11.29 (herdada)
- Versão anterior da API de baixo nível do DynamoDB (5/12/2011)
- Parâmetros condicionais legados do DynamoDB

Global Tables versão 2017.11.29 (herdada)

Important

Esta documentação, destinada à versão 2017.11.29 (herdada) de tabelas globais, deve ser evitada para novas tabelas globais. Os clientes devem usar Global Tables versão 2019.11.21 (atual) sempre que possível, pois ela oferece maior flexibilidade e eficiência, além de consumir menor capacidade de gravação que a 2017.11.29 (herdada).
Para determinar qual versão você está usando, consulte Determinar a versão de uma tabela global. Para atualizar tabelas globais existentes da versão 2017.11.29 (herdada) para a versão 2019.11.21 (atual), consulte Versões de tabelas globais do DynamoDB.

Tópicos

- Tabelas globais: como funcionam
- Práticas recomendadas e requisitos para gerenciar tabelas globais
- Criar uma tabela global
- Monitorar tabelas globais
- Usar o IAM com tabelas globais

Tabelas globais: como funcionam

> **Important**
>
> Esta documentação, destinada à versão 2017.11.29 (herdada) de tabelas globais, deve ser evitada para novas tabelas globais. Os clientes devem usar Global Tables versão 2019.11.21 (atual) sempre que possível, pois ela oferece maior flexibilidade e eficiência, além de consumir menor capacidade de gravação que a 2017.11.29 (herdada).
> Para determinar qual versão você está usando, consulte Determinar a versão de uma tabela global. Para atualizar tabelas globais existentes da versão 2017.11.29 (herdada) para a versão 2019.11.21 (atual), consulte Versões de tabelas globais do DynamoDB.

As seções a seguir ajudam você a entender os conceitos e o comportamento das tabelas globais no Amazon DynamoDB.

Conceitos de tabela global para a versão 2017.11.29 (herdada)

Uma tabela global é a coleção de uma ou mais tabelas-réplica, todas pertencentes a uma única conta da AWS.

Uma tabela-réplica (ou simplesmente réplica) é uma única tabela do DynamoDB que funciona como parte de uma tabela global. Cada réplica armazena o mesmo conjunto de itens de dados. Qualquer tabela global só pode ter uma tabela-réplica por região da AWS.

Veja a seguir uma visão geral conceitual de como uma tabela global é criada.

1. Crie uma tabela comum do DynamoDB, com o DynamoDB Streams habilitado, em uma região da AWS.
2. Repita a etapa 1 para todas as outras regiões em que você deseja replicar seus dados.
3. Defina uma tabela global do DynamoDB, com base nas tabelas que você criou.

O AWS Management Console automatiza essas tarefas para que você possa criar uma tabela global de forma rápida e fácil. Para obter mais informações, consulte Criar uma tabela global.

A tabela global resultante do DynamoDB consiste em várias tabelas-réplica, uma por região, que o DynamoDB trata como uma única unidade. Cada réplica possui o mesmo nome de tabela e o mesmo esquema de chave primária. Quando uma aplicação grava dados em uma tabela-réplica em

uma região, o DynamoDB propaga automaticamente a gravação para as outras tabelas-réplica nas demais regiões da AWS.

> **Important**
>
> Para manter os dados da tabela sincronizados, as tabelas globais criam automaticamente os seguintes atributos para cada item:
>
> - `aws:rep:deleting`
> - `aws:rep:updatetime`
> - `aws:rep:updateregion`
>
> Não modifique esses atributos ou crie atributos com o mesmo nome.

Você pode adicionar tabelas-réplica à tabela global para que ela fique disponível em outras regiões. (Para fazer isso, a tabela global precisa estar vazia. Em outras palavras, nenhuma das tabelas-réplica pode conter informações.)

Você também pode remover uma tabela-réplica de uma tabela global. Se você fizer isso, a tabela será completamente desassociada da tabela global. Essa tabela que se tornou independente não interagirá mais com a tabela global, e os dados não serão mais propagados de nem para a tabela global.

> **Warning**
>
> Saiba que a remoção de uma réplica não é um processo atômico. Para garantir um comportamento consistente e o estado conhecido, você pode considerar desviar o tráfego de gravação da aplicação para longe da réplica a ser removida antecipadamente. Depois de removê-la, aguarde até que todos os endpoints da região de réplica mostrem a réplica como desassociada antes de fazer outras gravações nela como sua própria tabela de regiões isolada.

Tarefas comuns

As tarefas comuns para tabelas globais funcionam da maneira a seguir.

Não é possível excluir uma tabela de réplicas de uma tabela global da mesma forma que uma tabela normal. Isso interromperá a replicação nessa região e excluirá a cópia da tabela mantida nessa região. Não é possível romper a replicação e ter cópias da tabela como entidades independentes.

> **Note**
>
> Só será possível excluir uma tabela de origem no mínimo 24 horas depois que ela for usada para iniciar uma nova região. Se você tentar excluí-la antes disso, receberá um erro.

Conflitos poderão ocorrer se as aplicações atualizarem o mesmo item em regiões diferentes e quase ao mesmo tempo. Para garantir a consistência final, as tabelas globais do DynamoDB usam um método do tipo “último gravador ganha” entre as atualizações simultâneas. Todas as réplicas concordarão com a atualização mais recente e serão convertidas em um estado em que todas têm dados idênticos.

> **Note**
>
> Há várias formas de evitar conflitos, incluindo:
>
> - Usar uma política do IAM para permitir somente gravações na tabela de uma região.
> - Usar uma política do IAM para encaminhar usuários a apenas uma região e manter a outra em espera ou rotear alternadamente usuários ímpares para uma região e usuários pares para outra região.
> - Evitar o uso de atualizações não idempotentes, como Marcador = Marcador + 1, em favor de atualizações estáticas, como Marcador = 25.

Monitorar tabelas globais

Você pode usar o CloudWatch para observar a métrica `ReplicationLatency`. Essa métrica monitora o tempo decorrido entre quando um item atualizado aparece em um fluxo do DynamoDB para uma tabela-réplica e quando esse item aparece em outra réplica na tabela global. `ReplicationLatency` é expressa em milissegundos e é emitida para cada par de regiões de origem e destino. Essa é a única métrica do CloudWatch fornecida pelo Global Tables v2.

As latências que você observará dependerão da distância entre as regiões escolhidas, bem como de outras variáveis. Latências na faixa de 0,5 a 2,5 segundos para regiões podem ser comuns na mesma área geográfica.

Vida útil (TTL)

Você pode usar a vida útil (TTL) para especificar um nome de atributo cujo valor indica a hora de expiração do item. Esse valor é especificado como um número em segundos desde o início da época do Unix.

Com a versão herdada das tabelas globais, as exclusões de TTL não são replicadas automaticamente em outras réplicas. Quando um item é excluído por meio de uma regra de TTL, esse trabalho é realizado sem consumir unidades de gravação.

Lembre-se de que, se as tabelas de origem e de destino tiverem uma capacidade de gravação provisionada muito baixa, isso poderá acionar o controle de utilização, pois as exclusões por TTL exigem capacidade de gravação.

Fluxos e transações com tabelas globais

Cada tabela global produz um fluxo independente com base em todas as gravações, seja qual for o ponto de origem dessas gravações. Você pode optar por consumir esse fluxo do DynamoDB em uma região ou em todas as regiões de forma independente.

Se você quiser gravações locais processadas, mas não gravações replicadas, poderá adicionar seu próprio atributo de região a cada item. Depois, você pode usar um filtro de eventos do Lambda para invocar somente o Lambda para gravações na região local.

As operações transacionais fornecem garantia de atomicidade, consistência, isolamento e durabilidade (ACID) SOMENTE na região em que a gravação é realizada originalmente. As transações não são compatíveis entre regiões em tabelas globais.

Por exemplo, se você tiver uma tabela global com réplicas nas regiões Leste dos EUA (Ohio) e Oeste dos EUA (Oregon) e realizar uma operação TransactWriteItems na região Leste dos EUA (Ohio), poderá observar transações parcialmente concluídas na região Oeste dos EUA (Oregon) à medida que as alterações forem replicadas. As alterações só serão replicadas para outras regiões quando forem confirmadas na região de origem.

> **Note**
>
> - As tabelas globais “contornam” o DynamoDB Accelerator atualizando o DynamoDB diretamente. Por isso, o DAX não saberá que está mantendo dados obsoletos. O cache do DAX só será atualizado quando a TTL do cache expirar.
> - As etiquetas em tabelas globais não se propagam automaticamente.

Throughput de leitura e gravação

As tabelas globais gerenciam o throughput de leitura e gravação das maneiras a seguir.

- A capacidade de gravação deve ser a mesma em todas as instâncias de tabela em todas as regiões.
- Com a versão 2019.11.21 (atual), se a tabela estiver configurada para comportar ajuste de escala automático ou estiver no modo sob demanda, a sincronização da capacidade de gravação será automática. A quantidade atual de capacidade de gravação provisionada em cada região aumentará e diminuirá de maneira independente dentro dessas configurações de ajuste de escala automático sincronizado. Se a tabela for colocada no modo sob demanda, esse modo será sincronizado nas outras réplicas.
- A capacidade de leitura pode diferir entre as regiões porque as leituras podem não ser iguais. Ao adicionar uma réplica global a uma tabela, a capacidade da região de origem é propagada. Após a criação, é possível ajustar a capacidade de leitura de uma réplica, e essa nova configuração não é transferida para o outro lado.

Consistência e resolução de conflitos

Todas as alterações feitas em qualquer item de qualquer tabela-réplica serão replicadas para todas as outras réplicas dentro da mesma tabela global. Em uma tabela global, um item recém-gravado geralmente é propagado para todas as tabelas-réplica dentro de poucos segundos.

Com uma tabela global, cada tabela-réplica armazena o mesmo conjunto de itens de dados. O DynamoDB não oferece suporte à replicação parcial de apenas alguns dos itens.

Uma aplicação pode ler e gravar dados em qualquer tabela-réplica. O DynamoDB comporta leituras finais consistentes entre regiões, mas não leituras altamente consistentes entre regiões. Sua

aplicação usar somente leituras finais consistentes e emitir leituras somente para uma região da AWS, ela funcionará sem qualquer modificação. No entanto, se a aplicação exigir leituras altamente consistentes, ela precisará executar todas as leituras e gravações altamente consistentes na mesma região. Caso contrário, se você gravar em uma região e ler em outra, a resposta lida poderá incluir dados obsoletos que não refletirão os resultados das gravações concluídas recentemente na outra região.

Conflitos poderão ocorrer se as aplicações atualizarem o mesmo item em regiões diferentes e quase ao mesmo tempo. Para garantir a consistência eventual, as tabelas globais do DynamoDB usam uma conciliação o último a gravar ganha entre as atualizações simultâneas. Com ela, o DynamoDB emprega o melhor esforço para determinar o último a gravar. Com esse mecanismo de resolução de conflitos, todas as réplicas concordarão com a atualização mais recente e serão convertidas para um estado em que todas têm dados idênticos.

Disponibilidade e durabilidade

Se uma única região da AWS se tornar isolada ou degradada, sua aplicação poderá realizar redirecionamentos para uma região diferente e executar leituras e gravações em uma tabela-réplica diferente. Você pode aplicar lógica de negócios personalizada para determinar quando redirecionar solicitações para outras regiões.

Se uma região se tornar isolada ou degradada, o DynamoDB acompanhará as gravações executadas que ainda não foram propagadas para todas as tabelas-réplica. Quando a região voltar a ficar online, o DynamoDB retomará a propagação de todas as gravações pendentes dessa região para as tabelas-réplica nas outras regiões. Ele também retomará a propagação de gravações de outras tabelas-réplica para a região que está online novamente. Todas as gravações anteriores bem-sucedidas serão propagadas em algum momento, não importa por quanto tempo a região permaneça isolada.

Práticas recomendadas e requisitos para gerenciar tabelas globais

> **Important**
>
> Esta documentação, destinada à versão 2017.11.29 (herdada) de tabelas globais, deve ser evitada para novas tabelas globais. Os clientes devem usar Global Tables versão 2019.11.21 (atual) sempre que possível, pois ela oferece maior flexibilidade e eficiência, além de consumir menor capacidade de gravação que a 2017.11.29 (herdada).

Para determinar qual versão você está usando, consulte Determinar a versão de uma tabela global. Para atualizar tabelas globais existentes da versão 2017.11.29 (herdada) para a versão 2019.11.21 (atual), consulte Versões de tabelas globais do DynamoDB.

Ao usar as tabelas globais do Amazon DynamoDB, você pode replicar os dados da tabela nas regiões da AWS. É importante que as tabelas de réplica e índices secundários na tabela global tenham configurações idênticas de capacidade de gravação para garantir a replicação apropriada dos dados.

Tópicos

- Versão de tabelas globais
- Requisitos para adicionar uma nova tabela-réplica
- Práticas recomendadas e requisitos de gerenciamento de capacidade

Versão de tabelas globais

Há duas versões disponíveis das tabelas globais do DynamoDB: Global Tables versão 2019.11.21 (atual) e Global Tables versão 2017.11.29 (herdada). Os clientes devem usar Global Tables versão 2019.11.21 (atual) sempre que possível, pois ela oferece maior flexibilidade e eficiência, além de consumir menor capacidade de gravação que a 2017.11.29 (herdada).

Para determinar qual versão você está usando, consulte Determinar a versão de uma tabela global. Para atualizar tabelas globais da versão 2017.11.29 (herdada) para a versão 2019.11.21 (atual), consulte Atualizar as tabelas globais.

Requisitos para adicionar uma nova tabela-réplica

Se você quiser adicionar uma nova tabela-réplica a uma tabela global, todas estas condições precisarão ser verdadeiras:

- A tabela precisa ter a mesma chave de partição que todas as outras réplicas.
- A tabela deve ter as mesmas configurações de gerenciamento de capacidade de gravação especificadas.
- A tabela precisa ter o mesmo nome que todas as outras réplicas.
- A tabela precisa estar com o DynamoDB Streams ativado e o fluxo precisa conter as imagens novas e antigas do item.

- Nenhuma das tabelas-réplica novas ou existentes na tabela global pode conter dados.

Se os índices secundários globais forem especificados, as seguintes condições também deverão ser atendidas:

- Os índices secundários globais devem ter o mesmo nome.
- Os índices secundários globais devem ter a mesma chave de partição e chave de classificação (se presente).

> **Important**
>
> As configurações de capacidade de gravação devem ser definidas de maneira consistente em todas as tabelas-réplica das tabelas globais e corresponder aos índices secundários. Para atualizar as configurações de capacidade de gravação para a tabela global, é altamente recomendável usar o console do DynamoDB ou a operação `UpdateGlobalTableSettings` da API. `UpdateGlobalTableSettings` aplica automaticamente alterações nas configurações de capacidade de gravação para todas as tabelas-réplica e índices secundários correspondentes em uma tabela global. Se você usar as operações `UpdateTable`, `RegisterScalableTarget` ou `PutScalingPolicy`, deverá aplicar a alteração em cada tabela-réplica e índice secundário correspondente individualmente. Para obter mais informações, consulte UpdateGlobalTableSettings na Referência da API do Amazon DynamoDB.
>
> Recomendamos enfaticamente habilitar o Auto Scaling para gerenciar as configurações de capacidade de gravação provisionadas. Se preferir gerenciar as configurações de capacidade de gravação manualmente, você deverá provisionar unidades de capacidade de gravação replicadas iguais para todas as suas tabelas-réplica. Provisione também unidades de capacidade de gravação replicadas iguais para fazer a correspondência dos índices secundários na tabela global.
>
> Você também precisa ter as permissões adequadas do AWS Identity and Access Management (IAM). Para obter mais informações, consulte Usar o IAM com tabelas globais.

Práticas recomendadas e requisitos de gerenciamento de capacidade

Considere o seguinte ao gerenciar configurações de capacidade de tabelas-réplica no DynamoDB.

Usar o Auto Scaling do DynamoDB

Usar o Auto Scaling do DynamoDB é a maneira recomendada de gerenciar as configurações de capacidade de throughput para tabelas-réplica que usam o modo provisionado. O Auto Scaling do DynamoDB ajusta automaticamente as unidades de capacidade de leitura (RCUs) e as unidades de capacidade de gravação (WCUs) em cada tabela-réplica, com base na workload atual da aplicação. Para obter mais informações, consulte Gerenciar a capacidade de throughput automaticamente com o ajuste de escala automático do DynamoDB.

Se você criar suas tabelas-réplica usando o AWS Management Console, o Auto Scaling ficará ativado por padrão em cada tabela-réplica com as configurações padrão para o gerenciamento das unidades de capacidade de leitura e de gravação.

As alterações às configurações de Auto Scaling para uma tabela-réplica ou um índice secundário feitas por meio do console do DynamoDB ou pela utilização da chamada `UpdateGlobalTableSettings` são aplicadas a todas as tabelas-réplica e índices secundários correspondentes na tabela global automaticamente. Essas alterações substituem todas as configurações de Auto Scaling existentes. Isso garante que as configurações de capacidade de gravação provisionadas fiquem consistentes nas tabelas-réplica e nos índices secundários na tabela global. Se usar chamadas `UpdateTable`, `RegisterScalableTarget` ou `PutScalingPolicy`, você deverá aplicar a alteração a cada tabela-réplica e índice secundário correspondente individualmente.

> **Note**
>
> Se o Auto Scaling não satisfizer as alterações de capacidade de seu aplicativo (workload imprevisível) ou se você não quiser definir suas configurações (configurações de destino para limite mínimo, máximo ou de utilização), você poderá usar o modo sob demanda para gerenciar a capacidade das tabelas globais. Para obter mais informações, consulte Modo sob demanda.
>
> Se você habilitar modo sob demanda em uma tabela global, seu consumo de unidades de solicitação de gravação replicadas (rWCUs) será consistente com como rWCUs são provisionados. Por exemplo, se você realizar 10 gravações em uma tabela local que é replicada em duas regiões adicionais, você consumirá 60 unidades de solicitações de gravação (10 + 10 + 10 = 30; 30 x 2 = 60). As 60 unidades de solicitação de gravação consumidas incluem a gravação extra consumida pelas tabelas globais versão 2017.11.29 (herdada) para atualizar os atributos `aws:rep:deleting`, `aws:rep:updatetime` e `aws:rep:updateregion`.

Gerenciar capacidade manualmente

Se você decidir não usar o Auto Scaling do DynamoDB, deverá definir manualmente as configurações de capacidade de leitura e de gravação em cada tabela-réplica e no índice secundário.

As unidades de capacidade de gravação replicadas (rWCUs - Replicated write capacity units) provisionadas em cada tabela-réplica devem ser definidas para o número total de rWCUs necessárias para as gravações de aplicativo em todas as regiões multiplicado por dois. Isso acomoda as gravações do aplicativo que ocorrem na região local e as gravações replicadas do aplicativo vindas de outras regiões. Por exemplo, suponha que você espera 5 gravações por segundo em sua tabela-réplica em Ohio e 5 gravações por segundo em sua tabela-réplica no Norte da Virginia. Nesse caso, você deve provisionar 20 rWCUs para cada tabela-réplica (5 + 5 = 10; 10 x 2 = 20).

Para atualizar as configurações de capacidade de gravação para a tabela global, é altamente recomendável usar o console do DynamoDB ou a operação `UpdateGlobalTableSettings` da API. `UpdateGlobalTableSettings` aplica automaticamente alterações nas configurações de capacidade de gravação para todas as tabelas-réplica e índices secundários correspondentes em uma tabela global. Se você usar as operações `UpdateTable`, `RegisterScalableTarget` ou `PutScalingPolicy`, deverá aplicar a alteração em cada tabela-réplica e índice secundário correspondente individualmente. Para obter mais informações, consulte a Referência da API do Amazon DynamoDB.

> **Note**
>
> Para atualizar as configurações (`UpdateGlobalTableSettings`) para uma tabela global no DynamoDB, você deve ter as permissões `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget`. Para obter mais informações, consulte Usar o IAM com tabelas globais.

Criar uma tabela global

> **Important**
>
> Esta documentação, destinada à versão 2017.11.29 (herdada) de tabelas globais, deve ser evitada para novas tabelas globais. Os clientes devem usar Global Tables versão

2019.11.21 (atual) sempre que possível, pois ela oferece maior flexibilidade e eficiência, além de consumir menor capacidade de gravação que a 2017.11.29 (herdada).
Para determinar qual versão você está usando, consulte Determinar a versão de uma tabela global. Para atualizar tabelas globais existentes da versão 2017.11.29 (herdada) para a versão 2019.11.21 (atual), consulte Versões de tabelas globais do DynamoDB.

Esta seção descreve como criar uma tabela global usando o console do Amazon DynamoDB ou a AWS Command Line Interface (AWS CLI).

Tópicos

- Criar uma tabela global (console)
- Criar uma tabela global (AWS CLI)

Criar uma tabela global (console)

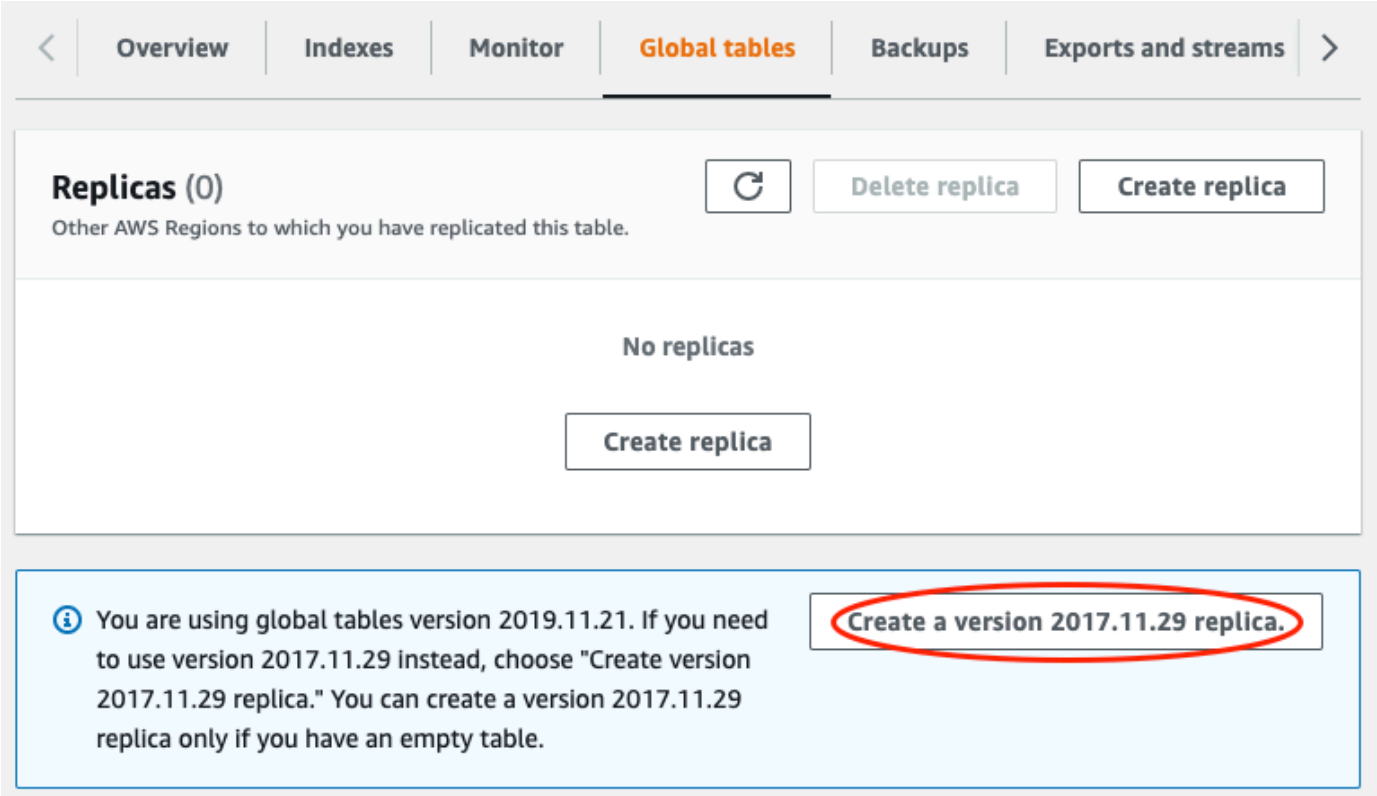
Siga estas etapas para criar uma tabela global usando o console. O exemplo a seguir cria uma tabela global com tabelas-réplica nos Estados Unidos e na Europa.

1. Abra o console do DynamoDB em https://console.aws.amazon.com/dynamodb/. Para este exemplo, escolha a região us-east-2 (Leste dos EUA (Ohio)).
2. No painel de navegação, no lado esquerdo do console, selecione Tables (Tabelas).
3. Selecione Create Table (Criar tabela).

   Em Table name (Nome da tabela), insira **`Music`**.

   Em Primary key (Chave primária) insira **`Artist`**. Escolha Add sort key (Adicionar chave de classificação) e insira **`SongTitle`** (**`Artist`** e **`SongTitle`** devem ser strings).

   Para criar a tabela, selecione Create (Criar). Essa tabela serve como a primeira tabela-réplica em uma nova tabela global. Ela é o protótipo das outras tabelas-réplica que serão adicionadas posteriormente.
4. Selecione a guia Tabelas globais, depois escolha Criar uma réplica da versão 2017.11.29 (herdada).



5. Do menu suspenso Available replication Regions (Regiões de replicação disponíveis), escolha US West (Oregon) (Oeste dos EUA – Oregon).

   O console faz uma verificação para garantir que não exista uma tabela com o mesmo nome na região selecionada. Se existir uma tabela com o mesmo nome, será necessário excluir a tabela existente para criar outra tabela-réplica nessa região.

6. Escolha Create replica (Criar réplica). Isso inicia o processo de criação da tabela na região Oeste dos EUA (Oregon).

   A guia Global Table (Tabela global) da tabela selecionada (e de qualquer outra tabela-réplica) mostra que a tabela foi replicada em várias regiões.

7. Agora, adicione outra região para que a tabela global seja replicada e sincronizada nos Estados Unidos e na Europa. Para fazer isso, repita a Etapa 5, mas desta vez especifique Europe (Frankfurt) (Europa – Frankfurt), em vez de US West (Oregon) (Oeste dos EUA – Oregon).

8. Você ainda deve estar usando o AWS Management Console na região US East (Ohio) (Leste dos EUA – Ohio). No menu de navegação à esquerda, selecione Items (Itens), selecione a tabela Music (Música) e, em seguida, escolha Create Item (Criar item).

   a. Em Artist (Artista), insira **`item_1`**.

b. Em SongTitle (Nome da música), insira **`Song Value 1`**.

c. Para gravar o item, selecione Create Item (Criar item).

9. Pouco tempo depois o item será replicado em todas as três regiões da tabela global. Para verificar, no console, no seletor de regiões no canto superior direito do console, selecione Europe (Frankfurt) (Europa (Frankfurt)). A tabela `Music` na região Europa (Frankfurt) deve conter o novo item.
10. Repita a etapa 9 e escolha US West (Oregon) (Oeste dos EUA [Oregon]) para verificar a replicação nessa região.

Criar uma tabela global (AWS CLI)

Siga estas etapas para criar uma tabela global `Music` usando a AWS CLI. O exemplo a seguir cria uma tabela global com tabelas-réplica nos Estados Unidos e na Europa.

1. Crie uma nova tabela (`Music`) na região Leste dos EUA (Ohio), com o DynamoDB Streams habilitado (`NEW_AND_OLD_IMAGES`).

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. Crie uma tabela `Music` idêntica na região Leste dos EUA (Norte da Virgínia).

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
```

```
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-1
```

3. Crie uma tabela global (`Music`) composta de tabelas-réplica nas regiões `us-east-2` e `us-east-1`.

```
aws dynamodb create-global-table \
    --global-table-name Music \
    --replication-group RegionName=us-east-2 RegionName=us-east-1 \
    --region us-east-2
```

> **Note**
>
> O nome da tabela global (`Music`) deve corresponder ao nome de cada uma das tabelas-réplica (`Music`). Para obter mais informações, consulte Melhores práticas para tabelas globais.

4. Crie outra tabela na região Europa (Irlanda), com as mesmas configurações daquela que você criou na etapa 1 e na etapa 2:

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region eu-west-1
```

Após terminar essa etapa, adicione a nova tabela à tabela global `Music`.

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
```

```
    --region us-east-2
```

5. Para verificar se a replicação está funcionando, adicione um novo item à tabela `Music` na região Leste dos EUA (Ohio).

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. Aguarde alguns segundos e verifique se o item foi replicado com êxito nas regiões Leste dos EUA (Norte da Virgínia) e Europa (Irlanda).

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1
```

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Monitorar tabelas globais

> **Important**
>
> Esta documentação, destinada à versão 2017.11.29 (herdada) de tabelas globais, deve ser evitada para novas tabelas globais. Os clientes devem usar Global Tables versão 2019.11.21 (atual) sempre que possível, pois ela oferece maior flexibilidade e eficiência, além de consumir menor capacidade de gravação que a 2017.11.29 (herdada).
> Para determinar qual versão você está usando, consulte Determinar a versão de uma tabela global. Para atualizar tabelas globais existentes da versão 2017.11.29 (herdada) para a versão 2019.11.21 (atual), consulte Versões de tabelas globais do DynamoDB.

Você pode usar o Amazon CloudWatch para monitorar o comportamento e a performance de uma tabela global. O Amazon DynamoDB publica as métricas `ReplicationLatency` e `PendingReplicationCount` para cada réplica na tabela global.

- **`ReplicationLatency`**: o tempo decorrido entre quando um item atualizado aparece em um fluxo do DynamoDB para uma tabela-réplica e quando esse item aparece em outra réplica na tabela global. `ReplicationLatency` é expresso em milissegundos e é emitido para cada par de regiões de origem e destino.

  Durante o funcionamento normal, `ReplicationLatency` deve ser constante. Um valor elevado para `ReplicationLatency` pode indicar que as atualizações de uma réplica não se propagaram para outras tabelas-réplica em tempo hábil. Com o tempo, isso pode fazer com que outras tabelas-réplica fiquem para trás já que elas não recebem atualizações de forma consistente. Nesse caso, você deve verificar se as unidades de capacidade de leitura (RCUs) e as unidades de capacidade de gravação (WCUs) são idênticas em cada uma das tabelas-réplica. Além disso, ao escolher as configurações da WCU, siga as recomendações em Versão de tabelas globais.

  A `ReplicationLatency` pode aumentar se uma região da AWS se tornar degradada e houver uma tabela-réplica nessa região. Nesse caso, você pode redirecionar temporariamente as atividades de leitura e gravação da aplicação para outra região da AWS.

- **`PendingReplicationCount`**: o número de atualizações de itens que foram gravadas em uma tabela-réplica, mas que ainda não foram gravadas em outra réplica na tabela global. `PendingReplicationCount` é expressa em número de itens e emitida para cada par de regiões de origem e destino.

  Durante a operação normal, `PendingReplicationCount` deve ser muito baixa. Se `PendingReplicationCount` aumentar por períodos prolongados, você deverá investigar se as configurações de capacidade de gravação provisionadas das tabelas-réplica são suficientes para a sua workload atual.

  A `PendingReplicationCount` pode aumentar se uma região da AWS se tornar degradada e houver uma tabela-réplica nessa região. Nesse caso, você pode redirecionar temporariamente as atividades de leitura e gravação da aplicação para outra região da AWS.

Para obter mais informações, consulte Métricas e dimensões do DynamoDB.

Usar o IAM com tabelas globais

> **Important**
>
> Esta documentação, destinada à versão 2017.11.29 (herdada) de tabelas globais, deve ser evitada para novas tabelas globais. Os clientes devem usar Global Tables versão 2019.11.21 (atual) sempre que possível, pois ela oferece maior flexibilidade e eficiência, além de consumir menor capacidade de gravação que a 2017.11.29 (herdada).
> Para determinar qual versão você está usando, consulte Determinar a versão de uma tabela global. Para atualizar tabelas globais existentes da versão 2017.11.29 (herdada) para a versão 2019.11.21 (atual), consulte Versões de tabelas globais do DynamoDB.

Na primeira vez que você cria uma tabela global, o Amazon DynamoDB cria automaticamente para você uma função vinculada ao serviço do AWS Identity and Access Management (IAM). Essa função é chamada e permite que o DynamoDB gerencie a replicação em tabelas globais entre regiões em seu nome. Não exclua essa função vinculada ao serviço. Se fizer isso, todas as suas tabelas globais não funcionarão mais.

Para obter mais informações sobre funções vinculadas a serviços, consulte Usando funções vinculadas a serviços no Guia do Usuário do IAM.

Para criar e manter tabelas globais no DynamoDB, é preciso ter permissão `dynamodb:CreateGlobalTable` para acessar cada um dos itens a seguir:

- A tabela-réplica que você deseja adicionar.
- Toda réplica existente que já faça parte da tabela global.
- A própria tabela global.

Para atualizar as configurações (`UpdateGlobalTableSettings`) para uma tabela global no DynamoDB, você deve ter as permissões `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget`.

As permissões `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget` são necessárias ao atualizar uma política de

dimensionamento existente. Isso ocorre para que o serviço de tabelas globais possa remover a política de dimensionamento antiga antes de anexar a nova política à tabela ou ao índice secundário.

Se você usar uma política do IAM para gerenciar o acesso a uma tabela-réplica, deverá aplicar uma política idêntica a todas as outras réplicas dentro da tabela global. Esse procedimento ajuda você a manter um modelo de permissões consistente em todas as tabelas-réplica.

Ao usar políticas idênticas do IAM em todas as réplicas em uma tabela global, você também pode evitar a concessão não intencional de acesso de leitura e gravação a seus dados na tabela global. Por exemplo, considere um usuário que tem acesso a uma única réplica em uma tabela global. Se esse usuário puder gravar nessa réplica, o DynamoDB propagará a gravação para todas as outras tabelas-réplica. Na verdade, o usuário poderá gravar (indiretamente) em todas as outras réplicas na tabela global. Essa situação pode ser evitada usando políticas consistentes do IAM em todas as tabelas-réplica.

Exemplo: permitir a ação CreateGlobalTable

Para adicionar uma réplica a uma tabela global, você precisa ter a permissão `dynamodb:CreateGlobalTable` para a tabela global e para cada uma de suas tabelas-réplica.

A política do IAM a seguir concede permissões para a ação `CreateGlobalTable` em todas as tabelas.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

Exemplo: permitir as ações UpdateGlobalTable, DescribeLimits, application-autoscaling:DeleteScalingPolicy e application-autoscaling:DeregisterScalableTarget

Para atualizar as configurações (`UpdateGlobalTableSettings`) para uma tabela global no DynamoDB, você deve ter as permissões `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget`.

A política do IAM a seguir concede permissões para a ação `UpdateGlobalTableSettings` em todas as tabelas.

JSON

```
{
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

Exemplo: permitir a ação CreateGlobalTable para um nome de tabela global específico com réplicas permitidas somente em determinadas regiões

A política do IAM a seguir concede permissões para permitir que a ação `CreateGlobalTable` crie uma tabela global chamada `Customers` com réplicas em duas regiões.

JSON

```
{
```

```
    "Version": "2012-10-17"       ,
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:CreateGlobalTable",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Customers",
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers"
            ]
        }
    ]
}
```

Versão anterior da API de baixo nível do DynamoDB (5/12/2011)

Esta seção documenta as operações disponíveis na versão anterior da API de baixo nível do DynamoDB (2011-12-05). Essa versão da API de baixo nível é mantida para compatibilidade com versões anteriores de aplicativos existentes.

Novos aplicativos devem usar a versão atual da API (2012-08-10). Para obter mais informações, consulte Referência de API do DynamoDB.

> **Note**
>
> Recomendamos migrar suas aplicações para a versão mais recente da API (2012-08-10), pois a compatibilização com a versão anterior da API não será oferecida para novos recursos do DynamoDB.

Tópicos

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables

- GetItem
- ListTables
- PutItem
- Consulta
- Verificar
- UpdateItem
- UpdateTable

BatchGetItem

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

A operação `BatchGetItem` retorna os atributos para vários itens de várias tabelas usando suas chaves primárias. O número máximo de itens que podem ser recuperados para uma única operação é 100. Além disso, o número de itens recuperados é restrito por um limite de tamanho de 1 MB. Se o limite de tamanho da resposta for excedido ou se um resultado parcial for retornado porque o throughput provisionado da tabela foi excedida, ou devido a uma falha de processamento interno, o DynamoDB retornará um valor de `UnprocessedKeys` para que você possa tentar novamente a operação começando com o próximo item a ser obtido. O DynamoDB ajusta automaticamente o número de itens retornados por página para impor esse limite. Por exemplo, mesmo que você solicite o retorno de 100 itens, mas cada item tem um tamanho de 50 KB, o sistema retornará 20 itens e um valor de `UnprocessedKeys` apropriado para que você possa obter a próxima página de resultados. Se você desejar, o aplicativo pode incluir sua própria lógica para reunir as páginas de resultados em um único conjunto.

Se nenhum item pôde ser processado devido a um throughput provisionado insuficiente em cada uma das tabelas envolvidas na solicitação, o DynamoDB retornará um erro `ProvisionedThroughputExceededException`.

> **Note**
>
> Por padrão, o `BatchGetItem` realiza leituras finais consistentes em cada tabela da solicitação. Você pode definir o parâmetro `ConsistentRead` como `true` para cada tabela se quiser leituras consistentes.
> `BatchGetItem` busca itens em paralelo para minimizar latências de resposta.
> Ao projetar sua aplicação, tenha em mente que o DynamoDB não garante como os atributos são ordenados na resposta retornada. Inclua os valores de chaves primárias no `AttributesToGet` dos itens na sua solicitação para ajudar a analisar a resposta por item.
> Se os itens solicitados não existirem, nada será retornado na resposta para esses itens.
> Solicitações de itens inexistentes consomem as unidades mínimas de capacidade de leitura de acordo com o tipo de leitura. Para ter mais informações, consulte Tamanhos e formatos de item do DynamoDB.

Solicitações

Sintaxe

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
    {"Table1":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
    "Table2":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}}],
        "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
        }
    }
```

```
}
```

Nome	Descrição	Obrigatório
`RequestItems`	Um contêiner do nome da tabela e de itens correspon dentes a serem obtidos por chave primária. Ao solicitar itens, cada nome de tabela pode ser invocado apenas uma vez por operação. Tipo: string Padrão: nenhum	Sim
`Table`	O nome da tabela que contém os itens a serem obtidos. A entrada é simplesmente uma string especificando uma tabela existente, sem rótulo. Tipo: string Padrão: nenhum	Sim
`Table:Keys`	Os valores de chave primária que definem os itens na tabela especificada. Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: Chaves	Sim
`Table:AttributesToGet`	Matriz de nomes de atributos dentro da tabela especificada. Se os nomes de atributos não forem especificados, todos os	Não

Nome	Descrição	Obrigatório
	atributos serão retornados. Se alguns atributos não forem encontrados, eles não serão exibidos no resultado. Tipo: matriz	
`Table:ConsistentRead`	Se definido como `true`, uma leitura consistente será emitida; caso contrário, uma leitura eventualmente consistente será usada. Tipo: booliano	Não

Respostas

Sintaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 855

{"Responses":
    {"Table1":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue",
 "AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1},
    "Table2":
        {"Items":
```

```
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue","AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":
        {"Table3":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]}
        }
}
```

Nome	Descrição
`Responses`	Nomes de tabelas e os respectivos atributos de itens das tabelas. Tipo: mapa
`Table`	O nome da tabela que contém os itens. A entrada é simplesmente uma string especific ando a tabela, sem rótulo. Tipo: string
`Items`	Contêiner para nomes de atributos e valores que atendem aos parâmetros de operação. Tipo: mapa de nomes de atributo e seus tipos de dados e valores.

Nome	Descrição
`ConsumedCapacityUnits`	O número de unidades de capacidade de leitura consumidas, para cada tabela. Esse valor mostra o número utilizado no throughput provisionado. As solicitações de itens não existentes consomem o mínimo de unidades de capacidade de leitura, dependendo do tipo de leitura. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB. Tipo: número
`UnprocessedKeys`	Contém uma matriz de tabelas e suas respectivas chaves que não foram processadas com a resposta atual, possivelmente devido a um limite atingido no tamanho da resposta. O valor `UnprocessedKeys` está no mesmo formato que um parâmetro `RequestItems` (portanto, ele pode ser fornecido diretamente a uma operação `BatchGetItem` posterior). Para obter mais informações, consulte o parâmetro `RequestItems` acima. Tipo: matriz
`UnprocessedKeys` : `Table`: `Keys`	Os valores de atributos de chave primária que definem os itens e os atributos associados a esses itens. Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: matriz de pares de nome-valor de atributo.

Nome	Descrição
`UnprocessedKeys` : `Table`: `AttributesToGet`	Nomes de atributos na tabela especificada. Se os nomes de atributos não forem especificados, todos os atributos serão retornados. Se alguns atributos não forem encontrados, eles não serão exibidos no resultado. Tipo: Matriz de nomes de atributos.
`UnprocessedKeys` : `Table`: `ConsistentRead`	Se definido como `true`, uma leitura consistente será usada para a tabela especificada; caso contrário, uma leitura eventualmente consistente será usada. Tipo: booliano.

Erros especiais

Erro	Descrição
`ProvisionedThroughputExceededException`	O throughput provisionado máximo permitido foi excedido.

Exemplos

Os exemplos a seguir mostram uma solicitação HTTP POST e uma resposta usando a operação BatchGetItem. Para obter exemplos sobre o uso do AWS SDK, consulte Trabalhar com itens e atributos no DynamoDB.

Exemplo de solicitação

O exemplo a seguir solicita atributos de duas tabelas diferentes.

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
    {"comp1":
        {"Keys":
            [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
            {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
            {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
        "AttributesToGet":["user","status"]},
    "comp2":
        {"Keys":
            [{"HashKeyElement":{"S":"Julie"}},{"HashKeyElement":{"S":"Mingus"}}],
        "AttributesToGet":["user","friends"]}
    }
}
```

Exemplo de resposta

O exemplo a seguir é a resposta.

```
HTTP/1.1 200 OK
x-amzn-RequestId: GTPQVRM4VJS792J1UFJTKUBVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 373
Date: Fri, 02 Sep 2011 23:07:39 GMT

{"Responses":
    {"comp1":
        {"Items":
            [{"status":{"S":"online"},"user":{"S":"Casey"}},
            {"status":{"S":"working"},"user":{"S":"Riley"}},
            {"status":{"S":"running"},"user":{"S":"Dave"}}],
        "ConsumedCapacityUnits":1.5},
    "comp2":
        {"Items":
            [{"friends":{"SS":["Elisabeth", "Peter"]},"user":{"S":"Mingus"}},
            {"friends":{"SS":["Dave", "Peter"]},"user":{"S":"Julie"}}],
        "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":{}
}
```

BatchWriteItem

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Essa operação permite que você insira ou exclua vários itens de várias tabelas em uma única chamada.

Para fazer upload de um item, você pode usar `PutItem` e, para excluir um item, você pode usar `DeleteItem`. No entanto, quando você deseja enviar ou excluir grandes quantidades de dados, por exemplo, carregar grandes quantidades de dados do Amazon EMR (Amazon EMR) ou migrar dados de outro banco de dados para o DynamoDB, `BatchWriteItem` oferece uma alternativa eficiente.

Se você usa linguagens como o Java, pode usar threads para carregar itens em paralelo. Isso adiciona complexidade à sua aplicação para manipular os threads. Outras linguagens não oferecem suporte para threads. Por exemplo, se você estiver usando o PHP, deverá carregar ou excluir itens um de cada vez. Em ambas as situações, `BatchWriteItem` fornece uma alternativa na qual as operações de inserção e exclusão especificadas são processadas em paralelo, proporcionando a você o poder da abordagem de pools de threads sem a necessidade de introduzir complexidade no seu aplicativo.

Observe que cada operação individual de inserção e exclusão especificada em uma operação `BatchWriteItem` tem o mesmo custo em termos de unidades de capacidade consumidas. No entanto, como `BatchWriteItem` realiza as operações especificadas em paralelo, a latência é menor. Operações de exclusão em itens inexistentes consomem 1 unidade de capacidade de gravação. Para obter mais informações sobre unidades de capacidade consumidas, consulte Trabalhar com tabelas e dados no DynamoDB.

Ao usar `BatchWriteItem`, observe as seguintes limitações:

- Operações máximas em uma única solicitação: é possível especificar um total de até 25 operações de inserção ou exclusão. No entanto, o tamanho total da solicitação não pode exceder 1 MB (a carga útil HTTP).
- A operação `BatchWriteItem` só pode ser usada para inserir e excluir itens. Ela não pode ser usada para atualizar itens existentes.
- Não é uma operação atômica: operações individuais especificadas em uma `BatchWriteItem` são atômicas. No entanto, `BatchWriteItem` como um todo é uma operação de "melhor esforço", e não uma operação atômica. Ou seja, em uma solicitação `BatchWriteItem`, algumas operações podem ser bem-sucedidas, enquanto outras podem falhar. As operações com falha são retornadas em um campo `UnprocessedItems` na resposta. Algumas dessas falhas podem ocorrer porque você excedeu o throughput provisionado configurado para a tabela ou devido a uma falha transitória, como um erro de rede. Você pode investigar e, opcionalmente, reenviar as solicitações. Em geral, você chama `BatchWriteItem` em um loop e em cada verificação de iteração em busca de itens não processados e envia uma nova solicitação `BatchWriteItem` com esses itens não processados.
- Não retorna itens: `BatchWriteItem` foi projetada para carregar grandes quantidades de dados de forma eficiente. Ela não oferece algumas das sofisticações oferecidas por `PutItem` e `DeleteItem`. Por exemplo, `DeleteItem` oferece suporte ao campo `ReturnValues` no corpo da solicitação para solicitar o item excluído na resposta. A operação `BatchWriteItem` não retorna itens na resposta.
- Ao contrário de `PutItem` e `DeleteItem`, `BatchWriteItem` não permite que você especifique condições em solicitações de gravação individuais na operação.
- Valores de atributos não devem ser nulos, atributos do tipo string e binários devem ter comprimentos maiores que zero, e atributos de tipo definido não podem permanecer vazios. Solicitações que têm valores vazios serão rejeitadas com uma `ValidationException`.

O DynamoDB rejeitará a operação de gravação em lote inteira em qualquer uma das seguintes situações:

- Se uma ou mais tabelas especificadas na solicitação `BatchWriteItem` não existir.
- Se atributos de chaves primárias especificados em um item na solicitação não corresponderem ao esquema de chave primária da tabela correspondente.
- Se você tentar realizar várias operações no mesmo item na mesma solicitação `BatchWriteItem`. Por exemplo, não é possível inserir e excluir o mesmo item na mesma solicitação `BatchWriteItem`.

- Se o tamanho total da solicitação exceder o limite de 1 MB (a carga útil HTTP).
- Se qualquer item individual em um lote exceder o limite de tamanho de item de 64 KB.

Solicitações

Sintaxe

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems" :  RequestItems
}

RequestItems
{
     "TableName1" :  [ Request, Request, ... ],
     "TableName2" :  [ Request, Request, ... ],
     ...
}

Request ::=
  PutRequest | DeleteRequest

PutRequest ::=
{
  "PutRequest" : {
     "Item" : {
        "Attribute-Name1" : Attribute-Value,
        "Attribute-Name2" : Attribute-Value,
         ...
     }
  }
}

DeleteRequest ::=
{
   "DeleteRequest" : {
      "Key" : PrimaryKey-Value
```

```
   }
}


PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK

HashTypePK ::=
{
   "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
   "HashKeyElement" : Attribute-Value,
   "RangeKeyElement" : Attribute-Value,
}

Attribute-Value ::= String | Numeric| Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
   "N": "Number"
}

String ::=
{
   "S": "String"
}

Binary ::=
{
    "B": "Base64 encoded binary data"
}

StringSet ::=
{
   "SS": [ "String1", "String2", ... ]
}

NumberSet ::=
{
   "NS": [ "Number1", "Number2", ... ]
}
```

```
BinarySet ::=
{
   "BS": [ "Binary1", "Binary2", ... ]
}
```

No corpo da solicitação, o objeto JSON `RequestItems` descreve as operações que você deseja realizar. As operações são agrupadas por tabelas. É possível usar `BatchWriteItem` para atualizar ou excluir vários itens em várias tabelas. Para cada solicitação de gravação específica, você deve identificar o tipo de solicitação (`PutItem`, `DeleteItem`) seguido de informações detalhadas sobre a operação.

- Para uma `PutRequest`, você fornece o item, ou seja, uma lista de atributos e seus valores.
- Para uma `DeleteRequest`, você fornece o nome e o valor da chave primária.

Respostas

Sintaxe

Veja a seguir a sintaxe do corpo JSON retornado na resposta.

```
{
  "Responses" :         ConsumedCapacityUnitsByTable
  "UnprocessedItems" :  RequestItems
}

ConsumedCapacityUnitsByTable
{
    "TableName1" : { "ConsumedCapacityUnits", : NumericValue },
    "TableName2" : { "ConsumedCapacityUnits", : NumericValue },
     ...
}

RequestItems
This syntax is identical to the one described in the JSON syntax in the request.
```

Erros especiais

Não há erros específicos para esta operação.

Exemplos

O exemplo a seguir mostra uma solicitação HTTP POST e a resposta de uma operação `BatchWriteItem`. A solicitação especifica as seguintes operações nas tabelas Reply e Thread:

- Inserir um item e excluí-lo da tabela Reply
- Inserir um item na tabela Thread

Para obter exemplos sobre o uso do AWS SDK, consulte Trabalhar com itens e atributos no DynamoDB.

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems":{
    "Reply":[
      {
        "PutRequest":{
          "Item":{
            "ReplyDateTime":{
              "S":"2012-04-03T11:04:47.034Z"
            },
            "Id":{
              "S":"DynamoDB#DynamoDB Thread 5"
            }
          }
        }
      },
      {
        "DeleteRequest":{
          "Key":{
            "HashKeyElement":{
              "S":"DynamoDB#DynamoDB Thread 4"
            },
            "RangeKeyElement":{
              "S":"oops - accidental row"
```

```
            }
          }
        }
      }
    ],
    "Thread":[
      {
        "PutRequest":{
          "Item":{
            "ForumName":{
              "S":"DynamoDB"
            },
            "Subject":{
              "S":"DynamoDB Thread 5"
            }
          }
        }
      }
    ]
  }
}
```

Exemplo de resposta

A seguinte resposta de exemplo mostra uma operação de inserção bem-sucedida nas tabelas Thread e Reply e uma operação de exclusão com falha na tabela Reply (por motivos como a limitação causada quando você excede o throughput provisionado na tabela). Observe o seguinte na resposta JSON:

- O objeto `Responses` mostra que uma unidade de capacidade foi consumida em ambas as tabelas `Thread` e `Reply` como resultado da operação bem-sucedida em cada uma dessas tabelas.
- O objeto `UnprocessedItems` mostra a operação de exclusão que falhou na tabela `Reply`. Em seguida, você pode emitir uma nova chamada `BatchWriteItem` para solucionar essas solicitações não processadas.

```
HTTP/1.1 200 OK
x-amzn-RequestId: G8M9ANLOE5QA26AEUHJKJE0ASBVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 536
Date: Thu, 05 Apr 2012 18:22:09 GMT
```

```
{
   "Responses":{
      "Thread":{
         "ConsumedCapacityUnits":1.0
      },
      "Reply":{
         "ConsumedCapacityUnits":1.0
      }
   },
   "UnprocessedItems":{
      "Reply":[
         {
            "DeleteRequest":{
               "Key":{
                  "HashKeyElement":{
                     "S":"DynamoDB#DynamoDB Thread 4"
                  },
                  "RangeKeyElement":{
                     "S":"oops - accidental row"
                  }
               }
            }
         }
      ]
   }
}
```

CreateTable

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

A operação `CreateTable` adiciona uma nova tabela à sua conta.

O nome da tabela deve ser exclusivo entre aqueles associados à conta da AWS que emite a solicitação e à região da AWS que recebe a solicitação (como dynamodb.us-west-2.amazonaws.com). Cada endpoint do DynamoDB é totalmente independente. Por exemplo, se você tiver duas tabelas chamadas “MyTable”, uma em dynamodb.us-west-2.amazonaws.com e outra em dynamodb.us-west-1.amazonaws.com, elas serão completamente independentes e não compartilharão nenhum dado.

A operação `CreateTable` desencadeia um fluxo de trabalho assíncrono para começar a criação da tabela. O DynamoDB retorna imediatamente o estado da tabela (`CREATING`) até que ela atinja o estado `ACTIVE`. Quando a tabela estiver no estado `ACTIVE`, você poderá realizar as operações de plano de dados.

Use a operação DescribeTables para verificar o status da tabela.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela a ser criada. Os caracteres permitidos são a-z, A-Z, 0-9, '_' (sublinha do), '-' (traço) e '.' (ponto). Os	Sim

Nome	Descrição	Obrigatório
	nomes podem ter de 3 a 255 caracteres. Tipo: string	
`KeySchema`	A estrutura da chave primária (simples ou composta) da tabela. O par de nome-valor de `HashKeyElement` é obrigatório, e um par de nome-valor de `RangeKeyElement` é opcional (obrigatório apenas para chaves primárias compostas). Para obter mais informações sobre chaves primárias, consulte Chave primária. Nomes de elementos de chave primária podem ter entre 1 e 255 caracteres de comprimento, sem restrições de caracteres. Os valores possíveis para AttributeType são "S" (string), "N" (numérico) ou "B" (binário). Tipo: mapa de `HashKeyElement`, ou `HashKeyElement` e `RangeKeyElement` para uma chave primária composta.	Sim

Nome	Descrição	Obrigatório
`ProvisionedThroughput`	Novo throughput da tabela especificada consistindo em valores para `ReadCapacityUnits` e `WriteCapacityUnits`. Para obter detalhes, consulte Modo de capacidade provisionada do DynamoDB. **Note** Para conhecer os valores máximos/mínimos atuais, consulte Cotas no Amazon DynamoDB. Tipo: matriz	Sim

Nome	Descrição	Obrigatório
`ProvisionedThroughput : ReadCapacityUnits`	Define o número mínimo de `ReadCapacityUnits` consistentes consumidas por segundo para a tabela especificada antes que o DynamoDB balanceie a carga com outras operações. As operações de leitura eventualmente consistente requerem menos esforço que uma operação de leitura consistente, portanto, uma definição de 50 `ReadCapacityUnits` consistentes por segundo oferece 100 `ReadCapacityUnits` eventualmente consistentes por segundo. Tipo: número	Sim
`ProvisionedThroughput : WriteCapacityUnits`	Define o número mínimo de `WriteCapacityUnits` consumidas por segundo para a tabela especificada antes que o DynamoDB balanceie a carga com outras operações. Tipo: número	Sim

Respostas

Sintaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"CREATING"
    }
}
```

Nome	Descrição
`TableDescription`	Um contêiner para as propriedades da tabela.
`CreationDateTime`	Data em que a tabela foi criada, no formato de tempo epoch UNIX. Tipo: número
`KeySchema`	A estrutura da chave primária (simples ou composta) da tabela. O par de nome-valor de `HashKeyElement` é obrigatório, e um par de nome-valor de `RangeKeyElement` é opcional (obrigatório apenas para chaves primárias compostas). Para obter mais informações sobre chaves primárias, consulte Chave primária.

Nome	Descrição
	Tipo: mapa de `HashKeyElement` , ou `HashKeyElement` e `RangeKeyElement` para uma chave primária composta.
`ProvisionedThroughput`	O throughput da tabela especificada consistindo em valores para `ReadCapacityUnits` e `WriteCapacityUnits` . Consulte Modo de capacidade provisionada do DynamoDB. Tipo: matriz
`ProvisionedThroughput:ReadCapacityUnits`	O número mínimo de `ReadCapacityUnits` consumidas por segundo antes que o DynamoDB balanceie a carga com outras operações. Tipo: número
`ProvisionedThroughput:WriteCapacityUnits`	O número mínimo de `ReadCapacityUnits` consumidas por segundo antes que `WriteCapacityUnits` balanceie a carga com outras operações. Tipo: número
`TableName`	O nome da tabela criada. Tipo: string

Nome	Descrição
`TableStatus`	O estado atual da tabela (`CREATING`). Quando a tabela estiver no estado `ACTIVE`, você poderá inserir dados nela. Use a API DescribeTables para verificar o status da tabela. Tipo: string

Erros especiais

Erro	Descrição
`ResourceInUseException`	Tentativa de recriar uma tabela já existente.
`LimitExceededException`	O número de solicitações simultâneas de tabela (número cumulativo de tabelas no estado `CREATING`, `DELETING` ou `UPDATING`) excede o máximo permitido. **Note** Para conhecer os valores máximos/mínimos atuais, consulte Cotas no Amazon DynamoDB. .

Exemplos

O exemplo a seguir cria uma tabela com uma chave primária composta que contém uma string e um número. Para obter exemplos sobre o uso do AWS SDK, consulte Trabalhar com tabelas e dados no DynamoDB.

Exemplo de solicitação

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0


{"TableName":"comp-table",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"comp-table",
    "TableStatus":"CREATING"
    }
}
```

Ações relacionadas

- DescribeTables
- DeleteTable

DeleteItem

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Exclui um item único em uma tabela por meio de uma chave primária Você pode executar uma operação de exclusão condicional que exclui o item, se ele existir, ou se ele tiver um valor de atributo esperado.

> **Note**
>
> Se você especificar `DeleteItem` sem atributos ou valores, todos os atributos do item serão excluídos.
>
> A não ser que você especifique as condições, `DeleteItem` é uma operação idempotente; executá-la várias vezes no mesmo item ou atributo não resultará em uma resposta de erro.
>
> As exclusões condicionais são úteis para excluir somente itens e atributos se condições específicas forem atendidas. Se as condições forem atendidas, o DynamoDB executará a exclusão. Caso contrário, o item não é excluído.
>
> Você pode executar a verificação condicional esperada em um atributo por operação.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
```

```
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
    "ReturnValues":"ALL_OLD"}
}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela que contém o item a ser excluído. Tipo: string	Sim
`Key`	A chave primária que define o item. Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: mapa de `HashKeyElement` para seu valor e `RangeKeyElement` para seu valor.	Sim
`Expected`	Designa um atributo para uma exclusão condicional. O parâmetro `Expected` permite que você forneça um nome de atributo e se o DynamoDB deve ou não verificar se o atributo tem um valor específico antes de excluí-lo. Tipo: mapa de nomes de atributo.	Não
`Expected:AttributeName`	O nome do atributo da operação put condicional.	Não

Nome	Descrição	Obrigatório
	Tipo: string	

Nome	Descrição	Obrigatório

`Expected:AttributeName: ExpectedAttributeValue`	Use esse parâmetro para especificar se o valor já existe ou não para o par de nome-valor do atributo.
	
	A notação JSON a seguir exclui o item se o atributo "Cor" não existir para esse item:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	A notação JSON a seguir verifica se o atributo com o nome "Cor" tem um valor existente "Amarelo" antes de excluir o item:
	
	`"Expected" : {"Color":{"Exists":true},{"Value": {"S":"Yellow"}}}`
	
	Por padrão, se você usar o parâmetro `Expected` e fornecer um `Value`, o DynamoDB presumirá que o atributo existe e tem um valor atual a ser substituído. Portanto, você não precisa especificar `{"Exists":true}` , pois ele está implícito. Você pode reduzir a solicitação para:
		Não

Nome	Descrição	Obrigatório
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Se você especificar `{"Exists":true}` sem um valor de atributo para verificar, o DynamoDB retornará um erro.	
`ReturnValues`	Use este parâmetro se você deseja obter os pares de nome-valor de atributo antes que eles sejam excluídos. Os possíveis valores de parâmetro são `NONE` (padrão) ou `ALL_OLD`. Se `ALL_OLD` for especificado, o conteúdo do item antigo será retornado. Se esse parâmetro não for fornecido ou for `NONE`, nada será retornado. Tipo: string	Não

Respostas

Sintaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
```

```
content-length: 353
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"Attributes":
    {"AttributeName3":{"SS":["AttributeValue3","AttributeValue4","AttributeValue5"]},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName1":{"N":"AttributeValue1"}
    },
"ConsumedCapacityUnits":1
}
```

Nome	Descrição
`Attributes`	Se o parâmetro `ReturnValues` for fornecido como `ALL_OLD` na solicitação, o DynamoDB retornará uma matriz de pares de nome-valor de atributo (basicamente, o item excluído). Caso contrário, a resposta conterá um conjunto vazio. Tipo: matriz de pares de nome-valor de atributo.
`ConsumedCapacityUnits`	O número de unidades de capacidade de gravação consumidas pela operação. Esse valor mostra o número utilizado no throughput provisionado. As solicitações de exclusão em itens não existentes consomem 1 unidade de capacidade de gravação. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB. Tipo: número

Erros especiais

Erro	Descrição
`ConditionalCheckFailedException`	Falha na verificação condicional. Um valor de atributo esperado não foi encontrado.

Exemplos

Exemplo de solicitação

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
    "Key":
        {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
    "Expected":
        {"status":{"Value":{"S":"shopping"}}},
    "ReturnValues":"ALL_OLD"
}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
x-amzn-RequestId: U9809LI6BBFJA5N2R0TB0P017JVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 22:31:23 GMT

{"Attributes":
    {"friends":{"SS":["Dooley","Ben","Daisy"]},
    "status":{"S":"shopping"},
    "time":{"N":"200"},
    "user":{"S":"Mingus"}
    },
"ConsumedCapacityUnits":1
```

```
}
```

Ações relacionadas

- PutItem

DeleteTable

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

A operação `DeleteTable` exclui uma tabela e todos os seus itens. Após uma solicitação `DeleteTable`, a tabela especificada permanece no estado `DELETING` até que o DynamoDB conclua a exclusão. Se a tabela estiver no estado `ACTIVE`, será possível excluí-la. Se uma tabela estiver no estado `CREATING` ou `UPDATING`, o DynamoDB retornará um erro `ResourceInUseException`. Se a tabela especificada não existir, o DynamoDB retornará um `ResourceNotFoundException`. Se a tabela já estiver no estado `DELETING`, nenhum erro será retornado.

> **Note**
>
> O DynamoDB pode continuar a aceitar solicitações de operação no plano de dados, como `GetItem` e `PutItem`, em uma tabela no estado `DELETING` até que a exclusão da tabela seja concluída.

As tabelas devem ser exclusivas entre aquelas associadas à conta da AWS que emite a solicitação e à região da AWS que recebe a solicitação (como dynamodb.us-west-1.amazonaws.com). Cada endpoint do DynamoDB é totalmente independente. Por exemplo, se você tiver duas tabelas chamadas “MyTable”, uma em dynamodb.us-west-2.amazonaws.com e outra em dynamodb.us-

west-1.amazonaws.com, elas serão completamente independentes e não compartilharão nenhum dado. Excluir uma não excluirá a outra.

Use a operação DescribeTables para verificar o status da tabela.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela a ser excluída. Tipo: string	Sim

Respostas

Sintaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Sun, 14 Aug 2011 22:56:22 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":10,"WriteCapacityUnits":10},
```

```
    "TableName":"Table1",
    "TableStatus":"DELETING"
    }
}
```

Nome	Descrição
`TableDescription`	Um contêiner para as propriedades da tabela.
`CreationDateTime`	Data em que a tabela foi criada. Tipo: número
`KeySchema`	A estrutura da chave primária (simples ou composta) da tabela. O par de nome-valor de `HashKeyElement` é obrigatório, e um par de nome-valor de `RangeKeyElement` é opcional (obrigatório apenas para chaves primárias compostas). Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: mapa de `HashKeyElement` , ou `HashKeyElement` e `RangeKeyElement` para uma chave primária composta.
`ProvisionedThroughput`	O throughput da tabela especificada consistindo em valores para `ReadCapacityUnits` e `WriteCapacityUnits` . Consulte Modo de capacidade provisionada do DynamoDB.
`ProvisionedThroughput` : `ReadCapacityUnits`	O número mínimo de `ReadCapacityUnits` consumidas por segundo para a tabela especificada antes que o DynamoDB balanceie a carga com outras operações. Tipo: número

Nome	Descrição
`ProvisionedThroughput:WriteCapacityUnits`	O número mínimo de `WriteCapacityUnits` consumidas por segundo para a tabela especificada antes que o DynamoDB balanceie a carga com outras operações. Tipo: número
`TableName`	O nome da tabela excluída. Tipo: string
`TableStatus`	O estado atual da tabela (`DELETING`). Assim que a tabela é excluída, as solicitações subsequentes para a tabela retornam `resource not found`. Use a operação DescribeTables para verificar o status da tabela. Tipo: string

Erros especiais

Erro	Descrição
`ResourceInUseException`	A tabela está no estado `CREATING` ou `UPDATING` e não pode ser excluída.

Exemplos

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
```

```
content-type: application/x-amz-json-1.0
content-length: 40

{"TableName":"favorite-movies-table"}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 160
Date: Sun, 14 Aug 2011 17:20:03 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"name","AttributeType":"S"}},
    "TableName":"favorite-movies-table",
    "TableStatus":"DELETING"
}
```

Ações relacionadas

- CreateTable
- DescribeTables

DescribeTables

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Retorna informações sobre a tabela, incluindo seu status atual, o esquema de chave primária e quando a tabela foi criada. Os resultados de DescribeTable são finais consistentes. Se você usar

DescribeTable muito cedo no processo de criação de uma tabela, o DynamoDB retornará um `ResourceNotFoundException`. Se você usar DescribeTable muito cedo no processo de atualizar uma tabela, os novos valores talvez não estejam imediatamente disponíveis.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela a ser descrita. Tipo: string	Sim

Respostas

Sintaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
Content-Length: 543


{"Table":
    {"CreationDateTime":1.309988345372E9,
    ItemCount:1,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": Date, "LastDecreaseDateTime":
 Date, "ReadCapacityUnits":10,"WriteCapacityUnits":10},
```

```
    "TableName":"Table1",
    "TableSizeBytes":1,
    "TableStatus":"ACTIVE"
    }
}
```

Nome	Descrição
`Table`	O contêiner da tabela que está sendo descrita. Tipo: string
`CreationDateTime`	Data em que a tabela foi criada, no formato de tempo epoch UNIX.
`ItemCount`	Número de itens na tabela especificada. O DynamoDB atualiza esse valor aproximad amente a cada seis horas. Alterações recentes podem não ser refletidas nesse valor. Tipo: número
`KeySchema`	A estrutura da chave primária (simples ou composta) da tabela. O par de nome-valor de `HashKeyElement` é obrigatório, e um par de nome-valor de `RangeKeyElement` é opcional (obrigatório apenas para chaves primárias compostas). O tamanho máximo da chave de hash é 2048 bytes. O tamanho máximo da chave de intervalo é 1024 bytes. Ambos os limites são aplicados separadamente (ou seja, você pode ter uma chave combinada de hash + intervalo 2048 + 1024). Para obter mais informações sobre chaves primárias, consulte Chave primária.
`ProvisionedThroughput`	O throughput da tabela especificada, consistin do em valores para `LastIncreaseDateTi me` (se aplicável), `LastDecreaseDateTi`

Nome	Descrição
	me (se aplicável), ReadCapacityUnits e WriteCapacityUnits . Se o throughput da tabela nunca tiver sido aumentada ou diminuída, o DynamoDB não retornará valores para esses elementos. Consulte Modo de capacidade provisionada do DynamoDB. Tipo: matriz
TableName	O nome da tabela solicitada. Tipo: string
TableSizeBytes	O tamanho total da tabela especificada, em bytes. O DynamoDB atualiza esse valor aproximadamente a cada seis horas. Alterações recentes podem não ser refletidas nesse valor. Tipo: número
TableStatus	O estado atual da tabela (CREATING, ACTIVE, DELETING ou UPDATING). Quando a tabela estiver no estado ACTIVE, você poderá adicionar dados.

Erros especiais

Nenhum erro é específico dessa operação.

Exemplos

Os exemplos a seguir mostram uma solicitação HTTP POST e uma resposta usando a operação DescribeTable para uma tabela chamada "comp-table". A tabela tem uma chave primária composta.

Exemplo de solicitação

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

Exemplo de resposta

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 543

{"Table":
    {"CreationDateTime":1.309988345372E9,
    "ItemCount":23,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": 1.309988345384E9,
 "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"users",
    "TableSizeBytes":949,
    "TableStatus":"ACTIVE"
    }
}
```

Ações relacionadas

- CreateTable
- DeleteTable
- ListTables

GetItem

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***

Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

A operação `GetItem` retorna um conjunto de `Attributes` para um item que corresponde à chave primária. Se não houver item correspondente, `GetItem` não retornará quaisquer dados.

A operação `GetItem` fornece uma leitura final consistente por padrão. Caso a aplicação não aceite leituras finais consistentes, use `ConsistentRead`. Embora essa operação possa demorar mais do que uma leitura padrão, ela sempre retorna o último valor atualizado. Para ter mais informações, consulte Consistência de leitura do DynamoDB.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
  "Key":
  {"HashKeyElement": {"S":"AttributeValue1"},
  "RangeKeyElement": {"N":"AttributeValue2"}
 },
 "AttributesToGet":["AttributeName3","AttributeName4"],
 "ConsistentRead":Boolean
}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela que contém o item solicitado. Tipo: string	Sim

Nome	Descrição	Obrigatório
`Key`	Os valores de chave primária que definem o item. Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: mapa de `HashKeyElement` para seu valor e `RangeKeyElement` para seu valor.	Sim
`AttributesToGet`	Matriz de nomes de atributos. Se os nomes de atributos não forem especificados, todos os atributos serão retornados. Se alguns atributos não forem encontrados, eles não serão exibidos no resultado. Tipo: matriz	Não
`ConsistentRead`	Se definido como `true`, uma leitura consistente será emitida; caso contrário, uma leitura eventualmente consistente será usada. Tipo: booliano	Não

Respostas

Sintaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
```

```
content-length: 144

{"Item":{
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName4":{"N":"AttributeValue4"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits": 0.5
}
```

Nome	Descrição
`Item`	Contém os atributos solicitados. Tipo: mapa de pares de nome-valor de atributo.
`ConsumedCapacityUnits`	O número de unidades de capacidade de leitura consumidas pela operação. Esse valor mostra o número utilizado no throughput provisionado. As solicitações de itens não existentes consomem o mínimo de unidades de capacidade de leitura, dependendo do tipo de leitura. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB. Tipo: número

Erros especiais

Não há erros específicos para esta operação.

Exemplos

Para obter exemplos sobre o uso do AWS SDK, consulte Trabalhar com itens e atributos no DynamoDB.

Exemplo de solicitação

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"comptable",
 "Key":
  {"HashKeyElement":{"S":"Julie"},
  "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

Exemplo de resposta

Observe que o valor ConsumedCapacityUnits é 1, pois o parâmetro opcional `ConsistentRead` está definido como `true`. Se `ConsistentRead` estiver definido como `false` (ou não especificado) para a mesma solicitação, a resposta é eventualmente consistente e o valor ConsumedCapacityUnits seria 0,5.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 72

{"Item":
 {"friends":{"SS":["Lynda, Aaron"]},
 "status":{"S":"online"}
 },
"ConsumedCapacityUnits": 1
}
```

ListTables

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Retorna uma matriz de todas as tabelas associadas à conta e ao endpoint atuais.

Cada endpoint do DynamoDB é totalmente independente. Por exemplo, se você tiver duas tabelas chamadas “MyTable”, uma em dynamodb.us-west-2.amazonaws.com e outra em dynamodb.us-east-1.amazonaws.com, elas serão completamente independentes e não compartilharão nenhum dado. A operação ListTables retorna todos os nomes de tabelas associados à conta que está fazendo a solicitação, para o endpoint que recebe a solicitação.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

Por padrão, a operação ListTables solicita todos os nomes de tabelas associados à conta que está fazendo a solicitação, para o endpoint que recebe a solicitação.

Nome	Descrição	Obrigatório
`Limit`	Um número de nomes de tabela máximos a serem retornados. Tipo: número inteiro	Não
`ExclusiveStartTableName`	O nome da tabela que inicia a lista. Se você já executou uma operação ListTables e recebeu um valor `LastEvaluatedTableName` na resposta, use esse valor aqui para continuar a lista.	Não

Nome	Descrição	Obrigatório
	Tipo: string	

Respostas

Sintaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"TableNames":["Table1","Table2","Table3"], "LastEvaluatedTableName":"Table3"}
```

Nome	Descrição
`TableNames`	Os nomes das tabelas associadas à conta atual no endpoint atual. Tipo: matriz
`LastEvaluatedTableName`	O nome da última tabela na lista atual, somente se algumas tabelas da conta e do endpoint não tiverem sido retornadas. Esse valor não existirá em uma resposta se todos os nomes de tabelas já tiverem sido retornados. Use esse valor como o `ExclusiveStartTableName` em uma nova solicitação para continuar a lista até que todos os nomes das tabelas sejam retornados. Tipo: string

Erros especiais

Nenhum erro é específico dessa operação.

Exemplos

Os exemplos a seguir mostram uma solicitação e uma resposta HTTP POST usando a operação ListTables.

Exemplo de solicitação

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"LastEvaluatedTableName":"comp5","TableNames":["comp3","comp4","comp5"]}
```

Ações relacionadas

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Cria um novo item ou substitui um item antigo por um novo (incluindo todos os atributos). Se algum item existe em uma tabela específica com a mesma chave primária, o novo item substitui completamente o item já existente. Você pode executar uma operação Put condicional (inserir um novo item, caso não exista um com a chave primária) ou substituir um item existente se ele tiver determinados valores de atributo.

Os valores de atributo não podem ser nulos; os atributos do tipo string e binário devem ter tamanhos maiores que zero; e os atributos do tipo conjunto não devem estar vazios. As solicitações com valores vazios serão rejeitadas com `ValidationException`.

> **Note**
>
> Para garantir que um novo item não substitua um item existente, use uma operação put condicional com `Exists` definido como `false` para o atributo de chave primária, ou atributos.

Para obter mais informações sobre o uso de `PutItem`, consulte Trabalhar com itens e atributos no DynamoDB.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Item":{
        "AttributeName1":{"S":"AttributeValue1"},
        "AttributeName2":{"N":"AttributeValue2"},
        "AttributeName5":{"B":"dmFsdWU="}
    },
    "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
    "ReturnValues":"ReturnValuesConstant"}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela que conterá o item. Tipo: string	Sim
`Item`	Um mapa dos atributos do item, e deve incluir os valores de chave primária que definem o item. Outros pares de nome-valor de atributo podem ser fornecidos para o item. Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: mapa de nomes de atributo para valores de atributo.	Sim
`Expected`	Designa um atributo para uma operação Put condicional. O parâmetro `Expected` permite que você forneça o nome do atributo, e se o DynamoDB deve ou não verificar se o valor de atributo já existe; ou se o valor de atributo existe e tem um valor específico antes de alterá-lo. Tipo: mapa de um nome de atributo para um valor de atributo, e se ele existe.	Não

Nome	Descrição	Obrigatório
`Expected:AttributeName`	O nome do atributo da operação put condicional. Tipo: string	Não

Nome	Descrição	Obrigatório

`Expected:AttributeName: ExpectedAttributeValue`	Use esse parâmetro para especificar se o valor já existe ou não para o par de nome-valor do atributo.
	
	A notação JSON seguinte substitui o item, se o atributo "Cor" ainda não existir para esse item:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	A notação JSON a seguir verifica se o atributo com o nome "Cor" tem um valor existente "Amarelo" antes de substituir o item:
	
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}`
	
	Por padrão, se você usar o parâmetro `Expected` e fornecer um `Value`, o DynamoDB presumirá que o atributo existe e tem um valor atual a ser substituído. Portanto, você não precisa especificar `{"Exists":true}` , pois ele está implícito. Você pode reduzir a solicitação para:
		Não

Nome	Descrição	Obrigatório
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Se você especificar `{"Exists":true}` sem um valor de atributo para verificar, o DynamoDB retornará um erro.	
`ReturnValues`	Use esse parâmetro se você quiser obter os pares de nome-valor de atributo antes que eles sejam atualizados com a solicitação `PutItem`. Os possíveis valores de parâmetro são `NONE` (padrão) ou `ALL_OLD`. Se `ALL_OLD` for especificado, e `PutItem` sobrescreveu um par de nome-valor de atributo, o conteúdo do item antigo é retornado. Se esse parâmetro não for fornecido ou for `NONE`, nada será retornado. Tipo: string	Não

Respostas

Sintaxe

A sintaxe a seguir presume que a solicitação especificou um parâmetro `ReturnValues` de `ALL_OLD`; caso contrário, a resposta terá apenas o elemento `ConsumedCapacityUnits`.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 85

{"Attributes":
 {"AttributeName3":{"S":"AttributeValue3"},
 "AttributeName2":{"SS":"AttributeValue2"},
 "AttributeName1":{"SS":"AttributeValue1"},
 },
"ConsumedCapacityUnits":1
}
```

Nome	Descrição
`Attributes`	Os valores de atributo antes da operação Put, mas somente se o parâmetro `ReturnValues` for especificado `ALL_OLD` na solicitação. Tipo: mapa de pares de nome-valor de atributo.
`ConsumedCapacityUnits`	O número de unidades de capacidade de gravação consumidas pela operação. Esse valor mostra o número utilizado no throughput provisionado. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB. Tipo: número

Erros especiais

Erro	Descrição
`ConditionalCheckFailedException`	Falha na verificação condicional. Um valor de atributo esperado não foi encontrado.
`ResourceNotFoundException`	O item especificado ou o atributo não foi encontrado.

Exemplos

Para obter exemplos sobre o uso do AWS SDK, consulte Trabalhar com itens e atributos no DynamoDB.

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Item":
  {"time":{"N":"300"},
  "feeling":{"S":"not surprised"},
  "user":{"S":"Riley"}
  },
 "Expected":
  {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
 "ReturnValues":"ALL_OLD"
}
```

Exemplo de resposta

```
HTTP/1.1 200
x-amzn-RequestId: 8952fa74-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 84
```

```
{"Attributes":
 {"feeling":{"S":"surprised"},
 "time":{"N":"300"},
 "user":{"S":"Riley"}},
"ConsumedCapacityUnits":1
}
```

Ações relacionadas

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

Consulta

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Uma operação `Query` obtém os valores de um ou mais itens e seus atributos por chave primária (`Query` só está disponível para tabelas de chave primária de hash e intervalo). Você deve fornecer um `HashKeyValue` específico e pode restringir o escopo da consulta usando operadores de comparação no `RangeKeyValue` da chave primária. Use o parâmetro `ScanIndexForward` para obter resultados em ordem progressiva ou inversa, por chave de intervalo.

Consultas que não retornam resultados consomem as unidades de capacidade de leitura mínimas de acordo com o tipo de leitura.

> **Note**
>
> Se o número total de itens que atendem aos parâmetros de consulta exceder o limite de 1 MB, a consulta será interrompida e os resultados serão retornados ao usuário com uma `LastEvaluatedKey` para continuar a consulta em uma operação subsequente. Ao contrário de uma operação de Verificação, uma operação de Consulta nunca retorna um conjunto de resultados vazio e uma `LastEvaluatedKey`. A `LastEvaluatedKey` apenas será fornecida se o resultado exceder 1 MB, ou se você tiver usado o parâmetro `Limit`.
> O resultado pode ser definido para uma leitura consistente usando o parâmetro `ConsistentRead`.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
[{"N":"AttributeValue2"}],"ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"AttributeName1"},
  "RangeKeyElement":{"N":"AttributeName2"}
 },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela que contém os itens solicitados.	Sim

Nome	Descrição	Obrigatório
	Tipo: string	
`AttributesToGet`	Matriz de nomes de atributos. Se os nomes de atributos não forem especificados, todos os atributos serão retornados. Se alguns atributos não forem encontrados, eles não serão exibidos no resultado. Tipo: matriz	Não

Nome	Descrição	Obrigatório
`Limit`	O número máximo de itens a serem retornados (não necessariamente o número de itens correspondentes). Se o DynamoDB processar o número de itens até o limite enquanto consulta a tabela, ele interromperá essa consulta e retornará os valores correspondentes até esse ponto, bem como uma `LastEvaluatedKey` a ser aplicada em uma operação subsequente para continuar a consulta. Além disso, se o conjunto de resultados exceder 1 MB antes que o DynamoDB atinja esse limite, ele interromperá a consulta e retornará os valores correspondentes, bem como uma `LastEvaluatedKey` a ser aplicada a uma operação subsequente para continuar a consulta. Tipo: número	Não
`ConsistentRead`	Se definido como `true`, uma leitura consistente será emitida; caso contrário, uma leitura eventualmente consistente será usada. Tipo: booliano	Não

Nome	Descrição	Obrigatório
`Count`	Se definido como `true`, o DynamoDB retornará um número total de itens que correspondem aos parâmetros de consulta, em vez de uma lista dos itens correspondentes e seus atributos. É possível aplicar o parâmetro `Limit` a consultas somente de contagem. Não defina `Count` como `true` ao fornecer uma lista de `AttributesToGet` . Caso contrário, o DynamoDB retornará um erro de validação . Para ter mais informações, consulte Contar os itens nos resultados. Tipo: booliano	Não
`HashKeyValue`	O valor do atributo do componente de hash da chave primária composta. Tipo: String, número ou binário	Sim

Nome	Descrição	Obrigatório
`RangeKeyCondition`	Um contêiner dos valores de atributos e operadores de comparação a serem usados para a consulta. Uma solicitação de consulta não requer uma `RangeKeyCondition`. Se você fornecer apenas o `HashKeyValue`, o DynamoDB retornará todos os itens com o valor do elemento de chave de hash especificado. Tipo: mapa	Não
`RangeKeyCondition : AttributeValueList`	Os valores de atributos a serem avaliados para os parâmetros de consulta. O `AttributeValueList` contém um valor de atributo, a menos que uma comparação `BETWEEN` seja especificada. Para a comparação `BETWEEN`, o `AttributeValueList` contém dois valores de atributo. Tipo: Um mapa de `AttributeValue` para uma `ComparisonOperator`.	Não

Nome	Descrição	Obrigatório
`RangeKeyCondition : ComparisonOperator`	Os critérios para avaliar os atributos fornecidos, como igual a, maior que etc. Veja a seguir os operadores de comparação válidos para uma operação de Consulta. **Note** As comparações de valor de string para maior que, igual a ou menor que são baseadas em valores de código de caractere s ASCII. Por exemplo, `a` é maior que `A`, e `aa` é maior que `B`. Para obter uma lista de valores de código, consulte http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Para Binary, o DynamoDB trata cada byte dos dados binários como não atribuído ao comparar valores binários, por exemplo, ao avaliar as expressões de consulta.	Não

Nome	Descrição	Obrigatório
	Tipo: string ou binário	
	`EQ` : Igual. Para `EQ`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não é igual a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
	`LE` : Menor ou igual a. Para `LE`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	
	`LT` : Menor que. Para `LT`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
	`GE` : Maior ou igual a. Para `GE`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	
	`GT` : Maior que. Para `GT`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
	`BEGINS_WITH` : procura um prefixo. Para `BEGINS_WITH` , `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String ou Binary (e não um Number ou conjunto) . O atributo de destino da comparação deve ser String ou Binary (não Number ou um conjunto).	

Nome	Descrição	Obrigatório
	`BETWEEN` : Maior ou igual ao primeiro valor e menor que ou igual ao segundo valor. Para `BETWEEN`, `AttributeValueList` deve conter dois elementos `AttributeValue` do mesmo tipo, seja String, Number ou Binary (e não um conjunto). Um atributo de destino corresponderá se o valor de destino for maior que ou igual ao primeiro elemento e menor que ou igual ao segundo elemento. Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não se compara a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
`ScanIndexForward`	Especifica o percurso crescente ou decrescente do índice. O DynamoDB retornará resultados que refletem a ordem solicitada determinada pela chave de intervalo: se o tipo de dados for Number, os resultados serão retornados em ordem numérica. Caso contrário, o percurso se baseará em valores do código de caracteres ASCII. Tipo: booliano O padrão é `true` (crescente).	Não

Nome	Descrição	Obrigatório
`ExclusiveStartKey`	A chave primária do item a partir do qual a consulta anterior deve continuar. Uma consulta anterior poderá fornecer esse valor como a `LastEvaluatedKey` se essa operação de consulta tiver sido interrompida antes da conclusão da consulta, seja devido ao tamanho do conjunto de resultados ou ao parâmetro `Limit`. A `LastEvaluatedKey` pode ser retornada em uma nova solicitação de consulta para continuar a operação a partir desse ponto. Tipo: `HashKeyElement` ou `HashKeyElement` e `RangeKeyElement` para uma chave primária composta.	Não

Respostas

Sintaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"N":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
```

```
    },{
    "AttributeName1":{"S":"AttributeValue3"},
    "AttributeName2":{"N":"AttributeValue4"},
    "AttributeName3":{"S":"AttributeValue3"},
    "AttributeName5":{"B":"dmFsdWU="}
}],
    "LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},
                        "RangeKeyElement":{"AttributeValue4":"N"}
     },
     "ConsumedCapacityUnits":1
}
```

Nome	Descrição
`Items`	Atributos do item que atendem aos parâmetros de consulta. Tipo: mapa de nomes de atributo e seus tipos de dados e valores.
`Count`	Número de itens na resposta. Para ter mais informações, consulte Contar os itens nos resultados. Tipo: número
`LastEvaluatedKey`	Chave primária do item no qual a operação de consulta foi interrompida, incluindo o conjunto de resultados anterior. Use esse valor para iniciar uma nova operação, excluindo o valor na nova solicitação. A `LastEvaluatedKey` é `null` quando o conjunto inteiro de resultados da consulta está completo (ou seja, a operação processou a "última página"). Tipo: `HashKeyElement` ou `HashKeyElement` e `RangeKeyElement` para uma chave primária composta.

Nome	Descrição
`ConsumedCapacityUnits`	O número de unidades de capacidade de leitura consumidas pela operação. Esse valor mostra o número utilizado no throughput provisionado. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB. Tipo: número

Erros especiais

Erro	Descrição
`ResourceNotFoundException`	A tabela especificada não foi encontrada.

Exemplos

Para obter exemplos sobre o uso do AWS SDK, consulte Consultar tabelas no DynamoDB.

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"John"},
  "RangeKeyElement":{"S":"The Matrix"}
 }
```

```
}
```

Exemplo de resposta

```
HTTP/1.1 200
x-amzn-RequestId: 3647e778-71eb-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The End"}
 },{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The Beatles"}
 }],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"John"},"RangeKeyElement":{"S":"The
 Beatles"}},
"ConsumedCapacityUnits":1
}
```

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
 "ScanIndexForward":false}
```

Exemplo de resposta

```
HTTP/1.1 200
x-amzn-RequestId: 8b9ee1ad-774c-11e0-9172-d954e38f553a
content-type: application/x-amz-json-1.0
content-length: 119

{"Count":1,"Items":[{
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"},
 "year":{"N":"1980"}
 }],
"ConsumedCapacityUnits":1
}
```

Ações relacionadas

- Verificar

Verificar

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

A operação `Scan` retorna um ou mais itens e seus atributos realizando uma verificação completa de uma tabela. Forneça um `ScanFilter` para obter resultados mais específicos.

> **Note**
>
> Se o número total de itens verificados exceder o limite de 1 MB, a verificação será interrompida e os resultados serão retornados para o usuário com `LastEvaluatedKey` para continuar a verificação em uma operação posterior. Os resultados também incluem o número

de itens que excedem o limite. O resultado de uma verificação pode ser que nenhum dado corresponda aos critérios de filtro.
O conjunto de resultados é eventualmente consistente.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Limit": 2,
    "ScanFilter":{
        "AttributeName1":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
    },
    "ExclusiveStartKey":{
        "HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"}
    },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela que contém os itens solicitados. Tipo: string	Sim
`AttributesToGet`	Matriz de nomes de atributos. Se os nomes de atributos não forem especificados, todos os atributos serão retornados. Se alguns atributos não forem	Não

Nome	Descrição	Obrigatório
	encontrados, eles não serão exibidos no resultado. Tipo: matriz	
`Limit`	O número máximo de itens a serem avaliados (não necessariamente o número de itens correspondentes). Se o DynamoDB processa o número de itens até o limite ao processar os resultado s, ele para e retorna os valores correspondentes até esse ponto, além de um `LastEvaluatedKey` para aplicar em uma operação subsequente para continuar a recuperação de itens. Além disso, se o tamanho do conjunto de dados exceder 1 MB antes que o DynamoDB atinja esse limite, a verificação será interrompida e retornará os valores correspondentes até o limite, além de uma `LastEvaluatedKey` para aplicar em uma operação subsequente para continuar a verificação. Tipo: número	Não

Nome	Descrição	Obrigatório
`Count`	Se definido como `true`, o DynamoDB retorna um número total de itens para a operação Scan, mesmo se a operação não tiver itens correspondentes ao filtro associado. Você pode aplicar o parâmetro de limite somente para contar verificações. Não defina `Count` como `true` ao fornecer uma lista de `AttributesToGet` . Caso contrário, o DynamoDB retornará um erro de validação . Para ter mais informações, consulte Contar os itens nos resultados. Tipo: booliano	Não
`ScanFilter`	Avalia os resultados da verificação e retorna apenas os valores desejados. Várias condições são tratadas como operações "AND": todas as condições devem ser atendidas para serem incluídas nos resultados. Tipo: um mapa de nomes de atributo para valores com operadores de comparação.	Não

Nome	Descrição	Obrigatório
`ScanFilter` :`AttributeValueList`	Os valores e as condições para avaliar os resultados da verificação do filtro. Tipo: Um mapa de `AttributeValue` para uma `Condition` .	Não

Nome	Descrição	Obrigatório
`ScanFilter : ComparisonOperator`	Os critérios para avaliar os atributos fornecidos, como igual a, maior que etc. A seguir estão os operadores de comparação válidos para uma operação de verificação. **Note** As comparações de valor de string para maior que, igual a ou menor que são baseadas em valores de código de caracteres ASCII. Por exemplo, `a` é maior que `A`, e `aa` é maior que `B`. Para obter uma lista de valores de código, consulte http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Para Binary, o DynamoDB trata cada byte dos dados binários como não atribuído ao comparar valores binários, por exemplo, ao avaliar as expressões de consulta.	Não

Nome	Descrição	Obrigatório
	Tipo: string ou binário	
	`EQ` : Igual. Para `EQ`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não é igual a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
	`NE` : Não é igual. Para `NE`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não é igual a `{"NS":["6", "2", "1"]}`.	
	`LE` : Menor ou igual a. Para `LE`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
	`LT` : Menor que. Para `LT`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	
	`GE` : Maior ou igual a. Para `GE`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
	`GT` : Maior que. Para `GT`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	
	`NOT_NULL` : o atributo existe.	
	`NULL` : o atributo não existe.	

Nome	Descrição	Obrigatório
	`CONTAINS` : verifica uma subsequência ou valor em um conjunto. Para `CONTAINS`, `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se o atributo de destino da comparação for uma String, a operação procurará uma substring correspondente. Se o atributo de destino da comparação for Binary, a operação procurará uma subsequência do destino que corresponda à entrada. Se o atributo de destino da comparação for um conjunto ("SS", "NS" ou "BS"), a operação procurará um membro do conjunto (não uma substring).	

Nome	Descrição	Obrigatório
	`NOT_CONTAINS` : verifica a ausência de uma subsequência ou a ausência de um valor em um conjunto. Para `NOT_CONTAINS` , `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (e não um conjunto). Se o atributo de destino da comparação for String, a operação verificará a ausência de uma substring correspondente. Se o atributo de destino da comparação for Binary, a operação verificará a ausência de uma subsequência do destino que corresponda à entrada. Se o atributo de destino da comparação for um conjunto ("SS", "NS" ou "BS"), a operação verificará a ausência de um membro do conjunto (não uma substring).	

Nome	Descrição	Obrigatório
	`BEGINS_WITH` : procura um prefixo. Para `BEGINS_WITH` , `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String ou Binary (e não um Number ou conjunto) . O atributo de destino da comparação deve ser String ou Binary (não Number ou um conjunto).	
	`IN` : verifica se existem correspondências exatas. Para `IN`, `AttributeValueList` pode conter mais de um `AttributeValue` do tipo String, Number ou Binary (não um conjunto). O atributo de destino da comparação deve ser do mesmo tipo e ter o valor exato para correspon der. Um atributo String nunca corresponde a um String set.	

Nome	Descrição	Obrigatório
	`BETWEEN` : Maior ou igual ao primeiro valor e menor que ou igual ao segundo valor. Para `BETWEEN`, `AttributeValueList` deve conter dois elementos `AttributeValue` do mesmo tipo, seja String, Number ou Binary (e não um conjunto). Um atributo de destino corresponderá se o valor de destino for maior que ou igual ao primeiro elemento e menor que ou igual ao segundo elemento. Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não se compara a `{"N":"6"}` . Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.	

Nome	Descrição	Obrigatório
`ExclusiveStartKey`	A chave primária do item a partir do qual a verificação anterior deve continuar. Uma verificação anterior deve fornecer esse valor, se essa operação Scan foi interrompida antes da verificação da tabela inteira, seja por causa do tamanho do conjunto de resultados ou do parâmetro `Limit`. O valor `LastEvaluatedKey` pode ser passado de volta em uma nova solicitação de verificação para continuar a operação a partir desse ponto. Tipo: `HashKeyElement` ou `HashKeyElement` e `RangeKeyElement` para uma chave primária composta.	Não

Respostas

Sintaxe

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 229

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
```

```
    "AttributeName1":{"S":"AttributeValue4"},
    "AttributeName2":{"S":"AttributeValue5"},
    "AttributeName3":{"S":"AttributeValue6"},
    "AttributeName5":{"B":"dmFsdWU="}
    }],
    "LastEvaluatedKey":
        {"HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"},
    "ConsumedCapacityUnits":1,
    "ScannedCount":2}
}
```

Nome	Descrição
`Items`	Contêiner dos atributos que atendem aos parâmetros da operação. Tipo: mapa de nomes de atributo e seus tipos de dados e valores.
`Count`	Número de itens na resposta. Para ter mais informações, consulte Contar os itens nos resultados. Tipo: número
`ScannedCount`	Número de itens na verificação completa antes de todos os filtros serem aplicados. Um valor `ScannedCount` alto com poucos ou sem `Count` resultados indica uma operação Scan ineficiente. Para ter mais informações, consulte Contar os itens nos resultados. Tipo: número
`LastEvaluatedKey`	Chave primária do item em que a operação Scan foi interrompida. Forneça esse valor em uma operação Scan subsequente para continuar a operação a partir desse ponto.

Nome	Descrição
	O valor `LastEvaluatedKey` é `null` quando o conjunto de resultados da verificação inteira está completo (ou seja, a operação processou a “última página”).
`ConsumedCapacityUnits`	O número de unidades de capacidade de leitura consumidas pela operação. Esse valor mostra o número utilizado no throughput provisionado. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB. Tipo: número

Erros especiais

Erro	Descrição
`ResourceNotFoundException`	A tabela especificada não foi encontrada.

Exemplos

Para obter exemplos sobre o uso do AWS SDK, consulte Verificar tabelas no DynamoDB.

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

Exemplo de resposta

```
HTTP/1.1 200
x-amzn-RequestId: 4e8a5fa9-71e7-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 465

{"Count":4,"Items":[{
 "date":{"S":"1980"},
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"}
 },{
 "date":{"S":"1999"},
 "fans":{"SS":["Ziggy","Laura","Dean"]},
 "name":{"S":"Matrix"},
 "rating":{"S":"*****"}
 },{
 "date":{"S":"1976"},
 "fans":{"SS":["Riley"]},"
 name":{"S":"The Shaggy D.A."},
 "rating":{"S":"**"}
 },{
 "date":{"S":"1985"},
 "fans":{"SS":["Fox","Lloyd"]},
 "name":{"S":"Back To The Future"},
 "rating":{"S":"****"}
 }],
    "ConsumedCapacityUnits":0.5
 "ScannedCount":4}
```

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
 "ScanFilter":
  {"time":
```

```
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 }
}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 262
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":2,
 "Items":[
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Casey"}},
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Fredy"}
  }],
"ConsumedCapacityUnits":0.5
"ScannedCount":4
}
```

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 },
```

```
 "ExclusiveStartKey":
  {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 232
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":1,
 "Items":[
  {"friends":{"SS":["Jane","James","John"]},
  "status":{"S":"exercising"},
  "time":{"N":"2200"},
  "user":{"S":"Roger"}}
 ],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"250"}},
"ConsumedCapacityUnits":0.5
"ScannedCount":2
}
```

Ações relacionadas

- Consulta
- BatchGetItem

UpdateItem

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Edita os atributos de um item existente. Você pode executar uma atualização condicional (insira um novo par de nome-valor de atributo se ele não existir, ou substitua um par de nome-valor existente se ele tiver determinados valores de atributos esperados).

> **Note**
>
> Não é possível atualizar os atributos da chave primária usando UpdateItem. Em vez disso, exclua o item e use PutItem para criar um novo item com novos atributos.

A operação UpdateItem inclui um parâmetro `Action` que define como executar a atualização. Você pode inserir, excluir ou adicionar valores de atributo.

Os valores de atributo não podem ser nulos; os atributos do tipo string e binário devem ter tamanhos maiores que zero; e os atributos do tipo conjunto não devem estar vazios. As solicitações com valores vazios serão rejeitadas com `ValidationException`.

Se um item existente tiver a chave primária especificada:

- PUT: adiciona o atributo especificado. Se o atributo existir, ele será substituído pelo novo valor.
- DELETE: se nenhum valor for especificado, o atributo e seu valor serão removidos. Se um conjunto de valores for especificado, os valores no conjunto especificado serão removidos do conjunto antigo. Portanto, se o valor do atributo incluir [a, b, c] e a ação de exclusão incluir [a, c], o valor do atributo final será [b]. O tipo de valor especificado deve corresponder ao tipo de valor existente. Não é válido especificar um conjunto vazio.
- ADD: use a ação de adicionar apenas para números ou se o atributo de destino for um conjunto (incluindo conjuntos de strings). ADD não funcionará se o atributo de destino for um único valor de string ou um valor binário escalar. O valor especificado é adicionado a um valor numérico (aumentando ou diminuindo o valor numérico existente) ou adicionado como um valor extra em um conjunto de strings. Se um conjunto de valores for especificado, os valores serão adicionados ao conjunto existente. Por exemplo, se o conjunto original for [1,2] e o valor fornecido for [3], após a operação de adição, o conjunto será [1,2,3] e não [4,5]. Ocorrerá um erro se uma ação Adicionar for especificada para um atributo de conjunto e o tipo de atributo especificado não corresponder ao tipo de conjunto existente.

  Se você usar ADD para um atributo que não existe, o atributo e seus valores serão adicionados ao item.

Se nenhum item corresponder à chave primária especificada:

- PUT: cria um novo item com a chave primária especificada. Em seguida, adiciona o atributo especificado.
- DELETE - Nada acontece.
- ADD: cria um item com a chave primária e o número fornecidos (ou conjunto de números) para o valor de atributo. Não é válido para um tipo string ou binário.

> **Note**
>
> Se você usar ADD para aumentar ou reduzir um valor de número de um item que não existe antes da atualização, o DynamoDB usará `0` como o valor inicial. Além disso, se você atualizar um item usando ADD para aumentar ou diminuir um valor de número de um atributo que não existe antes da atualização (mas o item existe), o DynamoDB usará `0` como o valor inicial. Por exemplo, você usa ADD para adicionar `+3` a um atributo que não existia antes da atualização. O DynamoDB usa `0` para o valor inicial, e o valor após a atualização é `3`.

Para obter mais informações sobre o uso dessa operação, consulte Trabalhar com itens e atributos no DynamoDB.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},
        "RangeKeyElement":{"N":"AttributeValue2"}},
    "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
    "ReturnValues":"ReturnValuesConstant"
```

```
}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela que contém o item a ser atualizado. Tipo: string	Sim
`Key`	A chave primária que define o item. Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: mapa de `HashKeyElement` para seu valor e `RangeKeyElement` para seu valor.	Sim
`AttributeUpdates`	Mapa de nome de atributo para o novo valor e ação para a atualização. Os nomes de atributo especificam os atributos a serem modificados, e não podem conter quaisquer atributos da chave primária. Tipo: mapa de nome de atributo, valor e uma ação para a atualização do atributo.	
`AttributeUpdates :Action`	Especifica como executar a atualização. Valores possíveis : `PUT` (padrão) `ADD` ou `DELETE`. As semânticas são explicadas na descrição de UpdateItem.	Não

Nome	Descrição	Obrigatório
	Tipo: string Padrão: PUT	
`Expected`	Designa um atributo para uma atualização condicional. O parâmetro `Expected` permite que você forneça o nome do atributo, e se o DynamoDB deve ou não verificar se o valor de atributo já existe; ou se o valor de atributo existe e tem um valor específico antes de alterá-lo. Tipo: mapa de nomes de atributo.	Não
`Expected:AttributeName`	O nome do atributo da operação put condicional. Tipo: string	Não

Nome	Descrição	Obrigatório

`Expected:AttributeName: ExpectedAttributeValue`	Use esse parâmetro para especificar se o valor já existe ou não para o par de nome-valor do atributo.
	
	A notação JSON seguinte atualiza o item, se o atributo "Cor" ainda não existir para esse item:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	A notação JSON seguinte verifica se o atributo com o nome "Cor" tem um valor existente "Amarelo" antes de atualizar o item:
	
	`"Expected" : {"Color":{"Exists":true},{"Value": {"S":"Yellow"}}}`
	
	Por padrão, se você usar o parâmetro `Expected` e fornecer um `Value`, o DynamoDB presumirá que o atributo existe e tem um valor atual a ser substituído. Portanto, você não precisa especificar `{"Exists" :true}` , pois ele está implícito. Você pode reduzir a solicitação para:
		Não

Nome	Descrição	Obrigatório
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Se você especificar `{"Exists":true}` sem um valor de atributo para verificar, o DynamoDB retornará um erro.	

Nome	Descrição	Obrigatório
`ReturnValues`	Use esse parâmetro se você quiser obter os pares de nome-valor de atributo antes que eles sejam atualizados com a solicitação `UpdateItem` . Possíveis valores de parâmetro são `NONE` (padrão) ou `ALL_OLD`, `UPDATED_OLD` , `ALL_NEW` ou `UPDATED_NEW` . Se `ALL_OLD` for especificado, e `UpdateItem` sobrescreveu um par de nome-valor de atributo, o conteúdo do item antigo é retornado. Se esse parâmetro não for fornecido ou for `NONE`, nada será retornado. Se `ALL_NEW` for especificado, todos os atributos da nova versão do item serão retornados. Se `UPDATED_NEW` for especificado, somente as novas versões dos atributos atualizados serão retornadas. Tipo: string	Não

Respostas

Sintaxe

A sintaxe a seguir presume que a solicitação especificou um parâmetro `ReturnValues` de `ALL_OLD`; caso contrário, a resposta terá apenas o elemento `ConsumedCapacityUnits`.

```
HTTP/1.1 200
```

```
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 140

{"Attributes":{
 "AttributeName1":{"S":"AttributeValue1"},
 "AttributeName2":{"S":"AttributeValue2"},
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits":1
}
```

Nome	Descrição
`Attributes`	Um mapa de pares de nome-valor de atributo, mas somente se o parâmetro `ReturnValues` for especificado como algo diferente de `NONE` na solicitação. Tipo: mapa de pares de nome-valor de atributo.
`ConsumedCapacityUnits`	O número de unidades de capacidade de gravação consumidas pela operação. Esse valor mostra o número utilizado no throughput provisionado. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB. Tipo: número

Erros especiais

Erro	Descrição
`ConditionalCheckFailedException`	Falha na verificação condicional. Atributo ("+ name +"), o valor é ("+ value +") mas o esperado era ("+ expValue +")

Erro	Descrição
`ResourceNotFoundExceptions`	O item especificado ou o atributo não foi encontrado.

Exemplos

Para obter exemplos sobre o uso do AWS SDK, consulte Trabalhar com itens e atributos no DynamoDB.

Exemplo de solicitação

```
// This header is abbreviated. For a sample of a complete header, see API de baixo
 nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
    "Key":
        {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
    "AttributeUpdates":
        {"status":{"Value":{"S":"online"},
        "Action":"PUT"}},
    "Expected":{"status":{"Value":{"S":"offline"}}},
    "ReturnValues":"ALL_NEW"
}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMHO7F01Q9P7Q6QMKMMI3R3QRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 121
Date: Fri, 26 Aug 2011 21:05:00 GMT

{"Attributes":
    {"friends":{"SS":["Lynda, Aaron"]},
    "status":{"S":"online"},
    "time":{"N":"1307654350"},
    "user":{"S":"Julie"}},
```

```
"ConsumedCapacityUnits":1
}
```

Ações relacionadas

- PutItem
- DeleteItem

UpdateTable

> **Important**
>
> ***Esta seção refere-se à versão de API 2011-12-05, que está obsoleta e não deve ser usada para novos aplicativos.***
>
> Para obter a documentação da API de baixo nível atual, consulte a Referência da API do Amazon DynamoDB.

Descrição

Atualiza os valores de throughput provisionado da tabela especificada. Definir o throughput de uma tabela ajuda você a gerenciar a performance e é parte do recurso de throughput provisionado do DynamoDB. Para ter mais informações, consulte Modo de capacidade provisionada do DynamoDB.

Os valores de throughput provisionado podem sofrer upgrade ou downgrade com base nos máximos e mínimos listados em Cotas no Amazon DynamoDB.

A tabela deverá estar no estado `ACTIVE` para que essa operação seja bem-sucedida. UpdateTable é uma operação assíncrona. Durante a execução da operação, a tabela está no estado `UPDATING`. Enquanto estiver no estado `UPDATING`, a tabela ainda terá o throughput provisionado de antes da chamada. A nova configuração de throughput provisionado entra em vigor somente quando a tabela retorna para o estado `ACTIVE` após a operação UpdateTable.

Solicitações

Sintaxe

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
```

```
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Nome	Descrição	Obrigatório
`TableName`	O nome da tabela a ser atualizada. Tipo: string	Sim
`ProvisionedThroughput`	Novo throughput da tabela especificada consistindo em valores para `ReadCapacityUnits` e `WriteCapacityUnits`. Consulte Modo de capacidade provisionada do DynamoDB. Tipo: matriz	Sim
`ProvisionedThroughput :ReadCapacityUnits`	Define o número mínimo de `ReadCapacityUnits` consistentes consumidas por segundo para a tabela especificada antes que o DynamoDB balanceie a carga com outras operações. As operações de leitura eventualmente consistente requerem menos esforço que uma operação de leitura consistente, portanto, uma	Sim

Nome	Descrição	Obrigatório
	definição de 50 `ReadCapacityUnits` consistentes por segundo oferece 100 `ReadCapacityUnits` eventualmente consistentes por segundo. Tipo: número	
`ProvisionedThroughput:WriteCapacityUnits`	Define o número mínimo de `WriteCapacityUnits` consumidas por segundo para a tabela especificada antes que o DynamoDB balanceie a carga com outras operações. Tipo: número	Sim

Respostas

Sintaxe

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/json
Content-Length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeValue1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeValue2","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
```

```
        "WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"UPDATING"}}
```

Nome	Descrição
`CreationDateTime`	Data em que a tabela foi criada. Tipo: número
`KeySchema`	A estrutura da chave primária (simples ou composta) da tabela. O par de nome-valor de `HashKeyElement` é obrigatório, e um par de nome-valor de `RangeKeyElement` é opcional (obrigatório apenas para chaves primárias compostas). O tamanho máximo da chave de hash é 2048 bytes. O tamanho máximo da chave de intervalo é 1024 bytes. Ambos os limites são aplicados separadamente (ou seja, você pode ter uma chave combinada de hash + intervalo 2048 + 1024). Para obter mais informações sobre chaves primárias, consulte Chave primária. Tipo: mapa de `HashKeyElement` , ou `HashKeyElement` e `RangeKeyElement` para uma chave primária composta.
`ProvisionedThroughput`	Configurações de throughput atuais da tabela especificada, incluindo valores para `LastIncreaseDateTime` (se aplicável), `LastDecreaseDateTime` (se aplicável), Tipo: matriz
`TableName`	O nome da tabela atualizada. Tipo: string

Nome	Descrição
`TableStatus`	O estado atual da tabela (`CREATING`, `ACTIVE`, `DELETING` ou `UPDATING`), que deve ser `UPDATING`. Use a operação DescribeTables para verificar o status da tabela. Tipo: string

Erros especiais

Erro	Descrição
`ResourceNotFoundException`	A tabela especificada não foi encontrada.
`ResourceInUseException`	A tabela não está no estado `ACTIVE`.

Exemplos

Exemplo de solicitação

```
// This header is abbreviated.
// For a sample of a complete header, see API de baixo nível do DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Exemplo de resposta

```
HTTP/1.1 200 OK
content-type: application/x-amz-json-1.0
content-length: 390
```

```
Date: Sat, 19 Nov 2011 00:46:47 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"comp1",
    "TableStatus":"UPDATING"}
}
```

Ações relacionadas

- CreateTable
- DescribeTables
- DeleteTable

Parâmetros condicionais legados do DynamoDB

Este documento oferece uma visão geral dos parâmetros condicionais herdados no DynamoDB e recomenda o uso dos novos parâmetros de expressão. Ele aborda detalhes sobre determinados parâmetros, como AttributesToGet, AttributeUpdates, ConditionalOperator, Expected, KeyConditions, QueryFilter e ScanFilter, e fornece exemplos de como usar os novos parâmetros de expressão como substitutos.

> **Important**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB.
> Além disso, o DynamoDB não permite combinar parâmetros condicionais herdados e parâmetros de expressão em uma única chamada. Por exemplo, chamar a operação `Query` com `AttributesToGet` e `ConditionExpression` resultará em um erro.

A tabela a seguir mostra as operações de API do DynamoDB que ainda comportam esses parâmetros herdados e quais parâmetros de expressão devem ser usados. Esta tabela pode ser útil se você estiver considerando atualizar os aplicativos para que eles usem os parâmetros de expressão.

Se você usar essa operação de API...	Com estes parâmetros herdados...	Use este parâmetro de expressão em vez disso
`BatchGetItem`	`AttributesToGet`	`ProjectionExpression`
`DeleteItem`	`Expected`	`ConditionExpression`
`GetItem`	`AttributesToGet`	`ProjectionExpression`
`PutItem`	`Expected`	`ConditionExpression`
`Query`	`AttributesToGet`	`ProjectionExpression`
	`KeyConditions`	`KeyConditionExpression`
	`QueryFilter`	`FilterExpression`
`Scan`	`AttributesToGet`	`ProjectionExpression`
	`ScanFilter`	`FilterExpression`
`UpdateItem`	`AttributeUpdates`	`UpdateExpression`
	`Expected`	`ConditionExpression`

As seções a seguir oferecem mais informações sobre os parâmetros condicionais herdados.

Tópicos

- AttributesToGet (herdado)
- AttributeUpdates (herdado)
- ConditionalOperator (herdado)
- Expected (herdado)

- KeyConditions (herdado)
- QueryFilter (herdado)
- ScanFilter (herdado)
- Criar condições com parâmetros herdados

AttributesToGet (herdado)

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB.
> Para obter informações específicas sobre o novo parâmetro que substitui este, Use ProjectionExpression em vez disso.

O parâmetro condicional herdado `AttributesToGet` é um conjunto de um ou mais atributos a serem recuperados do DynamoDB. Se os nomes de atributo não forem fornecidos, todos os atributos serão retornados. Se qualquer um dos atributos solicitados não for encontrado, ele não aparecerá no resultado.

`AttributesToGet` permite que você recupere atributos do tipo List ou Map; no entanto, ele não pode recuperar elementos individuais em uma lista ou um mapa.

Observe que `AttributesToGet` não afeta o consumo de throughput provisionado. O DynamoDB determina as unidades de capacidade consumidas com base no tamanho do item, e não na quantidade de dados que são retornados para um aplicativo.

Use ProjectionExpression em vez disso: exemplo

Suponha que você quisesse recuperar um item da tabela Music, mas desejasse retornar somente alguns dos atributos. Você poderia usar uma solicitação `GetItem` com um parâmetro `AttributesToGet`, como neste exemplo da AWS CLI:

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
```

```
    }'
```

Você pode usar `ProjectionExpression` em vez disso:

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (herdado)

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB. Para obter informações específicas sobre o novo parâmetro que substitui este, Use UpdateExpression em vez disso..

Em uma operação `UpdateItem`, o parâmetro condicional herdado `AttributeUpdates` contém os nomes dos atributos a serem modificados, a ação a ser realizada em cada um e o novo valor de cada um. Se você estiver atualizando um atributo que é um atributo de chave de índice de quaisquer índices dessa tabela, o tipo de atributo deverá coincidir com o tipo de chave de índice definido na `AttributesDefinition` da descrição da tabela. Você pode usar `UpdateItem` para atualizar todos os atributos que não são chave.

Os valores de atributo não podem ser nulos. Os atributos do tipo String e Binary devem ter tamanhos maior que zero. Os atributos do tipo Set não podem ficar vazios. As solicitações com valores vazios serão rejeitadas com uma exceção `ValidationException`.

Cada elemento `AttributeUpdates` consiste em um nome de atributo a ser modificado, junto com o seguinte:

- `Value` - O novo valor, se aplicável, desse atributo.
- `Action`: um valor que especifica como executar a atualização. Essa ação só é válida para um atributo existente cujo tipo de dados é Number ou é Set; não use `ADD` para outros tipos de dados.

Se um item com a chave primária especificada for encontrado na tabela, os seguintes valores executam as ações a seguir:

- `PUT`: adiciona o atributo especificado ao item. Se o atributo já existir, ele será substituído pelo novo valor.
- `DELETE`: remove o atributo e o respectivo valor, se nenhum valor for especificado para `DELETE`. O tipo de dados do valor especificado deve corresponder ao tipo de dados do valor existente.

  Se um conjunto de valores for especificado, esses valores serão subtraídos do conjunto antigo. Por exemplo, se o valor do atributo era o conjunto `[a,b,c]` e a ação `DELETE` especifica `[a,c]`, o valor do atributo final será `[b]`. Especificar um conjunto vazio é um erro.
- `ADD`: adiciona o valor especificado ao item se o atributo ainda não existir. Se o atributo não existir, o comportamento de `ADD` dependerá do tipo de dados do atributo:
  - Se o atributo existente for um número e se `Value` também for um número, então `Value` será matematicamente adicionado ao atributo existente. Se `Value` for um número negativo, ele será subtraído do atributo existente.

    > **Note**
    >
    > Se você usar `ADD` para aumentar ou reduzir um valor de número de um item que não existe antes da atualização, o DynamoDB usará `0` como o valor inicial.
    > Da mesma forma, se você usar `ADD` para um item existente para aumentar ou diminuir um valor de atributo que não existia antes da atualização, o DynamoDB usará `0` como o valor inicial. Por exemplo, suponha que o item que você deseja atualizar não tem um atributo chamado itemcount, mas você decide `ADD` o número `3` a esse atributo mesmo assim. O DynamoDB criará o atributo itemcount, definir seu valor inicial como `0` e, finalmente, adicionar `3` a ele. O resultado será um novo atributo itemcount, com um valor `3`.
  - Se o tipo de dados existente for um conjunto, e se `Value` também for um conjunto, então `Value` será anexado ao conjunto existente. Por exemplo, se o valor do atributo for o conjunto `[1,2]` e a ação `ADD` especificou `[3]`, o valor do atributo final será `[1,2,3]`. Ocorrerá um erro se uma ação `ADD` for especificada para um atributo de conjunto e o tipo de atributo especificado não corresponder ao tipo de conjunto existente.

    Ambos os conjuntos devem ter o mesmo tipo de dados primitivo. Por exemplo, se o tipo de dados existente for um conjunto de strings, `Value` também deve ser um conjunto de strings.

Se nenhum item com a chave especificada for encontrado na tabela, os seguintes valores executam as ações a seguir:

- `PUT` - faz com que o DynamoDB crie um novo item com a chave primária especificada e, em seguida, adiciona o atributo.
- `DELETE`: nada acontece, pois os atributos não podem ser excluídos de um item inexistente. A operação é bem-sucedida, mas o DynamoDB não cria um novo item.
- `ADD`: faz com que o DynamoDB crie um item com o número e a chave primária fornecidos (ou conjunto de números) para o valor de atributo. Os únicos tipos de dados permitidos são Number e Number Set.

Se você fornecer quaisquer atributos que sejam parte de uma chave de índice, então, os tipos de dados desses atributos devem corresponder aos do esquema na definição de atributo da tabela.

Use UpdateExpression em vez disso: exemplo

Suponha que você quisesse modificar um item na tabela Music. Você poderia usar uma solicitação `UpdateItem` com um parâmetro `AttributeUpdates`, como neste exemplo da AWS CLI:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --attribute-updates '{
        "Genre": {
            "Action": "PUT",
            "Value": {"S":"Rock"}
        }
    }'
```

Você pode usar `UpdateExpression` em vez disso:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
```

```
    --update-expression 'SET Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

ConditionalOperator (herdado)

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB.

O parâmetro condicional herdado `ConditionalOperator` é um operador lógico usado para ser aplicado às condições em um mapa `Expected`, `ScanFilter` ou `QueryFilter`:

- AND – Se todas as condições forem avaliadas como verdadeiras, o mapa inteiro será avaliado como verdadeiro.
- OR: se pelo menos uma das condições for avaliada como verdadeira, o mapa inteiro será avaliado como verdadeiro.

Se você omitir `ConditionalOperator`, então, `AND` será o padrão.

A operação será bem-sucedida somente se o mapa inteiro for avaliado como verdadeiro.

> **Note**
>
> Este parâmetro não tem suporte a atributos do tipo List ou Map.

Expected (herdado)

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB. Para obter informações específicas sobre o novo parâmetro que substitui este, Use ConditionExpression em vez disso..

O parâmetro condicional herdado `Expected` é um bloco condicional para uma operação `UpdateItem`. `Expected` é um mapa de pares de atributo/condição. Cada elemento do mapa consiste em um nome de atributo, um operador de comparação e um ou mais valores. O DynamoDB compara o atributo com os valores que você forneceu usando o operador de comparação. Para cada elemento `Expected`, o resultado da avaliação é verdadeiro ou falso.

Se você especificar mais de um elemento no mapa `Expected`, por padrão, todas as condições deverão ser avaliadas como verdadeiras. Em outras palavras, as condições são combinadas usando o operador `AND`. (Em vez disso, você pode usar o parâmetro `ConditionalOperator` processar as condições com o operador OR. Se fizer isso, pelo menos uma das condições deverá ser avaliada como true, em vez de todas elas.)

Se o mapa `Expected` for avaliado como verdadeiro, a operação condicional será bem-sucedida; caso contrário, há uma falha.

`Expected` contém o seguinte:

- `AttributeValueList`: um ou mais valores para avaliar em relação ao atributo fornecido. O número de valores na lista depende do `ComparisonOperator` que está sendo usado.

  Para o tipo Number, as comparações de valor são numéricas.

  As comparações de valor String para "maior que", "igual a" ou "menor que" são baseadas em Unicode com codificação UTF-8 binária. Por exemplo, `a` é maior que `A`, e `a` é maior que `B`.

  Para o tipo Binary, o DynamoDB trata cada byte de dados binários como não assinados ao comparar valores binários.
- `ComparisonOperator`: um comparador para avaliar atributos na `AttributeValueList`. Ao executar a comparação, o DynamoDB usa leituras fortemente consistentes.

  Os seguintes operadores de comparação estão disponíveis:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

  Veja a seguir as descrições de cada operador de comparação.

  - `EQ`: igual. `EQ` é aceito por todos os tipos de dados, incluindo listas e mapas.

    `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number, Binary, String Set, Number Set ou Binary Set. Se um item contém um elemento

`AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não é igual a `{"NS":["6", "2", "1"]}`.

- `NE`: não é igual. `NE` é aceito por todos os tipos de dados, incluindo listas e mapas.

  `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number, Binary, String Set, Number Set ou Binary Set. Se um item contém um `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não é igual a `{"NS":["6", "2", "1"]}`.

- `LE` : Menor ou igual a.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

- `LT` : Menor que.

  `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

- `GE` : Maior ou igual a.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

- `GT` : Maior que.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

- `NOT_NULL`: o atributo existe. `NOT_NULL` é aceito por todos os tipos de dados, incluindo listas e mapas.

  > **Note**
  >
  > Este operador testa a existência de um atributo, não o tipo de dados. Se o tipo de dados do atributo "a" for nulo e você avaliá-lo usando `NOT_NULL`, o resultado será um Boolean `true`. Isso acontece porque o atributo "a" existe; o tipo de dados não é relevante para o operador de comparação `NOT_NULL`.

- `NULL`: o atributo não existe. `NULL` é aceito por todos os tipos de dados, incluindo listas e mapas.

  > **Note**
  >
  > Este operador testa a não existência de um atributo, e não seu tipo de dados. Se o tipo de dados do atributo "a" for nulo e você avaliá-lo usando `NULL`, o resultado será um Boolean `false`. Isso acontece porque o atributo "a" existe; o tipo de dados não é relevante para o operador de comparação `NULL`.

- `CONTAINS` : verifica uma subsequência ou valor em um conjunto.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se o atributo de destino da comparação for do tipo String, o operador procurará uma substring correspondente. Se o atributo de destino da comparação for do tipo Binary, o operador procurará uma subsequência do destino que corresponda à entrada. Se o atributo de destino da comparação for um conjunto ("SS", "NS" ou "BS"), o operador será avaliado como verdadeiro, se ele encontrar uma correspondência exata com qualquer membro do conjunto.

  CONTAINS tem suporte em listas: ao avaliar "a `CONTAINS` b", " a" pode ser uma lista. No entanto, "b" não pode ser um conjunto, um mapa ou uma lista.

- `NOT_CONTAINS` : verifica a ausência de uma subsequência ou a ausência de um valor em um conjunto.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se o atributo de destino da comparação for String, o operador verificará a ausência de uma substring correspondente. Se o atributo de destino da comparação for Binary, o operador verificará a ausência de uma subsequência do destino

que corresponda à entrada. Se o atributo de destino da comparação for um conjunto ("SS", "NS" ou "BS"), o operador será avaliado como verdadeiro se ele `does not` encontrar uma correspondência exata com qualquer membro do conjunto.

NOT_CONTAINS tem suporte em listas: ao avaliar "`a NOT CONTAINS b`", " a" pode ser uma lista. No entanto, "b" não pode ser um conjunto, um mapa ou uma lista.

- `BEGINS_WITH` : procura um prefixo.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String ou Binary (não um tipo Number ou Set). O atributo de destino da comparação deve ser do tipo String ou Binary (e não Number ou um tipo de conjunto).

- `IN` : procura elementos correspondentes dentro de dois conjuntos.

  `AttributeValueList` pode conter um ou mais elementos `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Esses atributos são comparados com um atributo do tipo Set existente de um item. Se quaisquer elementos do conjunto de entradas estiverem presentes no atributo do item, a expressão será avaliada como verdadeira.

- `BETWEEN` : maior ou igual ao primeiro valor e menor ou igual ao segundo valor.

  `AttributeValueList` deve conter dois elementos `AttributeValue` do mesmo tipo, seja String, Number ou Binary (não um tipo Set). Um atributo de destino corresponderá se o valor de destino for maior que ou igual ao primeiro elemento e menor que ou igual ao segundo elemento. Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não se compara a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`

Os parâmetros a seguir podem ser usados em vez de `AttributeValueList` e `ComparisonOperator`:

- `Value` - um valor para o DynamoDB comparar com um atributo.
- `Exists` - um valor Booleano que faz com que o DynamoDB avalie o valor antes de tentar a operação condicional:
  - Se `Exists` for `true`, o DynamoDB verificará se esse valor de atributo já existe na tabela. Se ele for encontrado, a condição será avaliada como verdadeira; caso contrário, a condição será avaliada como falsa.

- Se `Exists` for `false`, o DynamoDB assumirá que o valor do atributo `not` existe na tabela. Se, na verdade, o valor não existir, a suposição será válida e a condição será avaliada como verdadeira. Se o valor for encontrado, apesar da suposição de que ele não existe, a condição será avaliada como falsa.

Observe que o valor padrão para `Exists` é `true`.

Os parâmetros `Value` e `Exists` são incompatíveis com `AttributeValueList` e `ComparisonOperator`. Observe que se você usar os dois conjuntos de parâmetros de uma só vez, o DynamoDB retornará uma exceção `ValidationException`.

> **Note**
>
> Este parâmetro não tem suporte a atributos do tipo List ou Map.

Use ConditionExpression em vez disso: exemplo

Suponha que você quisesse modificar um item na tabela Music, mas somente se uma determinada condição fosse verdadeira. Você poderia usar uma solicitação `UpdateItem` com um parâmetro `Expected`, como neste exemplo da AWS CLI:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --attribute-updates '{
        "Price": {
            "Action": "PUT",
            "Value": {"N":"1.98"}
        }
    }' \
    --expected '{
        "Price": {
            "ComparisonOperator": "LE",
            "AttributeValueList": [ {"N":"2.00"} ]
        }
    }'
```

Você pode usar `ConditionExpression` em vez disso:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --update-expression 'SET Price = :p1' \
    --condition-expression 'Price <= :p2' \
    --expression-attribute-values '{
        ":p1": {"N":"1.98"},
        ":p2": {"N":"2.00"}
    }'
```

KeyConditions (herdado)

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB.
> Para obter informações específicas sobre o novo parâmetro que substitui este, Use KeyConditionExpression em vez disso..

O parâmetro condicional herdado `KeyConditions` contém critérios de seleção para uma operação `Query`. Para consultar uma tabela, você pode ter condições apenas em atributos de chave primária da tabela. Você deve fornecer o nome e o valor da chave de partição como uma condição `EQ`. Opcionalmente, você pode fornecer uma segunda condição, referente à chave de classificação.

> **Note**
>
> Se você não fornecer uma condição de chave de classificação, todos os itens que correspondem à chave de partição serão recuperados. Se `FilterExpression` ou `QueryFilter` estiver presente, isso será aplicado depois que os itens forem recuperados.

Para consultar um índice, você pode ter condições apenas em atributos da chave do índice. Você deve fornecer o nome e o valor da chave de partição do índice como uma condição `EQ`.

Opcionalmente, você pode fornecer uma segunda condição, referente à chave de classificação do índice.

Cada elemento `KeyConditions` consiste em um nome de atributo a ser comparado, junto com o seguinte:

- `AttributeValueList`: um ou mais valores para avaliar em relação ao atributo fornecido. O número de valores na lista depende do `ComparisonOperator` que está sendo usado.

  Para o tipo Number, as comparações de valor são numéricas.

  As comparações de valor String para "maior que", "igual a" ou "menor que" são baseadas em Unicode com codificação UTF-8 binária. Por exemplo, `a` é maior que `A`, e `a` é maior que `B`.

  Para Binary, o DynamoDB trata cada byte de dados binários como não assinados ao comparar valores binários.
- `ComparisonOperator`: um comparador para avaliar atributos. Por exemplo: é igual a, maior que e, menor que.

  Para `KeyConditions`, somente os seguintes operadores de comparação têm suporte:

  `EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN`

  Veja a seguir as descrições desses operadores de comparação.

  - `EQ` : Igual.

    `AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do especificado na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não é igual a `{"NS":["6", "2", "1"]}`.
  - `LE` : Menor ou igual a.

    `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.
  - `LT` : Menor que.

`AttributeValueList` pode conter apenas um `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

- `GE` : Maior ou igual a.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

- `GT` : Maior que.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String, Number ou Binary (não um tipo Set). Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não é igual a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

- `BEGINS_WITH` : procura um prefixo.

  `AttributeValueList` pode conter apenas um elemento `AttributeValue` do tipo String ou Binary (não um tipo Number ou Set). O atributo de destino da comparação deve ser do tipo String ou Binary (e não Number ou um tipo de conjunto).

- `BETWEEN` : maior ou igual ao primeiro valor e menor ou igual ao segundo valor.

  `AttributeValueList` deve conter dois elementos `AttributeValue` do mesmo tipo, seja String, Number ou Binary (não um tipo Set). Um atributo de destino corresponderá se o valor de destino for maior que ou igual ao primeiro elemento e menor que ou igual ao segundo elemento. Se um item contém um elemento `AttributeValue` de um tipo diferente do fornecido na solicitação, os valores não coincidem. Por exemplo, `{"S":"6"}` não se compara a `{"N":"6"}`. Além disso, `{"N":"6"}` não se compara a `{"NS":["6", "2", "1"]}`.

Use KeyConditionExpression em vez disso: exemplo

Suponha que você quisesse recuperar vários itens com a mesma chave de partição da tabela Music. Você poderia usar uma solicitação `Query` com um parâmetro `KeyConditions`, como neste exemplo da AWS CLI:

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist":{
            "ComparisonOperator":"EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        },
        "SongTitle":{
            "ComparisonOperator":"BETWEEN",
            "AttributeValueList": [ {"S": "A"}, {"S": "M"} ]
        }
    }'
```

Você pode usar `KeyConditionExpression` em vez disso:

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a AND SongTitle BETWEEN :t1 AND :t2' \
    --expression-attribute-values '{
        ":a": {"S": "No One You Know"},
        ":t1": {"S": "A"},
        ":t2": {"S": "M"}
    }'
```

QueryFilter (herdado)

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB. Para obter informações específicas sobre o novo parâmetro que substitui este, Use FilterExpression em vez disso..

Em uma operação `Query`, parâmetro condicional herdado `QueryFilter` é uma condição que avalia os resultados da consulta depois que os itens são lidos e retorna apenas os valores desejados.

Este parâmetro não tem suporte a atributos do tipo List ou Map.

> **Note**
>
> Um `QueryFilter` é aplicado depois que os itens já foram lidos, o processo de filtragem não consome unidades de capacidade de leitura adicionais.

Se você fornecer mais de uma condição no mapa `QueryFilter`, por padrão, todas as condições deverão ser avaliadas como verdadeiras. Em outras palavras, as condições são combinadas usando o operador `AND`. (Em vez disso, você pode usar o parâmetro ConditionalOperator (herdado) processar as condições com o operador OR. Se fizer isso, pelo menos uma das condições deverá ser avaliada como true, em vez de todas elas.)

Observe que `QueryFilter` não permite atributos de chave. Você não pode definir uma condição de filtro em uma chave de partição ou uma chave de classificação.

Cada elemento `QueryFilter` consiste em um nome de atributo a ser comparado, junto com o seguinte:

- `AttributeValueList`: um ou mais valores para avaliar em relação ao atributo fornecido. O número de valores na lista depende do operador especificado em `ComparisonOperator`.

  Para o tipo Number, as comparações de valor são numéricas.

  As comparações de valor String para "maior que", "igual a" ou "menor que" são baseadas em codificação UTF-8 binária. Por exemplo, `a` é maior que `A`, e `a` é maior que `B`.

  Para o tipo Binary, o DynamoDB trata cada byte de dados binários como não assinados ao comparar valores binários.

  Para obter mais informações sobre como especificar tipos de dados em JSON, consulte API de baixo nível do DynamoDB.
- `ComparisonOperator`: um comparador para avaliar atributos. Por exemplo: é igual a, maior que e, menor que.

  Os seguintes operadores de comparação estão disponíveis:

```
EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
BEGINS_WITH | IN | BETWEEN
```

Use FilterExpression em vez disso: exemplo

Suponha que você quisesse consultar a tabela Music e aplicar uma condição aos itens correspondentes. Você poderia usar uma solicitação `Query` com um parâmetro `QueryFilter`, como neste exemplo da AWS CLI:

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        }
    }' \
    --query-filter '{
        "Price": {
            "ComparisonOperator": "GT",
            "AttributeValueList": [ {"N": "1.00"} ]
        }
    }'
```

Você pode usar `FilterExpression` em vez disso:

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a' \
    --filter-expression 'Price > :p' \
    --expression-attribute-values '{
        ":p": {"N":"1.00"},
        ":a": {"S":"No One You Know"}
    }'
```

ScanFilter (herdado)

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB. Para obter informações específicas sobre o novo parâmetro que substitui este, Use FilterExpression em vez disso..

Em uma operação `Scan`, o parâmetro condicional herdado `ScanFilter` é uma condição que avalia os resultados da verificação e retorna apenas os valores desejados.

> **Note**
>
> Este parâmetro não tem suporte a atributos do tipo List ou Map.

Se você especificar mais de uma condição no mapa `ScanFilter`, por padrão, todas as condições deverão ser avaliadas como verdadeiras. Em outras palavras, as condições são processadas com o operador AND. (Em vez disso, você pode usar o parâmetro ConditionalOperator (herdado) processar as condições com o operador OR. Se fizer isso, pelo menos uma das condições deverá ser avaliada como true, em vez de todas elas.)

Cada elemento `ScanFilter` consiste em um nome de atributo a ser comparado, junto com o seguinte:

- `AttributeValueList`: um ou mais valores para avaliar em relação ao atributo fornecido. O número de valores na lista depende do operador especificado em `ComparisonOperator`.

  Para o tipo Number, as comparações de valor são numéricas.

  As comparações de valor String para "maior que", "igual a" ou "menor que" são baseadas em codificação UTF-8 binária. Por exemplo, `a` é maior que `A`, e `a` é maior que `B`.

  Para Binary, o DynamoDB trata cada byte de dados binários como não assinados ao comparar valores binários.

Para obter mais informações sobre como especificar tipos de dados em JSON, consulte API de baixo nível do DynamoDB.

- `ComparisonOperator`: um comparador para avaliar atributos. Por exemplo: é igual a, maior que e, menor que.

  Os seguintes operadores de comparação estão disponíveis:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

Use FilterExpression em vez disso: exemplo

Suponha que você quisesse examinar a tabela Music e aplicar uma condição aos itens correspondentes. Você poderia usar uma solicitação `Scan` com um parâmetro `ScanFilter`, como neste exemplo da AWS CLI:

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

Você pode usar `FilterExpression` em vez disso:

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

Criar condições com parâmetros herdados

> **Note**
>
> Sugerimos que você use os novos parâmetros de expressão, em vez desses parâmetros herdados. Para ter mais informações, consulte Usar expressões no DynamoDB.

A seção a seguir descreve como criar condições a serem usadas com parâmetros herdados, como `Expected`, `QueryFilter` e `ScanFilter`.

> **Note**
>
> Em vez disso, as novas aplicações devem usar parâmetros de expressão. Para ter mais informações, consulte Usar expressões no DynamoDB.

Condições simples

Com valores de atributo, você pode escrever condições para comparações com atributos da tabela. Uma condição sempre é avaliada como verdadeira ou falsa, e consiste em:

- `ComparisonOperator` - maior que, menor que, igual a e assim por diante.
- `AttributeValueList` (opcional): valores de atributo para comparar. Dependendo do `ComparisonOperator` que está sendo usado, `AttributeValueList` pode conter um, dois ou mais valores ou pode não estar presente.

As seções a seguir descrevem os vários operadores de comparação, juntamente com exemplos de como usá-los em condições.

Operadores de comparação sem valores de atributo

- `NOT_NULL` - verdadeiro se um atributo existir.
- `NULL` - verdadeiro se um atributo não existir.

Use esses operadores para verificar se um atributo existe ou não. Como não há valor de comparação, não especifique `AttributeValueList`.

Exemplo

A expressão a seguir é avaliada como verdadeira se o atributo Dimensions existir.

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

Operadores de comparação com um valor de atributo

- `EQ` - verdadeiro se um atributo for igual a um valor.

  `AttributeValueList` pode conter apenas um valor do tipo String, Number, Binary, String Set, Number Set ou Binary Set. Se um item contém um valor de um tipo diferente do especificado na solicitação, os valores não coincidem. Por exemplo, a string `"3"` não é igual ao número `3`. Além disso, o número `3` não é igual ao conjunto de números `[3, 2, 1]`.
- `NE` - verdadeiro se um atributo não for igual a um valor.

  `AttributeValueList` pode conter apenas um valor do tipo String, Number, Binary, String Set, Number Set ou Binary Set. Se um item contém um valor de um tipo diferente do especificado na solicitação, os valores não coincidem.
- `LE` - verdadeiro se um atributo for menor ou igual a um valor.

  `AttributeValueList` pode conter apenas um valor do tipo String, Number ou Binary (não um tipo Set). Se um item contém um `AttributeValue` de um valor diferente do especificado na solicitação, os valores não coincidem.
- `LT` - verdadeiro se um atributo for menor que um valor.

  `AttributeValueList` pode conter apenas um valor do tipo String, Number ou Binary (não um tipo Set). Se um item contém um valor de um tipo diferente do especificado na solicitação, os valores não coincidem.
- `GE` - verdadeiro se um atributo for maior ou igual a um valor.

  `AttributeValueList` pode conter apenas um valor do tipo String, Number ou Binary (não um tipo Set). Se um item contém um valor de um tipo diferente do especificado na solicitação, os valores não coincidem.
- `GT` - verdadeiro se um atributo é maior que um valor.

`AttributeValueList` pode conter apenas um valor do tipo String, Number ou Binary (não um tipo Set). Se um item contém um valor de um tipo diferente do especificado na solicitação, os valores não coincidem.

- `CONTAINS` - verdadeiro se um valor estiver presente em um conjunto ou se um valor contiver outro.

  `AttributeValueList` pode conter apenas um valor do tipo String, Number ou Binary (não um tipo Set). Se o atributo de destino da comparação for String, o operador procurará uma substring correspondente. Se o atributo de destino da comparação for Binary, o operador procurará uma subsequência do destino que corresponda à entrada. Se o atributo de destino da comparação for um conjunto, o operador será avaliado como verdadeiro caso encontre uma correspondência exata com qualquer membro do conjunto.
- `NOT_CONTAINS` - verdadeiro se um valor não estiver presente em um conjunto, ou se um valor não contiver outro valor.

  `AttributeValueList` pode conter apenas um valor do tipo String, Number ou Binary (não um tipo Set). Se o atributo de destino da comparação for String, o operador verificará a ausência de uma substring correspondente. Se o atributo de destino da comparação for Binary, o operador verificará a ausência de uma subsequência do destino que corresponda à entrada. Se o atributo de destino da comparação for um conjunto, o operador será avaliado como verdadeiro se ele não encontrar uma correspondência exata com qualquer membro do conjunto.
- `BEGINS_WITH`: verdadeiro se os primeiros caracteres de um atributo corresponderem ao valor fornecido. Não use este operador para comparar números.

  `AttributeValueList` pode conter apenas um valor do tipo String ou Binary (não um tipo Number ou Set). O atributo de destino da comparação deve ser String ou Binary (não Number ou um conjunto).

Use esses operadores para comparar um atributo com um valor. Você deve especificar um `AttributeValueList` consistindo em um único valor. Para a maioria dos operadores, esse valor deve ser escalar. No entanto, os operadores `EQ` e `NE` também dão suporte a conjuntos.

Exemplos

As expressões a seguir serão avaliadas como verdadeiras se:

- O preço de um produto for maior que 100.

```
...
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
    }
...
```

- A categoria de um produto começar com "Bo".

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
        AttributeValueList: [ {"S":"Bo"} ]
    }
...
```

- Um produto estiver disponível em vermelho, verde ou preto:

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> Ao comparar tipos de dados Set, a ordem dos elementos não importa. O DynamoDB retornará apenas os itens com o mesmo conjunto de valores, independentemente da ordem em que você especificá-los em sua solicitação.

Operadores de comparação com dois valores de atributo

- `BETWEEN` - verdadeiro se um valor for entre um limite inferior e um limite superior, inclusive endpoints.

`AttributeValueList` deve conter dois elementos do mesmo tipo, seja String, Number ou Binary (não Set). Um atributo de destino corresponderá se o valor de destino for maior que ou igual ao primeiro elemento e menor que ou igual ao segundo elemento. Se um item contém um valor de um tipo diferente do especificado na solicitação, os valores não coincidem.

Use este operador para determinar se um valor de atributo está dentro de um intervalo. A `AttributeValueList` deve conter dois elementos escalares do mesmo tipo – String, Number ou Binary.

Exemplo

A expressão a seguir será avaliada como verdadeira, se o preço de um produto estiver entre 100 e 200.

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

Operadores de comparação com n valores de atributo

- `IN`: verdadeiro se um valor for igual a qualquer um dos valores em uma lista enumerada. Somente valores escalares são aceitos na lista, mas não conjuntos. O atributo de destino deve ser do mesmo tipo e o valor exato para corresponder.

  `AttributeValueList` pode conter um ou mais elementos do tipo String, Number ou Binary (não um tipo Set). Esses atributos são comparados com um atributo do tipo que não é Set existente de um item. Se quaisquer elementos do conjunto de entradas estiverem presentes no atributo do item, a expressão será avaliada como verdadeira.

  `AttributeValueList` pode conter um ou mais valores do tipo String, Number ou Binary (não um tipo Set). O atributo de destino da comparação deve ser do mesmo tipo e ter o valor exato para corresponder. Um atributo String nunca corresponde a um String set.

Use este operador para determinar se o valor fornecido está em uma lista enumerada. Você pode especificar qualquer número de valores escalares em `AttributeValueList`, mas todos eles devem ser do mesmo tipo de dados.

Exemplo

A expressão a seguir será avaliada como verdadeira se o valor para Id for 201, 203 ou 205.

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

Uso de várias condições

O DynamoDB permite combinar várias condições para formar expressões complexas. Isso é feito ao fornecer pelo menos duas expressões, com um ConditionalOperator (herdado) opcional.

Por padrão, quando você especifica mais de uma condição, todas as condições devem ser avaliadas como verdadeiras para que a expressão inteira também seja avaliada assim. Em outras palavras, uma operação AND implícita acontece.

Exemplo

A expressão a seguir será avaliada como verdadeira se um produto for um livro que tem 600 páginas, pelo menos. Ambas as condições devem ser avaliadas como verdadeiras, já que elas são implicitamente ANDed (associadas).

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600} ]
    }
...
```

Você pode usar ConditionalOperator (herdado) para esclarecer que uma operação AND acontecerá. O exemplo a seguir se comporta da mesma forma que o anterior.

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600"} ]
    }
...
```

Você também pode definir `ConditionalOperator` como OR, o que significa que pelo menos uma das condições deve ser avaliada como verdadeira.

Exemplo

A expressão a seguir será avaliada como verdadeira, se um produto for uma mountain bike, se ele for de uma marca específica ou se o preço for maior que 100.

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> Em uma expressão complexa, as condições são processadas na ordem, da primeira condição para a última.
> Não é possível usar AND e OR em uma única expressão.

Outros operadores condicionais

Em versões anteriores do DynamoDB, o parâmetro `Expected` se comportava de forma diferente em gravações condicionais. Cada item no mapa `Expected` representava um nome de atributo para o DynamoDB verificar, juntamente com o seguinte:

- `Value` - um valor para comparar com o atributo.
- `Exists` - determina se o valor existe antes de tentar a operação.

As opções `Value` e `Exists` continuam a ter suporte no DynamoDB; no entanto, elas só permitem que você teste uma condição de igualdade ou se um atributo existe. Recomendamos que você use `ComparisonOperator` e `AttributeValueList` em vez disso, pois essas opções permitem que você crie uma gama muito maior de condições.

Example

Uma operação `DeleteItem` pode verificar se um livro não está mais em publicação e excluí-lo somente se essa condição for verdadeira. Veja um exemplo da AWS CLI que usa uma condição herdada:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

O exemplo a seguir executa a mesma ação, mas não usa uma condição herdada:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

Uma operação `PutItem` pode proteger contra a substituição de um item existente com os mesmos atributos da chave primária. Veja um exemplo da que usa uma condição herdada:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

O exemplo a seguir executa a mesma ação, mas não usa uma condição herdada:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> Para condições no mapa `Expected`, não use as opções herdadas `Value` e `Exists` com `ComparisonOperator` e `AttributeValueList`. Se você fizer isso, sua gravação condicional falhará.