Trabalho da disciplina banco de dados II - UERJ 2020/2

Integrantes

- Dennis Ribeiro Paiva - 201610050611
- Igor Sousa Silva - 201610053411
- Paulo Victor Coelho - 201610049711
- Vinicius Sathler - 201610051611

Introdução

O Amazon DynamoDB (ADDB) é um banco de dados de Chave-Valor (Key-Value) e de documentos. Se comparado com um banco sql convencional, o ADDB não possui um esquema bem definido, cada tabela possui uma chave primária única, mas não existe nenhuma restrição para todos os outro atributos do elemento, mesmo entre elementos de uma mesma tabela.

O ADDB oferece um serviço de banco de dados através da Amazon Web Service (AWS) focado em aplicações on-line, otimizando latência de conexão e oferecendo um serviço altamente escalonável através de clusters dinâmicos de máquinas responsáveis pelo armazenamento de dados.

Arquitetura

Componentes

No ADDB cada tabela funciona como uma coleção de elementos, e cada elemento funciona como uma coleção de atributos. com exceção da chave primária, os elementos não possuem um conjunto de atributos fixos, ou seja, cada item de uma tabela pode possuir atributos distintos. Também é possível criar atriutos aninhados com até 32 níveis.

People

Exemplo da tabela "People" com 3 elementos

Chaves Primárias

O ADDB possui dois tipos de chaves primárias, Partition Key e Sort Key. Cada elemento da tabela possui uma chave primária (partition key) única. É possível que uma tabela possua chave primária composta de 2 atributos (Partition Key e sort key).

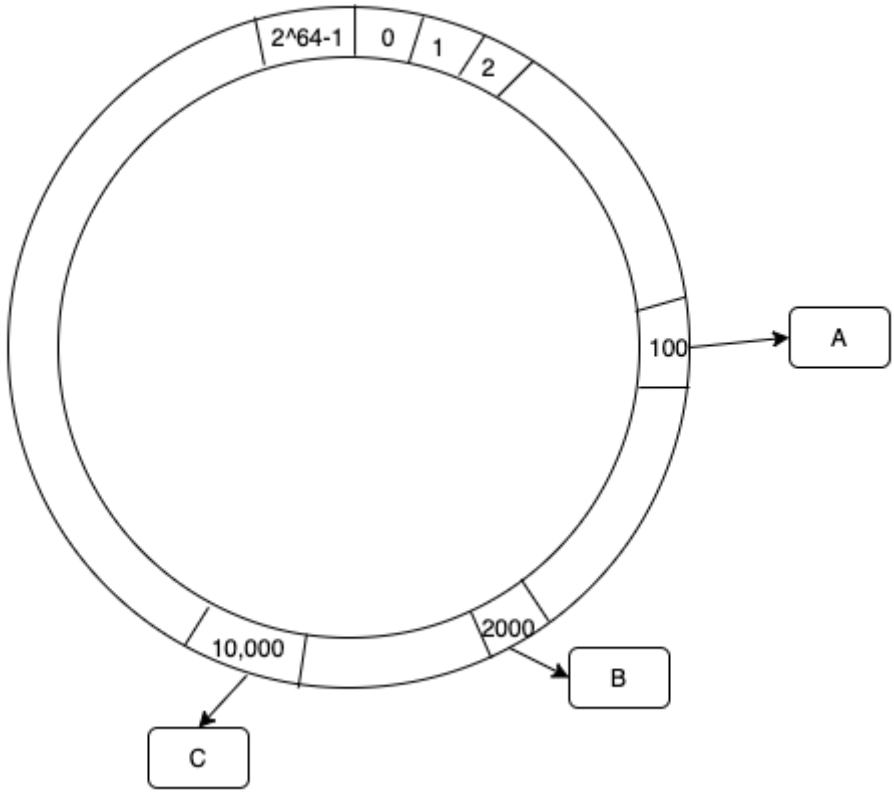
Music

Exemplo da tabela "Music" com chaves primárias compostas

O ADDB armazena todos os seus dados em "blocos" de memória chamados de "partitions" ou partições. O endereçamento desses dados funciona como um hash-map, onde cada elemento terá uma partição alvo e essas partições são distribuídas e replicadas em diversos servidores da AWS da região configurada.

Partição

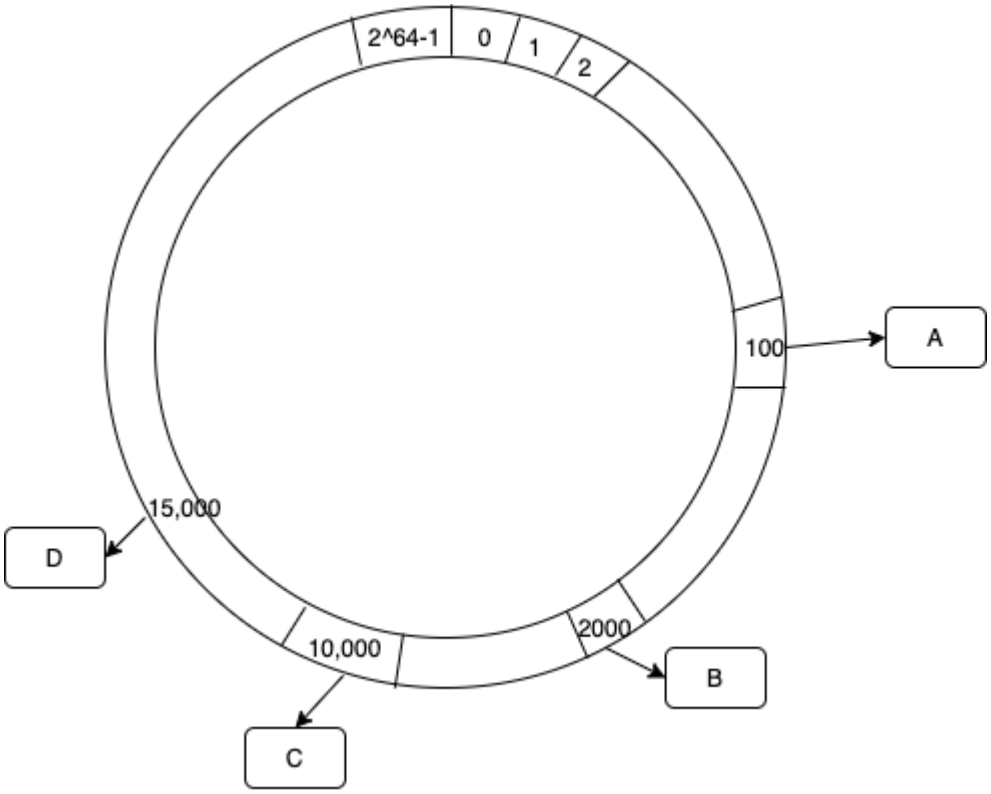
A AWS aloca máquinas para o banco de dados de acordo com a demanda e cada máquina que participa do "cluster" recebe uma "tag" com um valor inteiro no intervalo [0,2^64). Quando ocorre uma requisição para inserção de dados no banco, a chave do elemento passa por uma função hash que retorna um inteiro no intervalo anterior, o dado é então guardado na primeira máquina encontrada, a busca pela máquina é realizada em um esquema de "relógio" de acordo com a imagem seguinte.



Busca da máquina-alvo

No exemplo da imagem acima, se a função hash retornar 100000, o dado será armazenado na máquina A.

Caso uma máquina fique sobrecarregada de dados a AWS aloca máquinas adicionais para o banco. A nova máquina armazenará todos os dados do seu novo intervalo.

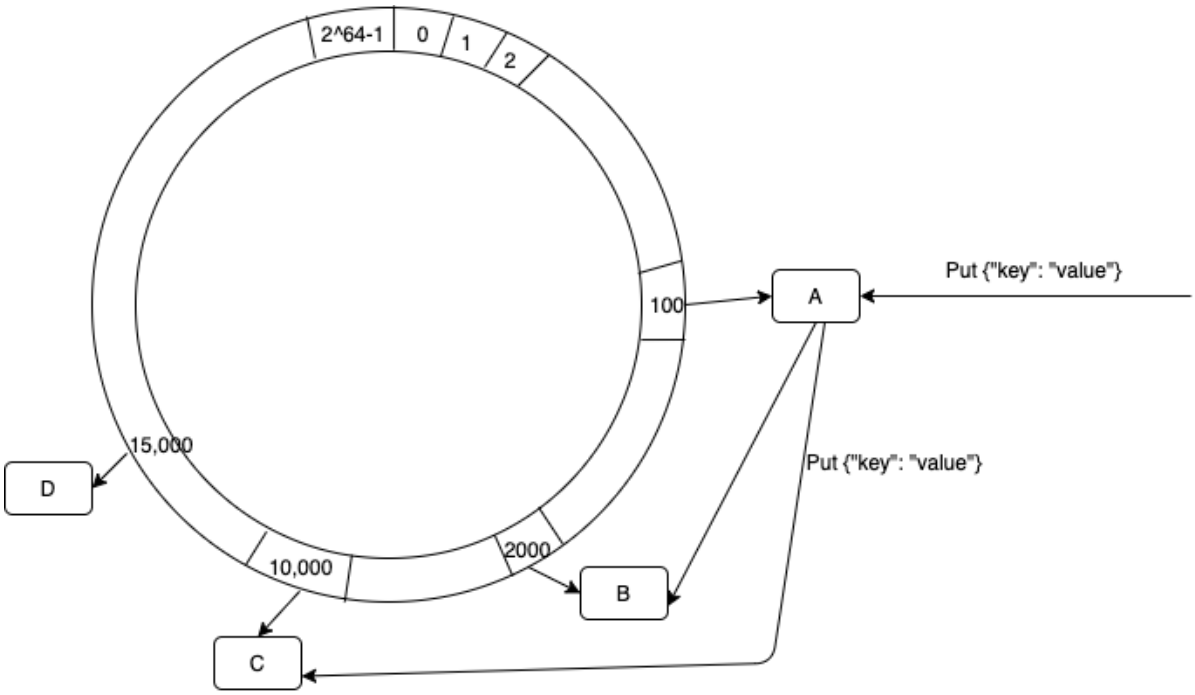


Adição da máquina "D" ao cluster

No exemplo acima todos os dados entre os endereços 10000 e 15000 serão transferidos para a máquina D. Caso uma máquina fique subutilizada, o banco pode remover uma máquina e transferir seus dados para a próxima máquina.

Replicação

Para evitar perda de dados caso haja um problema em alguma máquina, quando ocorre uma requisição de inserção no banco, a AWS replica a inserção da máquina-alvo nas N-1 próximas máquinas.



Replicação da requisição nas N-1 próximas máquinas (N=3)

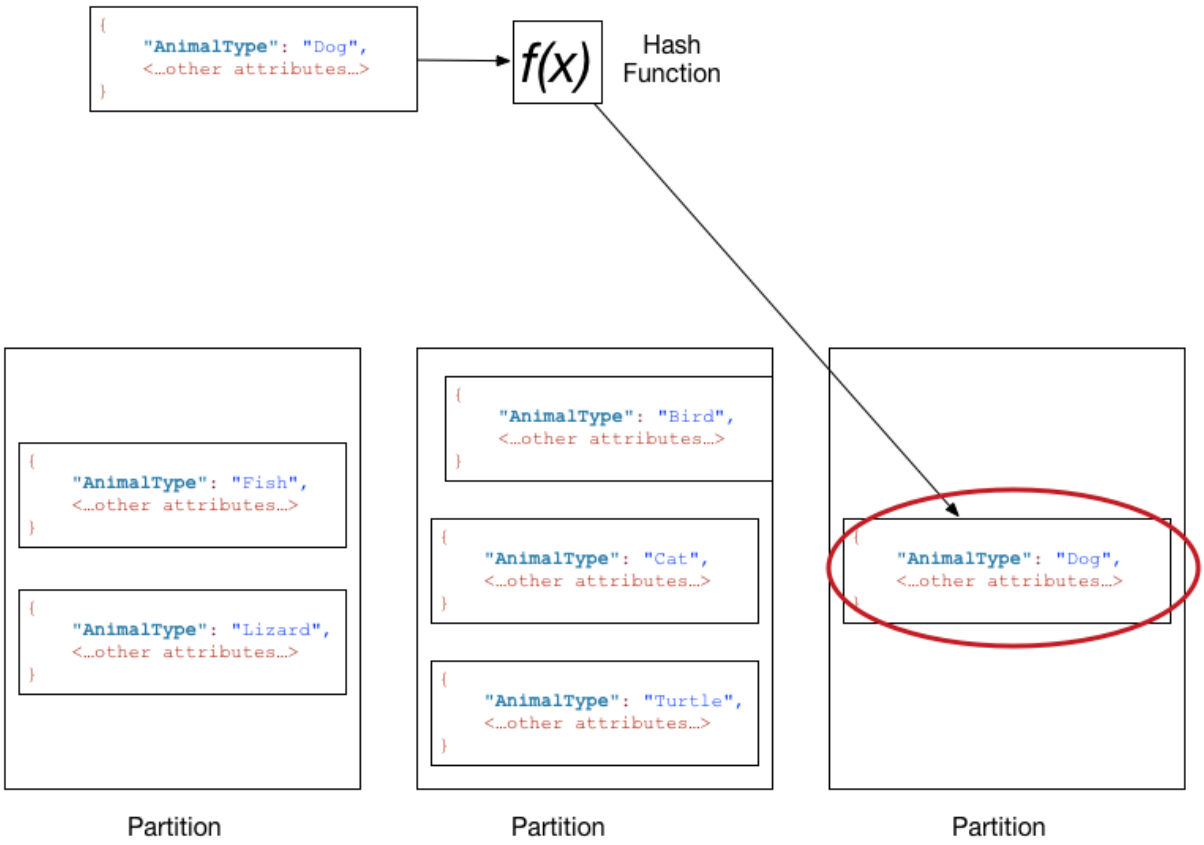
Para evitar perda de desempenho na operação de escrita, o banco enviará a resposta de conclusão da escrita caso uma quantidade W de máquinas termine a operação.

Um problema desse método é que uma requisição de leitura pode tentar recuperar dados de uma máquina que não tenha completado o processo de escrita, retornando dados inválidos ou desatualizados. Para resolver este problema o banco lê R cópias do dado entre as máquinas do cluster, retornando o valor mais recente.

Para garantir que a operação seja um sucesso, o valor de R+W deve ser superior ao valor de N, dessa maneira garante-se que o dado mais recente será recuperado.

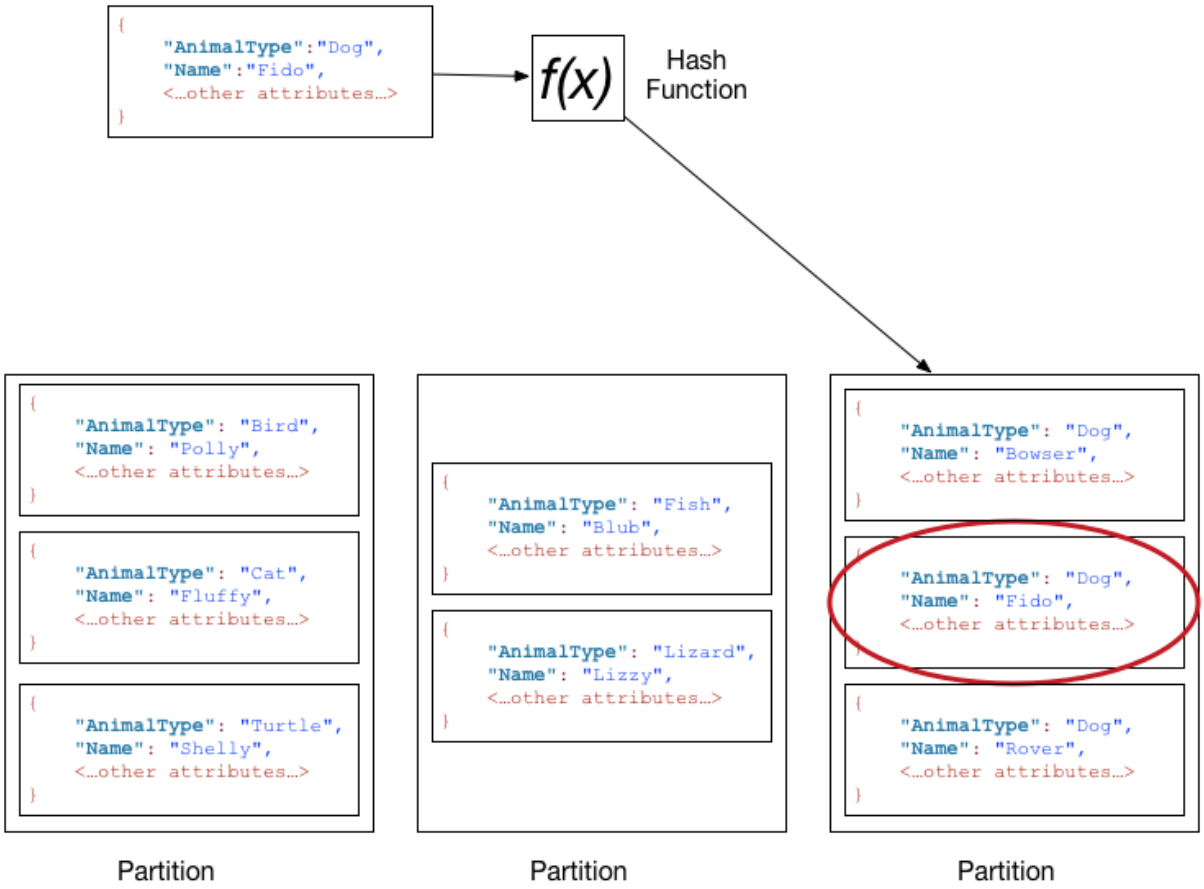
Otimização

Como explicado anteriormente, todo elemento de uma tabela possui uma chave primária chamada partition key, e essa chave passa por uma função hash para definir em qual partição o elemento será guardado.



Exemplo de alocação de um elemento em uma partição.

Se o elemento possuir uma chave secundária, serão alocados para a mesma partição todos os elementos com a mesma partition key e estes estarão ordenados pela sua sort key.



Exemplo de alocação de um elemento em uma partição e ordenado pela sua sort key.

Porém, caso você queira ordenar de maneira diferente e restringir os atributos projetados, pode-se criar uma *secondary index*, que é diferente do conceito de index de um banco relacional. Quando você cria a *secondary index* deve-se criar uma *partition key* (caso seja uma global, que será explicado a seguir) e uma sort key e defini-las. Após a criação, pode-se fazer uma query ou um scan igual como seria feito em uma tabela. ADDB não tem um otimizador de queries, então o *secondary index* é usado apenas quando se faz uma *query* ou um *scan* nele mesmo. Quando é gerado uma *secondary index*, uma outra tabela é criada com as chaves definidas, além da *primary key* da tabela original, ou seja, no final, mesmo tendo definido apenas duas chaves, a nova tabela terá 3.

No Dynamo pode-se usar dois tipos de indexes:

- *secondary indexes* globais: a *primary key* do index deve ter dois atributos da tabela (qualquer atributo). Para projetar posteriormente, apenas será mostrado os atributos definidos no parâmetro `Projection` no momento da criação do índex. Esse parâmetro será explicado.
- *secondary indexes* locais: a *partition key* do index deve ser a mesma *partition key* da tabela. Entretanto, a *sort key* pode ser qualquer atributo da tabela. Além disso, podemos projetar qualquer atributo a posteriori da criação do índex, mesmo que não tenha sido definido no `Projection`.

Suas diferenças:

Características	**Global Secondary Index**	**Local Secondary Index**
Key Schema	A chave primária pode ser simples (*partition key*) ou composta (*partition* e *sort key*).	A chave primária deve ser composta (*partition key* e *sort key*).
Key Attributes	A *partition* e a *sort key* (se definida) podem ser qualquer atributo da tabela base do tipo String, Número ou Binário.	A *partition key* do índice é a mesma da tabela base. A *sort key* pode ser qualquer atributo da tabela base do tipo String, Número ou Binário.
Restrições de tamanho por valores de *Partition Key*	Não há restrições	Para cada valor de *partition key*, o tamanho total de todos os itens indexados deve ser de <10 GB.
Operações Online de índices	Podem ser criados ao mesmo tempo da criação de uma tabela. Pode ser adicionado um novo índice a uma tabela existente ou excluir um índice existente.	São criados ao mesmo tempo em que se cria uma tabela. Não é possível ser adicionado a uma tabela existente nem excluir índices secundários locais existentes.
Queries e *Partitions*	Permite a consulta da tabela inteira, em todas as partições.	Permite consultar uma única partição, conforme especificado pelo valor da *partition key* na consulta.
Provisioned Throughput Consumption	Tem suas próprias configurações de *provisioned throughput* para operações de leitura e gravação. *Queries* ou *scans* consomem unidades de capacidade do índice, e não da tabela base. O mesmo vale para atualizações de índice secundário global devido a gravações de tabelas.	*Queries* ou *scans* consomem unidades de capacidade de leitura da tabela base. Quando você escreve em uma tabela, seus índices secundários locais também são atualizados. Essas atualizações consomem unidades de capacidade de gravação da tabela base.
Projected Attributes	Após ser definido o *projection*, não pode-se alterar os atributos que serão projetados na *query/scan* do índice.	Pode ser definido atributos a posteriori da criação do índice

Você deve prover os seguintes parâmetros para a `UpdateTable`

- `TableName` - A tabela que o index vai ser associado
- `AttributeDefinitions` - os tipos de dados para a *key schema*
- `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. Pode ser INCLUDE e definir certos atributos ou KEYS_ONLY para ser apenas chaves.
- `ProvisionedThroughput` - o número de leituras e gravações por segundo que você precisa para este índice. Isso é separado das configurações de throughput provisionado da tabela.

Exemplo de criação de um index

Pode-se adicionar uma index global em uma tabela existente, usando a ação 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
        }
      }
    }
  ]
}


Retirado da documentação do DynamoDB
```

Consultas

A operação de consulta no Amazon DynamoDB encontra itens com base em valores de chave primária.

Você deve fornecer o nome do atributo da chave primária e um único valor para esse atributo. A Query retorna todos os itens com esse valor de chave primária. Opcionalmente, você pode fornecer um atributo de sort key (chave de ordenação) e usar um operador de comparação para refinar os resultados da pesquisa.

KeyConditionExpression

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  file://values.json
```

Os argumentos para --expression-attribute-values estão armazenados no arquivo values.json abaixo.

```
{
   ":name":{"S":"Acme Band"}
}
```

Para especificar o critério de busca, é usado a `KeyConditionExpression`. Que é uma string que determina os itens a serem lidos da tabela ou índice. (Ex: `ForumName = :name`)

Deve-se especificar o nome e o valor da chave primária como uma condição de igualdade.

Pode-se usar qualquer atributo numa KeyConditionExpression, desde que o primeiro caractere seja [a-z] ou [A-Z].

Para items com uma chave primária entregue, DynamoDB armazena esses itens juntos, em ordem de classificação por valor de sort key. Numa operação de Query, DynamoDB recupera os itens de maneira organizada e então processa os itens usando as condições do `KeyConditionExpression` e qualquer "FilterExpression" que pode ser presente. Só então o resultado da Query é mandado de volta pra o cliente.

Os resultados da Query são sempre classificados pelo valor da sort key. Se o tipo de dados da sort key for número, os resultados serão retornados em ordem numérica. Caso contrário, os resultados são retornados na ordem de bytes UTF-8(alfabética). Por padrão, a ordem de classificação é crescente.

Uma única operação de consulta pode recuperar no máximo 1 MB de dados. Esse limite se aplica antes que qualquer `FilterExpression` seja aplicado aos resultados.

FilterExpression para a Query:

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject = :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

Os argumentos para --expression-attribute-values estao armazenados no arquivo values.json abaixo.

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Se você precisar refinar ainda mais os resultados da Query, poderá fornecer, opcionalmente, uma `FilterExpression`.Ela determina quais itens nos resultados da consulta devem ser retornados para o usuário. Todos os outros resultados são descartados. Exemplo: "#v >= :num"

Ela é aplicada depois que a consulta é terminada, mas antes dos resultados serem retornados. Portanto, uma consulta consome a mesma quantidade de capacidade de leitura, independentemente da presença de uma expressão de filtro.

Uma `FilterExpression` não pode conter chave primária ou atributos de sort key. Você precisa especificar esses atributos na `KeyConditionExpression`, não na expressão de filtro.

A sintaxe de uma `FilterExpression` é idêntica à de uma `KeyConditionExpression`.

Transações

Com as transações da Amazon DynamoDB, você pode agrupar várias ações e submetê-las como uma única operação de tudo-ou-nada com a TransactWriteItems ou TransactGetItems. Ambas com um limite de 25 itens distintos sendo requisitados ou alterados. Caso seja preciso que a transação seja feita ainda que algumas requisições falhem, pode-se usar chamadas de funções depreciadas que a Amazon não recomenda o uso, o BatchWriteItem e o BatchGetItem, onde a primeira tem um limite de 100 alterações e o segundo de 25. Caso seja necessário verificar se a requisição é idempotente. Precisa-se ter um *client token* que funciona por 10 minutos, assim, pode-se verificar se o item dentro da transação de fato foi alterado. Apenas o TransactWriteItems aceita essa funcionalidade. Se qualquer parâmetro for diferente, o dynamoDB retornará um erro. As transações custam 2x mais no consumo da capacidade, ou seja, se for usado 2kb em um item, na verdade, será gasto 4kb.

TransactWriteItems

Você pode adicionar os seguintes tipos de ações a uma transação:

- `Put` - Inicia uma operação PutItem para criar um novo item ou substituir um item antigo por um novo. Aceita `ConditionExpression`
- `Update` - Inicia uma operação UpdateItem para editar os atributos de um item existente ou adicionar um novo item à tabela se ele ainda não existir. Aceita `ConditionExpression`.
- `Delete` - Inicia uma operação DeleteItem para excluir um único item em uma tabela identificada por sua chave primária.

- `ConditionCheck` - Verifica se um item existe ou verifica a condição de atributos específicos do item.

Uma vez concluída uma transação, as mudanças feitas dentro dessa transação são propagadas para *secondary index* global (GSIs), fluxos e backups. Como a propagação não é imediata ou instantânea, se uma tabela for restaurada do backup para um ponto médio de propagação, ela pode conter algumas, mas não todas, as mudanças feitas durante uma transação recente.

Tratamento de erros de escrita

As transações de escrita não são bem sucedidas nas seguintes circunstâncias:

- Quando uma condição em uma das expressões de condição não é satisfeita.
- Quando um erro de validação de transação ocorre porque mais de uma ação na mesma TransactWriteItems operação visa o mesmo item.
- Quando uma solicitação TransactWriteItems entra em conflito com uma operação TransactWriteItems em andamento em um ou mais itens da solicitação TransactWriteItems. Neste caso, a solicitação falha com uma TransactionCanceledException.
- Quando não há capacidade provisionada suficiente para que a transação seja concluída.
- Quando um item se torna muito grande (maior que 400 KB), ou um índice secundário local (LSI) torna-se muito grande, ou um erro de validação similar ocorre devido a mudanças feitas pela transação.
- Quando há um erro do usuário, tal como um formato de dados inválido.

TransactGetItems

- Get - Inicia uma operação GetItem para recuperar um conjunto de atributos para o item com a chave primária dada. Se nenhum item correspondente for encontrado, Get não retorna nenhum dado.

Tratamento de erros de leitura

As transações lidas não têm sucesso sob as seguintes circunstâncias:

- Quando uma solicitação TransactGetItems entra em conflito com uma operação TransactWriteItems em andamento em um ou mais itens da solicitação TransactGetItems. Neste caso, a solicitação falha com uma TransactionCanceledException.
- Quando não há capacidade provisionada suficiente para que a transação seja concluída.
- Quando há um erro do usuário, tal como um formato de dados inválido.

Níveis de Isolamento para Transações DynamoDB

- Serializável
- Read-Commited

Exemplo de TransactWriteItem

AWS cli

```
aws dynamodb transact-write-items
    --transact-items file://transact-items.json
```

JSON de entrada Trasact-items

```
[
  {
    "Put": {
      "TableName": "Music",
      "Item": {
        "Artist": { "S": "Rhapsody of Fire" },
        "SongTitle": { "S": "Dawn of Victory" },
        "Genre": { "S": "Symphonic Power Metal"}
      },
      "ConditionExpression": "attribute_not_exists(Artist)"
    }
  }
]
```

Controle de Concorrência

Optimistic locking

Optimistic locking é uma estratégia que assegura que o item do lado do cliente que se está atualizando (ou excluindo) seja o mesmo que o item na *Amazon DynamoDB*. Se usar esta estratégia, as gravações de seu banco de dados serão protegidas de serem sobrescritas pelas gravações de outros, e vice versa

Com o *Optimistic locking*, cada item tem um atributo que funciona como um número de versão. Se retornar um item de uma tabela, a aplicação grava o número de versão daquele item. Pode-se atualizar o item, mas apenas se o número de versão no lado do servidor não tiver sido alterado. Se houver uma incompatibilidade de versão, significa que alguém modificou este item ateriormente. A tentativa de atualização falha, porque se tem uma versão desatualizada do item. Caso isto aconteça, simplesmente tente de novo recuperando o item e, em seguida, tentando atualizá-lo.

Optimistic locking tem o seguinte impacto nestes métodos do `DynamoDBWrapper`:

- `save`, `put`, `update` — Para um novo item, o `DynamoDBWrapper` atribui um número de versão inicial 1. Caso recupere um item, atualize um ou mais de suas propriedade e tente salvar as alterações, a operação `save`, `put` e `update` terão sucesso apenas se o número de versão no lado do cliente e no lado do servidor forem correspondentes.

- `delete` — o método `delete` necessita de um objeto como parametro e o `DynamoDBMapper` realiza a verificação da versão antes de remover o item. A verificação da versão pode ser desabilitada se

`DynamoDBMapperConfig.SaveBehavior.CLOBBER` for especificado na requisição.

Distributed Locking

Distributed Locking é uma técnica a qual, quando implementada corretamente, garante que dois processos não podem acessar um dado compartilhado ao mesmo tempo. Os processos dependem do protocolo de bloqueio para garantir que apenas um tenha permissão para prosseguir uma vez que se tenha um bloqueio.

Implementações comuns para *distributed locking* geralmente involvem um gerenciador de bloqueios (*lock manager*) com o qual os processos se comunicam. O gerenciador de bloqueios controla o estado dos bloqueios.

A implementação de um *distributed locking* é difícil de se conseguir porque há casos extremos a considerar tais como:

- Condições de corrida
- *Deadlocks*
- *Stale locks*

Uma solução para o *distributed lock* involve gravar o atual detentor do bloqueio (*holder lock*), o processo que está impedindo os demais processos de executarem. Enquanto processos concorrentes tentam ser os próximos detentores do bloqueio. Uma desvantagem desta abordagem é que um processo poderia estar sem sorte enquanto outros seriam muito sortudos. O primeiro processo que não puder se tornar o *holder lock*, aguarda por um certo tempo, e após este tempo ele percebe que outro processo vai tomar a sua frente, impedindo esse processo inicial. Isto pode continuar indefinidamente, levando o processo a sofrer de starvation.

Segurança

Proteção de Dados em DynamoDB

A AmazonDB fornece uma infarestrutura de armazenagem resiliente projetada para armazenamento de dados de missão crítica 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 usuários armazenados em repouso e também os que estão em trânsito entre os clientes e o DynamoDB, e entre o *DynamoDB* e outros recursos da AWS na mesma região da AWS.

Criptografia do DynamoDB em Repouso

A criptografia do *DynamoDB* em repouso fornece uma melhor segurança ao critografar os dados em repouso usando as chaves de criptografia armazenadas no *AWS Key Management Service (AWS KMS)*. Isto ajuda a reduzir a carga operacioanl e a complexidade involvida em proteger dados sensíveis. Também fornece uma camada adicional de proteção de dados ao assegurar os dados em uma tabela criptografada. Ao criar uma nova tabela, pode-se escolher uma das seguintes chaves mestras do cliente (CMK - customer master keys) para criptografar a tabela:

- CMK de posse da AWS - tipo de criptografia padrão
- CMK gerenciada pela AWS - a chave é armazenada na conta do cliente e gerecniada pela AWS KMS

- CMK gerenciada pelo cliente - a chave é armazenada na conta e criada, detetida e gerenciada pelo cliente.

Ao acessar ua tabela criptografada, o *DynamoDB* descriptografa os dadosda tabela transparentemente.

Identity and Access Management

Administradores IAM controlam quem opde ser autenticado (conectado) e autorizado (tem permissão) para usar os recursos da *Amazon DynamoDB*. Pode-se usar a AWS *Identity and Access Management* (IAM) para gerenciar as permissões de acesso e implementar políticas de segurança para ambos *Amazon DynamoDB* and *DynamoDB Accelerator* (DAX).

Controle de Acesso

Pode se ter credenciais válidas para autenticar as requisições, mas a não ser que tenha permissões não se pode criar ou acessar os recursos da *Amazon DynamoDB*. Por exemplo, se faz necessário ter permissões para criar uma tabela na *Amazon DynamoDB*.

Visão Geral do Gerenciamento de Permissões de Acesso aos Recursos da Amazon DynamoDB

Uma conta de administrador pode anexar políticas de permissões às identidades IAM e alguns serviços também permitem que se atribuam políticas de permissões aos recursos. Ao conceder permissões, pode-se escolher quem recebe as permissões, os recursos relacionados às permissões concedidas e as ações específicas que se deseja permitir nesses recursos.

Recursos e operações do DynamoDB

No *DynamoDB* os recursos primários são as tabelas. Ele também suporta tipos de recursos adicionais, *indexes* e *streams*. Contudo, pode-se criar *indexes* e *streams* apenas no contexto de uma tabela existente do *DynamoDB*. Estes são considerados como sub-recursos.

Entendendo Propriedade dos Recursos

O proprietário dos recursos é a conta AWS da entidade principal (seja uma conta de usuário root da AWS, um usuário IAM ou função IAM) que autentique o pedido de criação dos recursos.

- Contade usuário root da AWS – se utilizar das credenciais deste tipo de conta para criar uma tabela, a conta AWS será a proprietárias dos recursos (no *DynamoDB*, a tabela).
- usuário IAM – ao criar uma conta usuário IAM dentro da conta AWS e conceder permissões para criação de tabelas a este usuário, ele poderá criá-las mas os recursos da tabela pertencerão a conta.
- IAM user – ao criar uma função IAM dentro da conta AWS e conceder permissões para criação de tabelas a esta função, ele poderá criá-las mas os recursos da tabela pertencerão a conta.

Gerenciamento de Acesso a Recursos

Uma poítica de permissões define quem tem acesso a que. Políticas anexadas a uma identidade IAM são conhecidas como plíticas baseadas em identidade (políticas IAM). Políticas atreladas a um recurso são consideradas como políticas baseadas em recursos. *DynamoDB* suporta apenas políticas baseadas em identidade.

Políticas Baseadas em Identidade (Política IAM)

- Associar políticas de permissões a um usuário ou grupo – pode-se associar políticas de permissões a um usuário ou grupo ao qual o usuário pertence para criação de recursos da *Amazon DynamoDB*.
- Associar políticas de permissões a um função (concede permissões entre contas) – pode-se associar uma política de permissões baseada em identidade a uma função do IAM para conceder permissões entre contas. Por exemplo, o administrador na conta A pode criar uma função para conceder permissões entre contas a outra conta da AWS (por exemplo, conta B) ou um serviço da AWS da seguinte forma:

    1. O administrador da Conta A cria uma função do IAM e anexa uma política de permissões à função que concede permissões em recursos da conta A.
    2. Um administrador da conta A anexa uma política de confiança à função identificando a conta B como a principal, que pode assumir a função.
    3. O administrador da conta B pode acabar delegando permissões para assumir a função para todos os usuários na conta B. Isso permite que os usuários na conta B criem ou acessem recursos na conta A. O principal na política de confiança também poderá ser um serviço da AWS principal se você quiser conceder a um serviço da AWS permissões para assumir a função.

Veja a seguir um exemplo de política que concede permissões para uma ação do DynamoDB (dynamodb:ListTables). O caractere curinga (*) no valor Resource significa que você pode usar esta ação para obter os nomes de todas as tabelas de propriedade da conta da AWS na região da AWS atual.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListTables",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ListTables"
      ],
      "Resource": "*"
    }
  ]
}
```

O comando a seguir cria um usuário IAM chamado Bob na conta atual: `aws iam create-user --user-name Bob`

O comando a seguir associa a política de gerencia da AWS nomeada `AdminisratorAccess` ao usuário IAM chamado Alice: `aws iam attach-user-policy --policy-arn arn:aws:iam:ACCOUNT-ID:aws:policy/AdministratorAccess --user-name Alice`

O exemplo a a seguir remove a política de gerencia com o ARN (Amazon Resource Names) `arn:aws:iam::123456789012:policy/TesterPolicy` do usuário Bob: `aws iam detach-user-policy --user-name Bob --policy-arn arn:aws:iam::123456789012:policy/TesterPolicy`

Backup e Restauração

o ADDB oferece duas formas de back-up a partir da AWS, Point-in-time backup e On-demand backup, ambos descritos a seguir.

Point-in-time backup

Este modo de backup oferece a opção de restaurar o banco de dados para qualquer ponto a partir do momento em que ele é ativado (até no máximo 35 dias). O processo restaura a tabela antiga em uma nova tabela. Vale ressaltar que todas as configurações da tabela se manterão iguais a tabela mais recente.

No exemplo abaixo ativa-se o Point-in-time backup para a tabela music:

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

Logo após recupera-se o backup mais recente disponível(usualmente retorna os ultimos 5 minutos):

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time
```

Também é possível recuperar para um horário específico:

```
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
```

On-demand backup

É possível criar backups sob demanda com comandos como:

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

O backup on-demand é criado de maneira assíncrona, o usuário ainda pode executar querys nas tabelas que estão sendo salvas, mas não é possível pausar ou cancelar a operação e remover a tabela.
Após a criação do backup é possível listar todos backups:

```
aws dynamodb list-backups
```

E por fim é possível restaurar a tabela Music:

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-
1:123456789012:table/Music/backup/01489173575360-b308cd7d
```

Instalação do DynamoDB localmente

Para instalar o aws-cli:

1. Tem que instalar o java jdk e jre>8
2. https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-windows.html

Para configurar o aws cli

```
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]: json
```

Para rodar o servidor do banco localmente:

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

Para listar as tabelas:

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

esse localhost tem que ter em todas as queries pq ta local

o banco ja tem um default que ele usa quando voce omite o sharedDb

Para criar o tabela:

```
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
```

Para fazer queries

```
aws dynamodb batch-write-item --request-items file://Forum.json
```

```
{
    "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"}
                }
            }
        }
    ]
}
```

Referências

1. Amazon DynamoDB. What is Amazon DynamoDB. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html. Acessado em Abril 2021
2. Amazon DynamoDB. How it works. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.Partitions.html. Acessado em Abril 2021
3. Amazon DynamoDB. Core Components of Amazon DynamoDB. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html. Acessado em Maio 2021
4. Amazon DynamoDB. Relational (SQL) or NoSQL? Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.WhyDynamoDB.html. Acesso em Maio 2021
5. Amazon DynamoDB. Partitions and Data Distribution. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.Partitions.html. Acesso em Maio 2021
6. Amazon DynamoDB. Working with Queries in DynamoDB. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html. Acesso em Maio 2021
7. Amazon DynamoDB. Optimistic Locking with Version Number. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBMapper.OptimisticLocking.html. Acesso em Maio 2021
8. Amazon DynamoDB. Read Consistency. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html. Acesso em Maio 2021
9. Medium. The Architecture of Amazon's DynamoDB and Why Its Performance Is So High. Disponível em: https://medium.com/swlh/architecture-of-amazons-dynamodb-and-why-its-performance-is-so-high-31d4274c3129#:~:text=DynamoDB. Acesso em Abril 2021
10. Amazon DynamoDB Transactions: How It Works. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/transaction-apis.html
11. Security and Compliance in Amazon DynamoDB. Disponível em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/security.html
12. Understanding DynamoDB Condition Expressions. Disponível em: https://www.alexdebrie.com/posts/dynamodb-condition-expressions/#1-confirming-existence-or-non-existence-of-an-item
13. DynamoDB Transactions: Use Cases and Examples. Disponível em: https://www.alexdebrie.com/posts/dynamodb-transactions/