BANCO DE DADOS NOSQL

AULA 2

Prof. Alex Mateus Porn

CONVERSA INICIAL

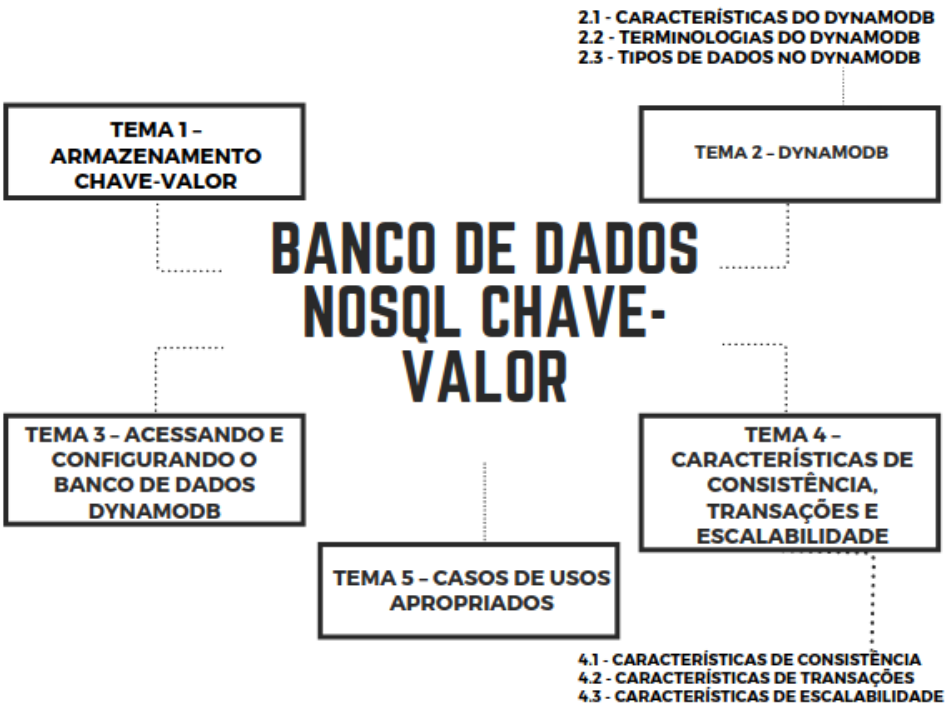
Olá!

Nesta aula falaremos sobre bancos de dados NoSQL chave-valor. O objetivo da aula é introduzir os principais conceitos sobre o modo de armazenamento e gerenciamento de dados do tipo chave-valor NoSQL e apresentar de forma prática, uma ferramenta para criação e gerenciamento de bancos de dados NoSQL chave-valor.

Esta aula se inicia com o modo de armazenamento chave-valor e as definições dos principais conceitos deste tipo de banco de dados. Você aprenderá como os dados são estruturados no tipo chave-valor e como são armazenados nas tabelas.

Também será apresentada a ferramenta de construção de bancos de dados NoSQL chave-valor DynamoDB, com a qual aprenderá a criar um banco de dados e a mensurar a capacidade de provisionamento necessária para a leitura e gravação dos dados em uma tabela.

A aula se encerra com a apresentação das principais aplicações de bancos de dados NoSQL chave-valor e, ao longo desta aula serão trabalhados os seguintes conteúdos:



TEMA 1 – ARMAZENAMENTO CHAVE-VALOR

Conforme vimos anteriormente, os bancos de dados NoSQL podem ser classificados de acordo com a estrutura em que os dados são armazenados. Os quatro modelos principais são os modelos orientado a chave-valor, orientado a documentos, orientado a colunas e orientado a grafos.

Conforme Marquesone (2017, p. 44), o modelo de banco de dados NoSQL orientado a chave-valor é o que possui a estrutura mais simples dentre os quatro modelos citados. Esse tipo de banco de dados NoSQL tem como estratégia o armazenamento de dados utilizando chaves como identificadores das informações gravadas em um campo identificado como valor. Normalmente, a chave é formada por um campo do tipo *String* e o campo valor pode armazenar diferentes tipos de dados, sem a necessidade de um esquema predefinido como ocorre nos bandos de dados relacionais.

O banco de dados orientado a chave-valor é uma categoria que veio para suprir necessidades presentes, em sua maioria, nas principais aplicações da Web 2.0, principalmente ao que se refere a cache/sessões de aplicações web. Conforme Marquesone (2017, p. 44), o banco de dados orientado a chave-valor pode ser utilizado tanto para persistir os dados em um banco quanto para mantê-los em memória e assim agilizar o acesso às informações. No caso de manter os dados em memória, é possível recuperar os dados em um banco de dados relacional e armazená-los em um cache, criando uma chave para cada valor armazenado. Assim, Marquesone (2017, p. 45), define que:

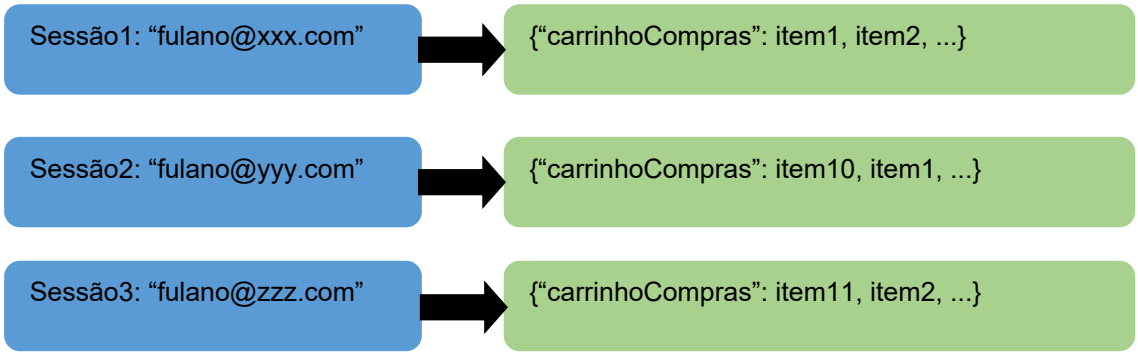
> Bancos de dados orientados a chave-valor são adequados para aplicações que realizam leituras frequentes, como por exemplo, um sistema de vendas online. Esse modelo de banco de dados NoSQL, possui uma estrutura bem mais simples do que o relacional, não sendo necessária a criação de tabelas, colunas e chaves estrangeiras. Apenas é necessário que cada registro tenha uma chave única e que se armazene um conjunto de informações referentes aos valores dessa chave.

Já Rockenbach (2017), menciona como principais utilizações dos bancos de dados orientados a chave-valor, aplicações para “gerenciar listas de itens

mais vendidos, carrinhos de compras de *e-commerce*, preferências do consumidor, gerenciamento de produtos, entre outras".

Como um exemplo prático de aplicação para um banco de dados orientado a chave-valor, podemos considerar um carrinho de compras de um site de *e-commerce*, adaptado de Marquesone (2017, p. 45), no qual os clientes acessam o catálogo de produtos e selecionam os itens que desejam, inserindo-os no carrinho, de modo que a aplicação necessita guardar essas informações até que o cliente finalize suas compras. A Figura 1 apresenta um exemplo da estrutura de armazenamento chave-valor para esse cenário.

Figura1 _ Exemplo de uma estrutura de armazenamento chave-valor.



Fonte: Elaborado com base em Marquesone, 2017, p. 45.

Simplificando, o modelo de armazenamento NoSQL orientado a chave-valor, é uma forma de armazenar um valor associado a uma chave, similar a estruturas de linguagens de programação, como as funções *Map* e *Reduce* nas linguagens Java ou Python. Conforme Furlaneto (2017):

> Podemos pensar em algo como várias gavetas, e cada gaveta possui um nome, que não pode se repetir, e dentro de cada gaveta possui um valor que a representa. Por exemplo, cada gaveta representa o nome de uma pessoa e dentro dela possui o valor da idade. Dessa forma, se queremos saber a idade de Pedro, necessitamos ir até a gaveta (chave) "Pedro", e acessar o valor correspondente.

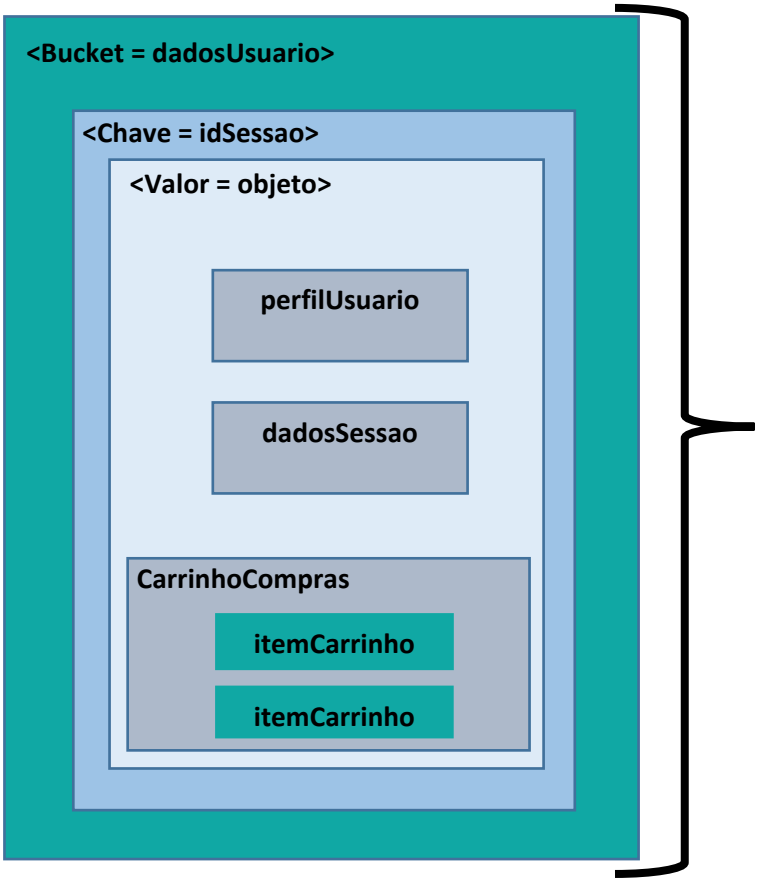
Os bancos de dados orientados a chave-valor têm como foco oferecer flexibilidade, desempenho e escalabilidade no gerenciamento de dados. Por esse motivo, conforme Marquesone (2017, p. 46) esse modelo de banco de

dados tende a resolver questões de lentidão para leitura e escrita de dados em grande variedade e volume, podendo otimizar o desempenho da consulta e realizar operações com alta vazão.

O modelo de depósito dos dados do banco de dados orientado a chave-valor obedece ao formato de uma tabela *hash,* chamada de *tabela* ou *bucket,* dependendo da terminologia do banco de dados. Um *bucket* é um contêiner lógico para um conjunto de itens associados, podendo receber itens como pares de chave-valor em que todo acesso é realizado por meio de uma chave.

Na Figura 2 os termos *bucket*, chave e valor são ilustrados, mostrando a composição desses elementos no cenário de armazenamento de dados do carrinho de compras.

Figura 2 – Representação dos elementos *bucket*, chave e valor.



Observe que no *bucket* os "dados do usuário" são identificados pela chave "id da sessão". Essa chave possui um valor que se refere a um "objeto" que contém: perfil do usuário, dados da sessão e o carrinho de compras com os itens do carrinho.

Fonte: Elaborado com base em Sadalage e Fowler, 2019, p. 125.

"A chave é um identificador único, associado a um item de dados e é usada para localizar esse item de dados rapidamente." (Elmasri, 2018, p. 807).

> “O valor é o termo usado para se referir ao objeto de dado armazenado, o qual corresponde a uma estrutura de dados aleatória, o que possibilita armazenar desde cadeia de bytes a listas variadas de dados.” (Elmasri, 2018, p. 807).

Portanto, não há uma preocupação com os dados a serem armazenados, como ocorre com os bancos de dados relacionais, em que são definidos os tipos de dados. O valor pode receber dados estruturados, semiestruturados ou ainda, não estruturados. Cabe ao aplicativo interpretar os dados recebidos (Sadalage; Fowler, 2019, p. 123 – 125). Porém, essa despreocupação com os tipos de dados do campo valor, também é uma limitação nesse tipo de banco de dados, não sendo possível fazer uma indexação com esse campo e uma consulta mais complexa. Conforme Marquesone (2017, p. 46) a única forma de realizar consultas no banco de dados orientado a chave-valor é por meio da chave. De acordo com Elmasri (2018, p. 807), em muitos desses bancos de dados orientados a chave-valor, não há linguagem de consulta, mas sim um conjunto de operações que podem ser usadas pelos programadores de aplicações.

Assim como ocorre com a realização de consultas nesse modelo chave-valor, os processos de inserção e exclusão dos dados também ocorre por meio da chave (Strauch, 2011).

Alguns dos sistemas de bancos de dados NoSQL que usam o modelo chave-valor são: DynamoDB, Couchbase, Riak, Redis, Memcached entre outros. Cada banco de dados apresenta características específicas. Sendo assim, cabe realizar uma análise da aplicação modelada para escolher o banco de dados mais apropriado. Para isso, vários fatores podem ser considerados, como a necessidade de crescimento, portabilidade dos dados para outros ambientes, e inevitavelmente também deve ser avaliado os custos envolvidos (Redislabs, 2020).

Para melhor compreensão e exemplificação do exposto no parágrafo anterior, no Quadro 1 é apresentada uma breve análise dos bancos de dados NoSQL orientados a chave-valor DynamoDB e Redis.

Quadro1 – Breve análise dos bancos de dados NoSQL DynamoDB e Redis.

DynamoDB	Redis
• Altamente disponível. • Mantém a durabilidade dos dados. • Totalmente gerenciado na nuvem por uma estrutura *serverless*. • Encarece a medida em que há alto processamento de dados.	• Mantém os pares chave-valor na memória. • Permite acesso rápido aos dados e bom desempenho. • Sacrifica a durabilidade dos dados. • Necessita gerenciamento e configuração de um servidor.

Neste tema desta aula falamos de um modo geral da estrutura de armazenamento de dados dos bancos de dados NoSQL orientados a chave-valor. A seguir, vamos conhecer um pouco mais sobre como é o funcionamento de um desses bancos dados, o DynamoDB.

TEMA 2 – DYNAMODB

Como já citado brevemente na sessão anterior, o DynamoDB, ou simplesmente Dynamo, é um banco de dados NoSQL de propriedade da *Amazon Web Services* – AWS. Mas antes de explorarmos um pouco mais o Dynamo precisamos entender o que é a estrutura *Serverless* da Amazon:

O AWS *Serverless Application Repository* (repositório de aplicativos sem servidor) é um modelo de computação em nuvem em que o servidor na nuvem executa o papel de servidor, gerenciando a alocação de recursos da máquina dinamicamente. Ele permite que equipes, organizações e desenvolvedores individuais armazenem e reutilizem aplicativos e montem e implantem com facilidade arquiteturas sem servidor (AWS, 2020). Para conhecer mais, acesse o site da AWS.

Em relação as suas características, o Dynamo suporta estrutura de dados chave-valor e documentos. Por ter uma estrutura *serverless*, todos os recursos necessários para o funcionamento arquitetural da aplicação são gerenciados

pela própria nuvem da AWS, a qual possibilita disponibilidade global da aplicação. Isso quer dizer que é possível selecionar localidades globais para o processamento do servidor, obtendo com isso melhores recursos de tempo de resposta. Diferentemente, há soluções em que o gerenciamento do servidor é um procedimento que precisa ser configurado e gerenciado manualmente, o que consequentemente torna sua configuração e uso mais complexo.

A estrutura *serverless* disponibiliza ao Dynamo integração com outros produtos da AWS, sendo essa uma vantagem atribuída a comunicação arquitetural, que possibilita ganhos de processamento e integração de serviços. Outro ponto a ser destacado é o custo atribuído pelo seu uso, quando o processamento de dados é baixo, o custo é reduzido, à medida que há a necessidade de mais recursos, o custo de utilização é proporcional.

Todavia, existe uma versão do banco de dados Dynamo para download, na qual é possível desenvolver e testar localmente aplicações, sem acessar a estrutura *serverless.* Isso possibilita usar o Dynamo mesmo sem conexão com a internet, e ainda sem utilizar recursos para armazenamento e transferência de dados.

2.1 Características do DynamoDB

Conforme abordado anteriormente, o DynamoDB é um banco de dados NoSQL totalmente gerenciado pela AWS, possuindo presença global, o que significa que o banco de dados não precisa estar hospedado em um único lugar, podendo ser disponibilizado em várias localidades geográficas em que a AWS esteja presente. A seguir são apresentadas algumas características específicas do DynamoDB:

- Capacidade de leitura e gravação adaptável, de acordo com as necessidades de uso;
- Alta escalabilidade, o que permite escalar o banco de dados de forma horizontal, sem a preocupação com o provisionamento de servidores;
- Backup e recuperação sob demanda;
- Recurso TTL (Time to Live), que permite ao administrador do banco definir o tempo em que o dado permanecerá no banco de dados.

Por se tratar de um banco de dados na nuvem, o DynamoDB disponibiliza alguns recursos para o controle de acesso, como por exemplo:

- Acesso via usuário ou grupos de usuários;
- Acesso via chaves privadas;
- Definir quais tipos de dados cada usuário pode acessar;
- Criptografia dos dados armazenados;
- Definição de métricas de utilização e operacionais.

Basicamente existem três formas de acesso ao DynamoDB:

- Acesso pelo AWS Console;
- Acesso pelo AWS CLI (*Command Line Interface*), ou seja, por linha de comando;
- Acesso pelo AWS SDK, normalmente utilizado pelos desenvolvedores para acesso as APIs do DynamoDB, com alguma linguagem de programação específica, como Python, Ruby, Java etc.)

2.2 Terminologias do DynamoDB

Ao contrário dos bancos de dados relacionais, em que no esquema é especificado o banco de dados, as tabelas e os relacionamentos entre essas tabelas por meio da definição das chaves primárias e chaves estrangeiras, no DynamoDB esse esquema é inexistente. As tabelas são independentes e possuem o nível mais alto no banco de dados, possuindo somente as chaves primárias, não existindo a definição de chaves estrangeiras. Esse tipo de estrutura apresenta um modelo bastante flexível, visto que uma tabela no DynamoDB pode possuir armazenamento de dados do tipo chave-valor e, outra pode possuir armazenamento de dados do tipo documento.

Para melhor compreensão e exemplificação do exposto no parágrafo anterior, no Quadro 2 é apresentada uma breve análise da comparação entre as terminologias do modelo de dados relacional e do DynamoDB.

Quadro 2 – Análise de terminologias entre os modelos relacional e DynamoDB

Modelo Relacional	DvnamoDB
• Database	• Não Existe
• Tabelas	• Tabelas
• Chave Primária	• Chave Primária
• Chave Estrangeira	• Não Existe
• Linhas	• Itens
• Colunas	• Atributos

Para uma análise um pouco mais detalhada, a Figura 3 apresenta um exemplo de uma tabela de vendas de um banco de dados relacional de um sistema online de *e-commerce*, com uma simulação dos mesmos dados dessa tabela armazenados no DynamoDB, no formato chave-valor.

Figura 3 – Modelo de tabela relacional *versus* DynamoDB

Modelo Relacional

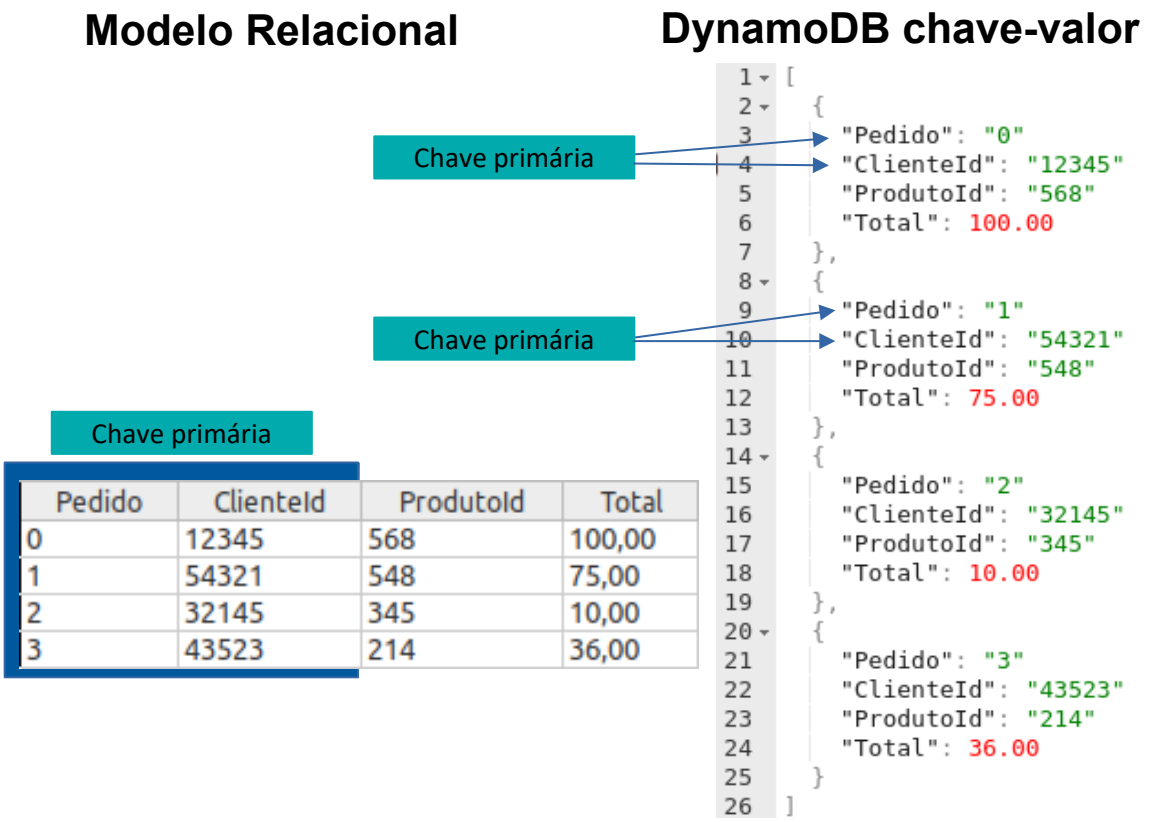
Pedido	ClienteId	ProdutoId	Total
0	12345	568	100,00
1	54321	548	75,00
2	32145	345	10,00
3	43523	214	36,00

DynamoDB chave-valor

```
[
  {
    "Pedido": "0"
    "ClienteId": "12345"
    "ProdutoId": "568"
    "Total": 100.00
  },
  {
    "Pedido": "1"
    "ClienteId": "54321"
    "ProdutoId": "548"
    "Total": 75.00
  },
  {
    "Pedido": "2"
    "ClienteId": "32145"
    "ProdutoId": "345"
    "Total": 10.00
  },
  {
    "Pedido": "3"
    "ClienteId": "43523"
    "ProdutoId": "214"
    "Total": 36.00
  }
]
```

Ao contrário das tabelas dos bancos de dados relacionais, em que a chave primária é opcional, no banco de dados NoSQL DynamoDB a chave primária é obrigatória. Ainda fazendo uma comparação com os bancos de dados relacionais, enquanto nesses a chave primária pode ser composta por múltiplas colunas, no DynamoDB ela é composta por no mínimo um e no máximo dois atributos. A Figura 4 apresenta a comparação entre as chaves primárias da tabela de vendas do modelo relacional do sistema online de *e-commerce* apresentada na Figura 3 e, da mesma tabela no DynamoDB.

Figura 4 – Representação da chave primária



2.3 Tipos de dados no DynamoDB

Os tipos de dados suportados no DynamoDB são divididos em três grupos: **Scalar Type**, **Set Type** e **Document Type**. O grupo de dados Scalar Type, somente permite um único valor por atributo, sendo pertencentes a este grupo os índices e chaves primárias do bando de dados, que por sua vez somente podem ser dos tipos string, número ou binário. Cinco tipos de dados compõem o grupo scalar type: String, Número, Boolean, Binário e Null, conforme representado no Quadro 3.

Quadro 3 – Tipos de dados Scalar Type

Scalar Type	**String**	Aceita texto no formato UTF8 Não aceita valores vazios
		Exemplo: “Faculdade”, “EaD”, “Graduação”
	Número	Aceita números positivos ou negativos
		Exemplo: 127, 12.56, -18.50, 0, 298
	Boolean	Aceita somente os valores verdadeiro ou falso
	Binário	Como o String, não aceita valores vazios
		Dados como imagens, vídeos, documentos (Dados binários BLOB)
	Null	Tipo de dados indefinido ou desconhecido

O grupo de dados **Set Type** representa um conjunto de valores do tipo Scalar, podendo ser um conjunto de Strings ou Números ou Boolean ou Binários ou Null. Esse tipo de dados não necessita ser ordenado, e por representar uma coleção de dados do tipo Scalar, também não pode conter valores vazios, e muito menos representar uma coleção vazia. Do mesmo modo, também não é permitido que um tipo de dados Set Type possua algum valor diferente dos tipos de dados suportados pelo tipo Scalar Type.

Exemplos de conjuntos de dados suportados pelo tipo Set Type:

- Strings: [“Faculdade”, “Educação a distância”, “Curitiba”, “Paraná”]
- Números: [2020, 109.20, -45, -29, 75, 0, 192]
- Boolean: [true, true, false, true, false, false, true]

Exemplo de um conjunto de dados não suportado pelo tipo Set Type:

- [“Faculdade”, 2020, true, “Curitiba”, 192, false]

Já o grupo de dados **Document Type**, diferentemente do Set Type que tem um único atributo com um conjunto de valores, o Document Type possui vários atributos, permite estruturas aninhadas e complexas, possibilitando a criação de uma estrutura de dados em até 32 níveis. Esse tipo de dados apresenta uma característica interessante, pois aceita os tipos *Maps* e *Lists*, porém não permite que sejam armazenados valores vazios dentro de *Maps* ou *Lists*, mas permite a existência de *Maps* e Lists vazios.

Exemplos de dados suportados pelo tipo Document Type:

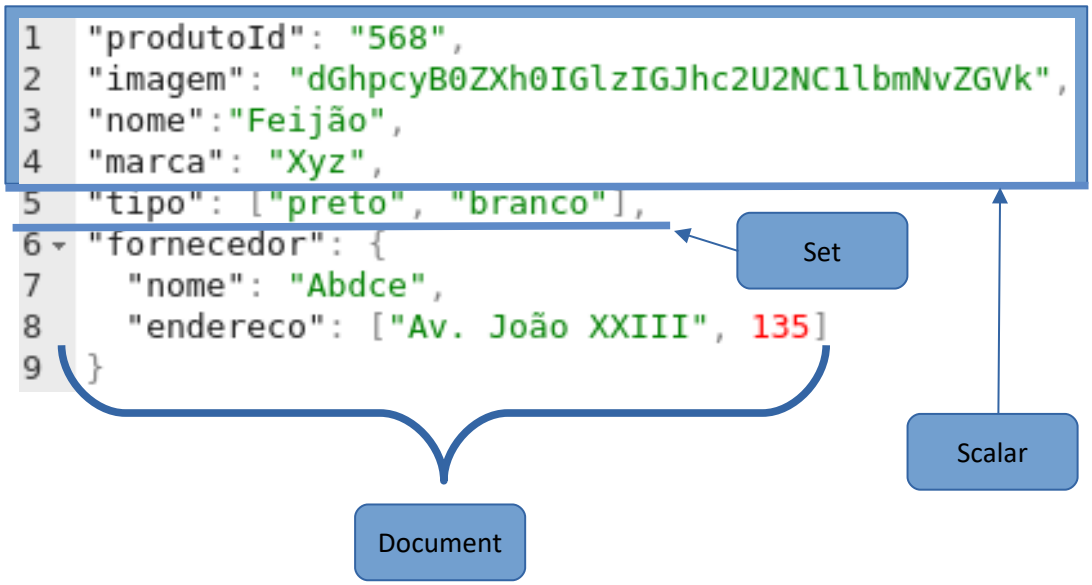
- [“Faculdade”, 2020, true, “Curitiba”, 192, false]
- [ ]
- {{“nome”:“Faculdade”}}

Exemplo de dados não suportados pelo tipo Document Type:

- [“ ”, “ ”]
- {{“nome”:” “}}

A Figura 5 apresenta os tipos de dados Scalar Type, Set Type e Document Type, representados nos atributos de uma tabela denominada *Produtos de um sistema online de e-commnerce.*

Figura 5 – Exemplo de aplicação dos tipos de dados

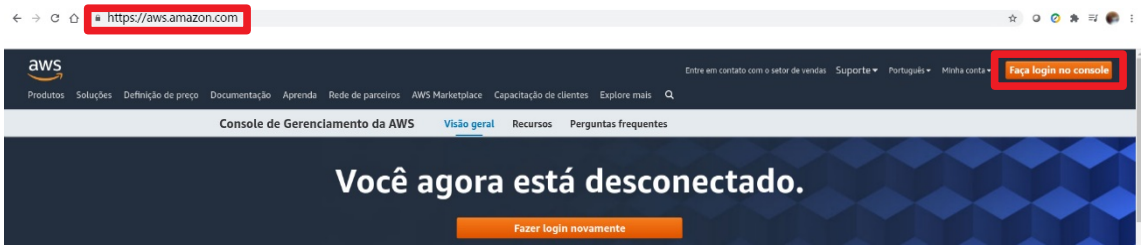


TEMA 3 – ACESSANDO E CONFIGURANDO O BANCO DE DADOS DYNAMODB

Conforme abordado nos tópicos anteriores, o DynamoDB por ser um banco de dados NoSQL totalmente gerenciado pela AWS, para configurá-lo primeiramente devemos acessar a página da Amazon: https://aws.amazon.com.

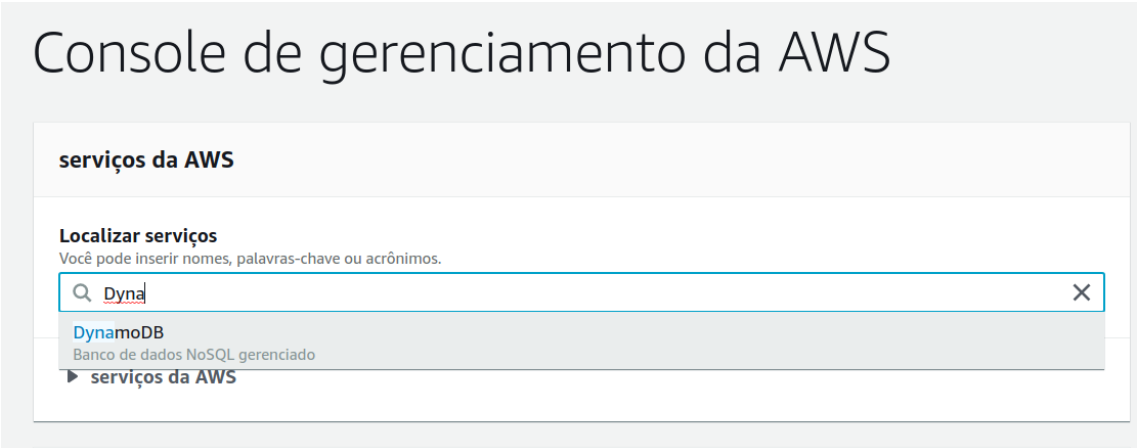
Caso seja o primeiro acesso a esta página, primeiramente será necessário criar uma conta na Amazon. Durante a criação da conta, a Amazon exige que seja fornecido um número de cartão de crédito, porém ao final do preenchimento do formulário, existe a possibilidade de escolha do plano gratuito. Porém, ao final da disciplina recomenda-se cancelar a conta.

Figura 6 – Página de acesso ao DynamoDB



Após a conta criada, basta clicar no botão “**faça login no console**”, conforme destacado em vermelho na Figura 6, informar o login e senha cadastrados e acessar o console da AWS. A Figura 7 apresenta a interface inicial do console do AWS.

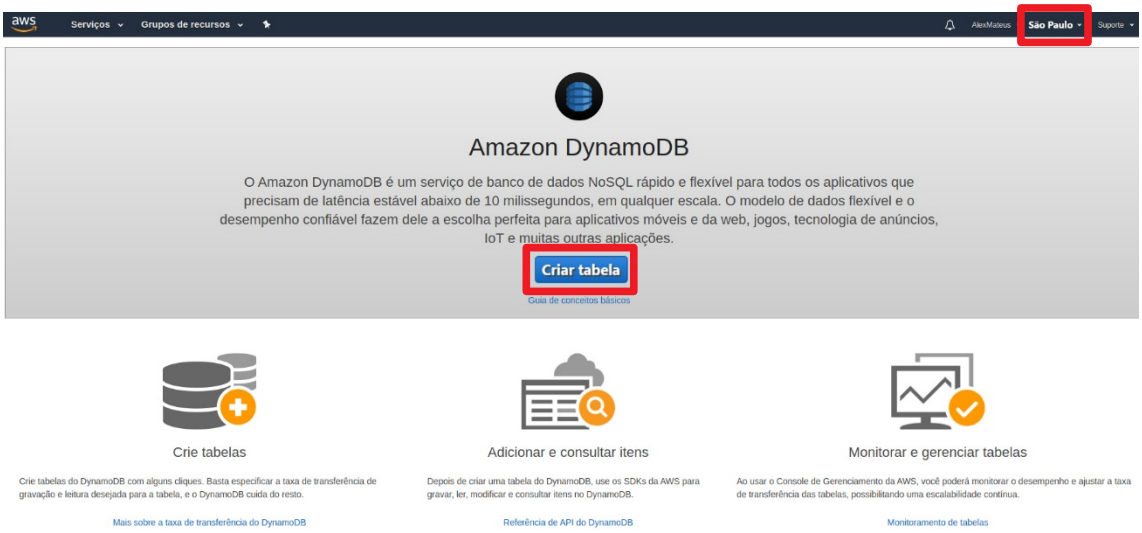
Figura 7 – Interface inicial da AWS



Para acessar o DynamoDB e criar as tabelas, basta digitar na caixa de seleção “Localizar serviços” o nome do banco de dados e selecionar o DynamoDB, conforme mostrado na Figura 7. Antes de criar a primeira tabela, é

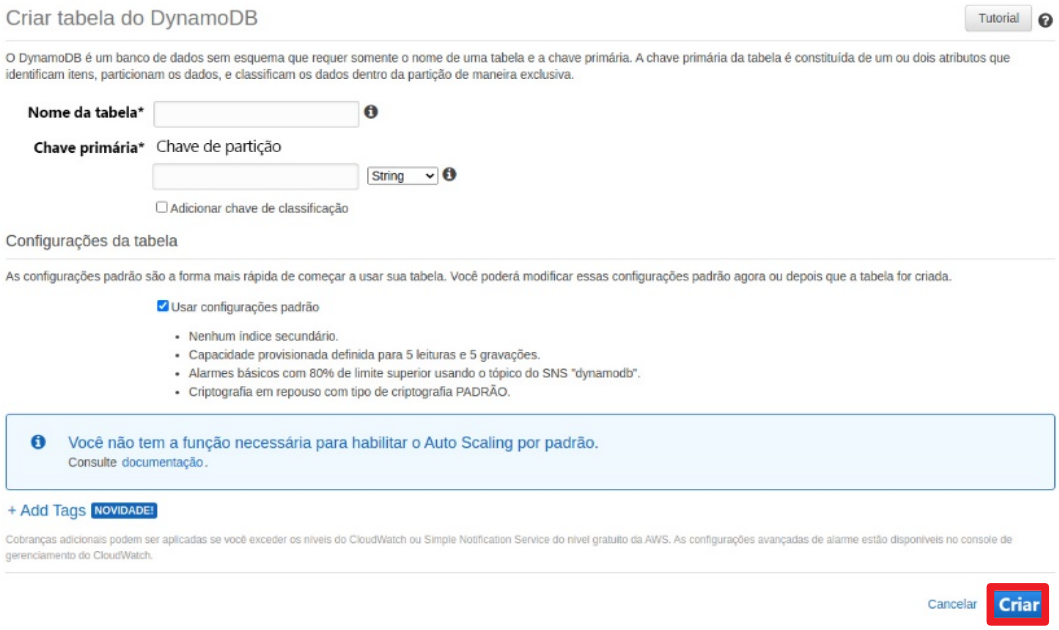
muito importante definir em qual região geográfica essa tabela será criada, pois em cada região os custos podem variar e, uma vez criada a tabela em uma região, não é mais possível alterá-la para outra. Mas como estamos utilizando a plataforma com fins acadêmicos e selecionamos o plano gratuito, independente da região não haverá custos. Na Figura 8 é possível observar destacado em vermelho, que a região selecionada é São Paulo. Ao clicar na seta ao lado direito da região, é listada todas as regiões disponíveis.

Figura 8 – Criação de tabelas no DynamoDb



Escolhida a região, basta clicar no botão “Criar tabela”, destacado em vermelho, ao centro da Figura 8, e será apresentada a interface conforme é mostrada na Figura 9.

Figura 9 – Interface de criação de tabelas no DynamoDB



A chave primária no DynamoDB é composta por dois campos, a chave de partição (p*artition key*) e, a chave de classificação (*sort key*).

- Chave de partição: é obrigatória, sendo utilizada para particionar os dados entre os *hosts* para escalabilidade e disponibilidade.

- Chave de classificação: é opcional, possibilita a criação de uma chave primária composta, sendo utilizada para pesquisar os dados dentro de uma partição.

Para exemplificar, criaremos uma tabela de filmes.

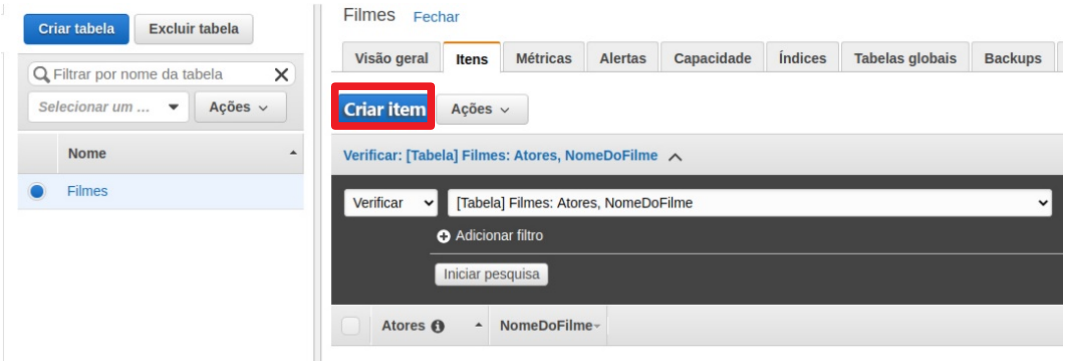
- **Nome da tabela:** Filmes
- **Chave primária:**
  - **Chave de partição:** Atores (tipo de dados String)
  - **Chave de classificação:** NomeDoFilme (tipo de dados String)
- **Usar configurações padrão:** Sim

A chave de partição “Atores” possibilita que sejam localizados todos os filmes em que determinado ator atuou. Já a chave de classificação “NomeDoFilme”, possibilita criar uma chave primária composta, permitindo a inserção de vários filmes para o mesmo ator.

O uso de configurações padrão possibilita uma capacidade de até cinco leituras e cinco gravações por segundo na tabela. No tema 4 abordaremos as capacidades provisionadas do DynamoDB, que influenciarão diretamente nos custos do banco de dados. O próximo passo é clicar no botão “Criar”, conforme destacado em vermelho no canto inferior direito na Figura 9, sendo carregada em seguida uma nova interface com as informações em destaque da tabela criada.

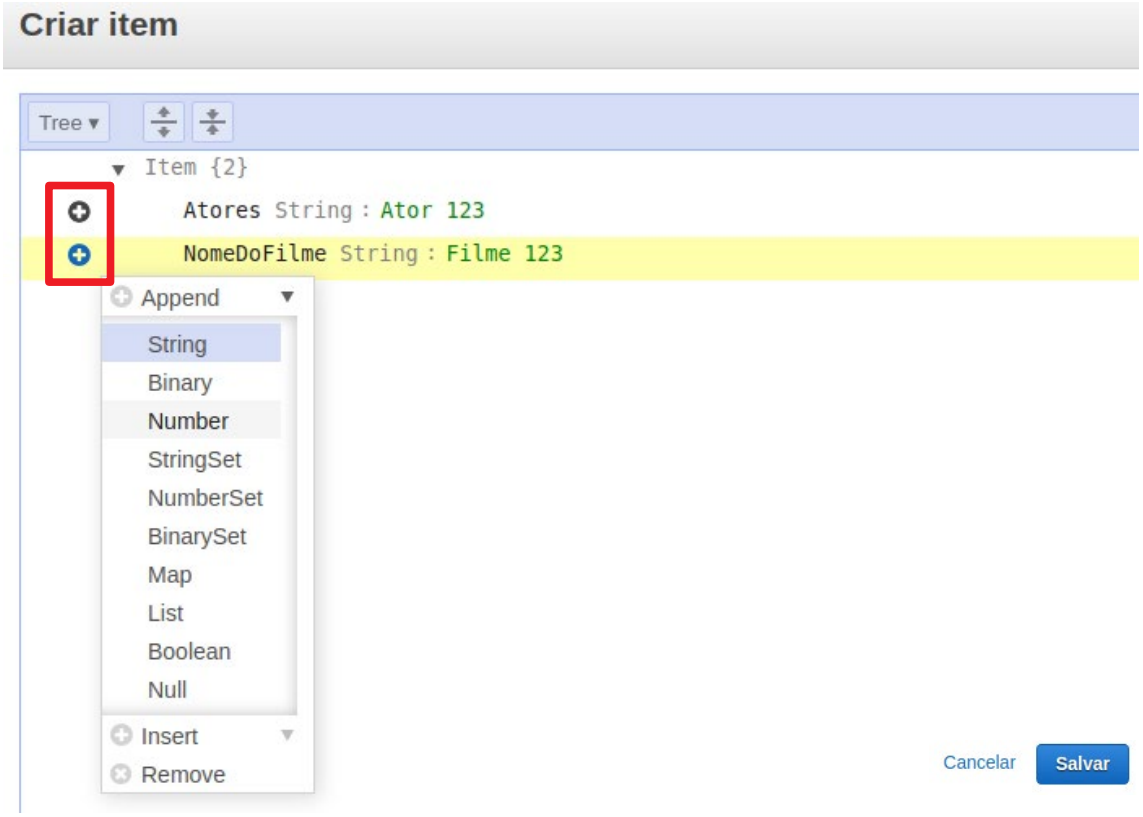
Para inserir registros ou novos atributos na tabela Filmes, basta clicar na aba “Itens” e em seguida no botão “Criar item”, conforme pode ser observado na Figura 10, destacado em vermelho.

Figura 10 – Interface da tabela Filmes no DynamoDB



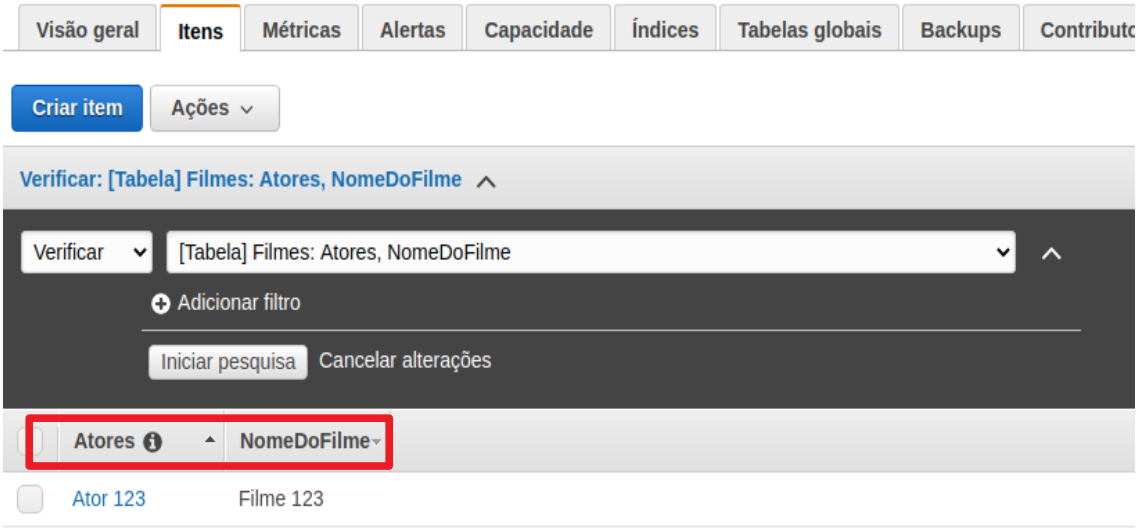
A Figura 11 apresenta a inserção dos dados “Ator 123” para o atributo “Atores” e, os dados “Filme 123” para o atributo “NomeDoFilme” da tabela “Filmes. Para a criação de novos atributos nessa tabela, basta clicar em um dos ícones destacados em vermelho ao lado de um dos atributos, escolher o tipo de dados desejado e confirmar informando o nome do novo atributo.

Figura 11 – Inserção de dados e criação de novos atributos



Após salvar os dados informados, os novos registros são listados no console, conforme destacado em vermelho na Figura 12.

Figura 12 – Registros da tabela Filmes



TEMA 4 – CARACTERÍSTICAS DE CONSISTÊNCIA, TRANSAÇÕES E ESCALABILIDADE

Como a AWS está presente em vários países do mundo, cada uma dessas presenças representa uma região, de modo que cada região é subdividida em zonas de disponibilidade e, cada zona possui no mínimo um ou mais *data centers* interligados entre si, sincronizados para ter confiabilidade, performance, redundância e tolerância a falhas.

Os dados armazenados em uma tabela no DynamoDB, são sincronizados entre os *data centers* da região escolhida, por isso o motivo de não ser possível alterar a região de uma tabela após a sua criação.

4.1 Características de consistência

A forma como os dados são lidos no DynamoDB ocorre por meio do modelo de Leitura de Consistência (*Read Consistency*). Esse modelo é subdividido em dois tipos: Eventualmente Consistente (*Eventually Consistent*) e Fortemente Consistente (*Strongly consistent*).

- Eventualmente Consistente: nesse modo, a leitura dos dados não é realizada diretamente na tabela do DynamoDB, mas sim nos dados armazenados em cache. O DynamoDB mantém um cache para leitura dos dados, de modo que o dado lido nem sempre é o último dado gravado na tabela.

- Fortemente Consistente: nesse modo, a leitura dos dados é realizada diretamente na tabela do DynamoDB, retornando sempre os dados mais atuais.

O modelo eventualmente consistente é o padrão de todas as operações realizadas no DynamoDB, sendo muito útil quando a tabela em questão demanda muita leitura e pouca gravação. Como exemplo, podemos imaginar uma tabela para cadastro de produtos que são disponibilizados em um site de *e-commerce* como um catálogo para os clientes visualizarem. Essa tabela, contendo milhares de itens armazenados, muito provavelmente, terá uma demanda de visualizações diárias dos produtos muito superior à demanda de gravação de dados, como atualização de preços ou inserção de novos itens.

4.2 Características de transações

Um dos recursos de configuração do DynamoDB está relacionado a capacidade de leitura e gravação dos dados. Essa configuração permite determinar quantos itens podem ser lidos ou escritos por segundo, estabelecendo uma taxa de transferência sob demanda ou provisionada, para que o banco consiga suprir o desempenho requerido pela aplicação de modo consistente e rápido (Rotenstein, 2017).

Essa capacidade de leitura e gravação é especificada durante a criação de cada tabela. Conforme abordamos no Tema 3, durante a criação da tabela Filmes, ao mantermos a caixa seleção “Usar configurações padrão” selecionada, por padrão essa tabela foi configurada automaticamente com uma capacidade provisionada de no máximo cinco leituras e cinco gravações por segundo.

A capacidade de leitura no DynamoDB é definida em RCU (*Read Capacity Unit*) e, a capacidade de gravação é definida em WCU (*Write Capacity Unit*). De modo que cada unidade de capacidade corresponde a uma requisição por segundo, sendo assim, nossa tabela Filmes, tem uma capacidade provisionada máxima de até cinco requisições de leitura e cinco de gravação por segundo. Conforme Rotenstein (2017):

- **1 RCU Fortemente Consistente** equivale a uma leitura por segundo em uma tabela.

- **1 RCU Eventualmente Consistente** equivale a duas leituras por segundo na tabela, pelo fato da leitura ser realizada no cache e não diretamente na tabela.

- **Cada RCU** corresponde a um bloco de 4Kb, ou seja, em cada leitura por segundo, é lido um bloco de 4Kb de dados.

- **1 WCU** equivale a uma gravação em uma tabela.

Vamos analisar agora qual é a capacidade máxima de leitura e de gravação de dados da nossa tabela Filmes, sabendo que ela possui uma capacidade provisionada de 5 WCUs.

Capacidade de Gravação (WCU) da tabela Filmes

1Kb (bloco de gravação) x 5 WCU = 5Kb/seg.

Agora vamos analisar qual é a capacidade máxima de leitura de dados da tabela Filmes, sabendo que ela possui uma capacidade provisionada de 5 RCUs.

Capacidade de Leitura (RCU) da tabela Filmes

Eventualmente consistente

4Kb (bloco de leitura) x 5 RCU = 20Kb/seg.

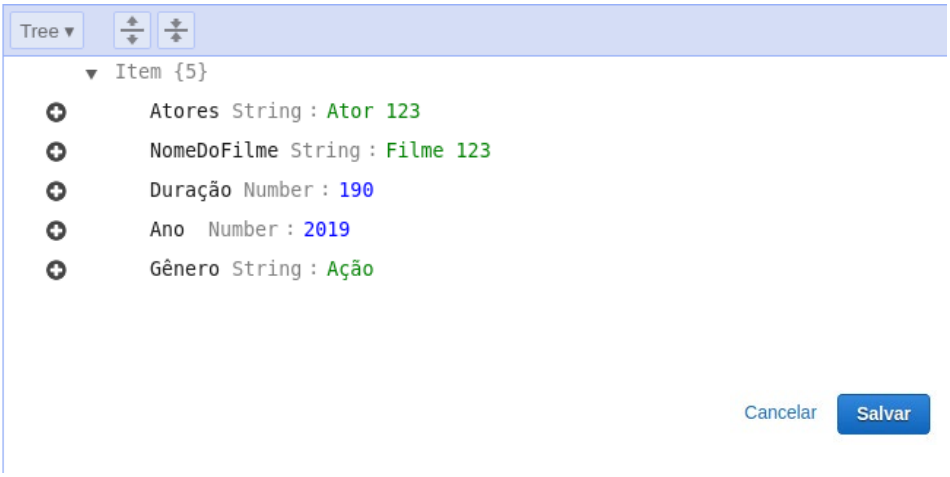
5 RCU x 2 tabelas = 40Kb/seg.

Fortemente consistente

4Kb x 5 RCU = 20Kb/seg.

Agora que sabemos a capacidade máxima de leitura e gravação da nossa tabela Filmes, ou seja, 40Kb por segundo de leitura no modo eventualmente consistente, 20Kb por segundo de leitura no modo fortemente consistente e, 5Kb por segundo de gravação, vamos analisar se essa capacidade de provisionamento é suficiente para armazenar os dados representados na Figura 11. Para termos um registro mais completo, vamos supor que além dos atributos “Atores” e “NomeDoFilme”, temos também os atributos “Duração”, “Ano” e “Gênero”, conforme representado na Figura 13.

Figura 13 – Atributos da tabela Filmes



Supondo que o registro apresentado na Figura 13 tenha um tamanho de 1.2Kb, o provisionamento necessário para ler e gravar esse registro é conforme apresentado abaixo:

Provisionamento necessário para gravação do registro

1,2Kb / 1Kb (bloco de gravação) = 1,2 RCU = 2 RCU

Tendo ainda como base o registro apresentado na Figura 13, e considerando seu tamanho em 1.2Kb, o provisionamento necessário para ler esse registro é conforme apresentado a seguir:

Provisionamento necessário para leitura do registro

Eventualmente consistente

1,2Kb / 4Kb = 0,3

0,3 / 2 tabelas = 0,15 RCU = 1 RCU

Fortemente consistente

1,2Kb / 4Kb (bloco de leitura) = 0,3 RCU = 1 RCU

4.3 Características de escalabilidade

O DynamoDB permite que a escalabilidade para cima seja realizada sempre que necessário. Por exemplo, havendo a necessidade de escalar nossa tabela Filmes de 5 para 7 RCUs e 7 WCUs, e durante o mesmo dia,

necessitamos realizar essa operação por mais 15 vezes, sempre elevando o provisionamento, todas as operações serão permitidas. Porém, a escalabilidade inversa, ou seja, diminuir a capacidade de provisionamento, somente é permitida que seja realizada por até quatro vezes ao dia.

TEMA 5 – CASOS DE USO APROPRIADOS

De acordo com os tópicos anteriores e conforme abordado pela AWS (2020), os “bancos de dados chave-valor podem lidar com a escalabilidade de grandes quantidades de dados e volumes extremamente altos de mudanças de estado enquanto atendem a milhões de usuários simultâneos por meio do processamento e armazenamento distribuído”. Em conformidade com essa definição, uma das principais aplicações de um banco de dados NoSQL chave-valor é na utilização de “carrinhos de compra” de sites de comércio eletrônico, conforme apresentado em vários exemplos durante esta aula, principalmente quando se trata de sistemas que em determinadas épocas do ano, como datas comemorativas, podem receber milhares de pedidos em questões de segundos.

Similar ao conceito de “carrinhos de compra” em sites de comércio eletrônico, aplicativos online que são orientados por sessão também são excelentes aplicações para uso de bancos de dados NoSQL chave-valor. Conforme a AWS (2020):

> Um aplicativo orientado por sessão, como um aplicativo online, começa uma sessão quando o usuário faz login e fica ativo até que se desconecte ou a sessão expire. Durante esse período, o aplicativo armazena todos os dados relativos à sessão na memória principal ou em um banco de dados. Os dados da sessão podem incluir informações de perfil do usuário, mensagens, dados e temas personalizados, recomendações, promoções direcionadas e descontos. Cada sessão de usuário tem um identificador exclusivo. Os dados de sessão nunca são consultados por nada além de uma chave primária, então um armazenamento de chave-valor rápido é mais adequado para dados de sessão. Em termos gerais, os bancos de dados de chave-valor podem proporcionar menor sobrecarga por página do que bancos de dados relacionais.

FINALIZANDO

Nesta aula abordamos de modo geral os conceitos e aplicações de uso sobre os bancos de dados NoSQL chave-valor e suas principais características.

A partir do tema 2, pudemos conhecer uma das principais ferramentas para criação e gerenciamento de bancos de dados NoSQL chave-valor, suas

características, modos de acesso, terminologias em relação aos tradicionais bancos de dados relacionais, e todos os tipos de dados suportados.

Utilizando um exemplo prático no Tema 3, aprendemos a criar tabelas no banco de dados NoSQL chave-valor, observamos em detalhes que essas tabelas são o nível mais alto da hierarquia desse tipo de banco de dados, ao contrário dos bancos de dados relacionais que possuem como nível mais alto o "*database*". Essas tabelas também não possuem relacionamentos entre chaves primárias e chaves estrangeiras, sendo independentes uma das outras.

Também compreendemos como é a capacidade de leitura e gravação de dados no banco de dados NoSQL chave-valor DynamoDB, usando exemplos práticos demonstrando o provisionamento necessário de uma tabela para leitura e gravação de um registro de dados. Compreendemos também que os bancos de dados NoSQL chave-valor são muito úteis principalmente para sistemas online que demandam excessivo número de acessos de leitura, se tornando mais rápidos e estáveis do que os bancos de dados relacionais, visto que possuem redundância incorporada, podendo lidar com a perda de nodos de armazenamento, caso falhas ocorram.

REFERÊNCIAS

AWS Serverless Application Repository: Descubra, implante e publique aplicativos sem servidor. **AWS**, c2020. Disponível em: <https://aws.amazon.com/pt/serverless/serverlessrepo/>. Acesso em: 1 mar. 2021.

O que é um banco de dados chave-valor? **AWS**, c2020. Disponível em: <https://aws.amazon.com/pt/nosql/key-value/>. Acesso em: 1 mar. 2021..

ELMASRI, N. **Sistemas de banco de dados**. 7ª ed. São Paulo: Pearson, 2018.

FURLANETO, L. **Introdução ao Redis, o NoSQL chave-valor mais famoso**. Disponível em: <https://imasters.com.br/banco-de-dados/introducao-ao-redis-o-nosql-chave-valor-mais-famoso >. Acesso em: 1 mar. 2021..

ROTENSTEIN, J. **Como calcular a unidade de capacidade de leitura e a unidade de capacidade de gravação para o DynamoDB**. It-swarm, 2017. Disponível em: <https://www.it-swarm.dev/pt/amazon-dynamodb/como-calcular-unidade-de-capacidade-de-leitura-e-unidade-de-capacidade-de-gravacao-para-o-dynamodb/829759160/>. Acesso em: 1 mar. 2021.

MARQUESONE, R. **Big Data:** Técnicas e tecnologias para extração de valor dos dados. São Paulo: Casa do Código, 2017.

ROCKENBACK, D. A., et al. **Estudo Comparativo de Banco de Dados Chave-Valor com Armazenamento em Memória**. Anais da XIII Escola Regional de Banco de Dados. SBC, 2017.

SADALAGE, P. J.; FOWLER, M. **NoSQL Essencial: Um guia conciso para o Mundo emergente da persistência poliglota**. 1ª ed. São Paulo: Novatec, 2019.

STRAUCH, C. **NoSQL databases:** Lecture Notes J. Stuttgart Media University, 2011.

Total Cost Comparison: Redis Enterprise vs AWS DynamoDB. **Redislabs**, c2020. Disponível em: <https://redislabs.com/docs/total-cost-comparison-redis-enterprise-vs-aws-dynamodb/>. Acesso em: 1 mar. 2021.