**BANCO DE DADOS NOSQL**

AULA 1

Prof. Alex Mateus Porn

**CONVERSA INICIAL**

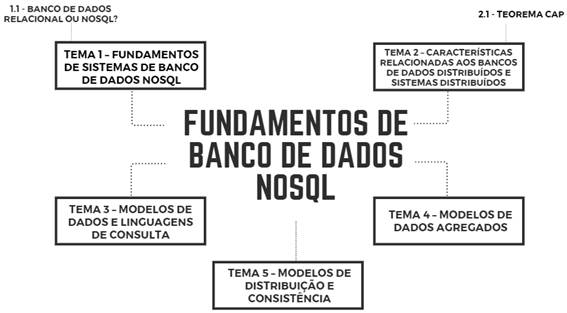
Olá! Nesta disciplina você vai estudar os conceitos relacionados ao universo de bancos de dados NoSQL.

Para começar, nesta aula serão apresentados os principais conceitos sobre sistemas de bancos de dados NoSQL e as suas diferentes categorias. Também será falado sobre os principais fatores que motivam o uso de um banco de dados NoSQL. Entre eles, destaca-se a obtenção de produtividade no desenvolvimento de aplicativos e a capacidade de processamento de grandes quantidades de dados, ou seja, alcançar escalabilidade.

Você aprenderá também que nos bancos de dados NoSQL os dados estão armazenados de forma isolada e em diferentes modelos, que não se tratam de tabelas e colunas como é o caso do modelo relacional, mas são novos tipos de estruturas, como chave-valor, documentos, colunas e grafos. Outra característica interessante é que os bancos de dados NoSQL apresentam esquemas de dados flexíveis, por meio da definição da agregação de dados. Essa agregação possibilita a adição de novos campos aos registros. Por fim, entenderá como é o comportamento de distribuição e consistência de dados.

Ao longo desta aula, serão trabalhados os seguintes conteúdos:

Figura 1 – Conteúdos



**TEMA 1 – FUNDAMENTOS DE SISTEMAS DE BANCOS DE DADOS NOSQL**

A computação, em linhas gerais, é caracterizada por apresentar constantes mudanças e progresso nos recursos disponíveis, sejam esses relacionados a software ou a hardware. No entanto, Sadalage e Fowler (2019, p. 13) relatam que ao longo de todos esses avanços, os bancos de dados relacionais conceitualmente se mostram estáveis, armazenando dados de forma estruturada e consolidada.

Entretanto, Elmasri (2018, p. 795) expõe que nos dias de hoje temos novas necessidades que requerem manipulação e gerenciamento de grandes volumes de dados, como ocorre em aplicações de mídias sociais, links da web, postagens em geral, entre outros.

Figura 2 – Mídias sociais



Créditos: solomon7/Shutterstock.

Como consequência dessa demanda, em 2009 foram manifestadas iniciativas provindas de vários projetos que buscavam armazenamento alternativo de dados, sendo esses apresentados em uma reunião envolvendo importantes desenvolvedores globais. A esta reunião, deu-se o nome de NoSQL, sendo desde então utilizado esse termo para denominar esse novo movimento de bancos de dados não relacionais.

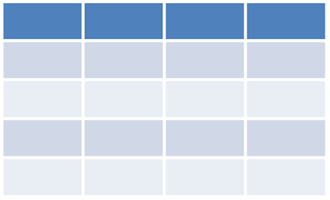
Para Elmasri (2018, p. 795):

A maioria dos sistemas NoSQL são bancos de dados distribuídos ou sistemas de armazenamento distribuído, com foco no armazenamento de dados semiestruturados, alto desempenho, disponibilidade, replicação de dados e escalabilidade, ao contrário da ênfase em consistência imediata de dados, linguagens de consulta poderosas e armazenamento de dados estruturados.

Antes de apresentarmos alguns modelos de bancos de dados NoSQL, vale ressaltar uma grande diferença entre esses e os modelos relacionais, deixando mais clara a compreensão e a diferença entre ambos.

Quando falamos em bancos de dados relacionais, logo nos vêm à mente alguns termos fundamentais, como esquema, relacionamentos, coleção de dados estruturados e inter-relacionados, etc.

Figura 3 – Bancos de dados relacionais



Já nos bancos de dados NoSQL, Sadalage e Fowler (2019, p. 36) destacam que não é necessário um esquema, pois os dados podem ser armazenados sem a necessidade de definição de uma estrutura, além de estes possuírem uma série de possibilidades de armazenamento semiestruturados.

Foi diante da necessidade de trabalhar com grandes volumes de dados que Elmasri (2018, p. 796) menciona a iniciativa de algumas empresas que desenvolveram seus próprios sistemas de gerenciamento de dados. A Google desenvolveu o Bigtable para usar em suas aplicações com necessidade de armazenamento de grandes quantidades de dados, como o Gmail, o Google Maps, entre outros. Além desse, desenvolveu outro sistema NoSQL para armazenamento de família de colunas, o Hbase, que estudaremos em outra aula. Já a Amazon desenvolveu o sistema NoSQL DynamoDB criando a categoria de armazenamento de dados usando o conceito de chave-valor. Ainda, o Facebook desenvolveu o sistema NoSQL chamado *Cassandra*, que usa conceitos de armazenamento de chave-valor e sistemas baseados em colunas. Além desses, outras empresas também desenvolveram soluções como os sistemas baseados em documentos, entre eles se destaca o MongoDB e CouchDB, e os sistemas baseados em grafos, incluindo soluções como o Neo4J e GraphBase.

Entretanto, antes de explorar mais detalhes sobre os bancos de dados NoSQL, precisamos de uma compreensão sobre computação distribuída no contexto de bancos de dados, que resultou em bancos de dados distribuídos (BDD). Um sistema de computação distribuída trabalha subdividindo a relação de um problema em relações menores – nodos ou nós; ao longo do texto será usado o termo *nodo*(s), mas algumas citações de Elmasri (2018) usarão o termo *nó*(s) –, que podem ser gerenciadas independentemente, mas quando interconectadas, trabalham de forma coordenada. Desse modo, os sistemas distribuídos possuem uma capacidade maior de processamento, apresentando bom desempenho, confiabilidade e suportando um número maior de usuários.

Inicialmente, como relatado por Elmasri (2018, p. 757), os bancos de dados distribuídos buscavam:

Resolver as questões de distribuição de dados, replicação de dados, consulta distribuída e processamento de transação, gerenciamento de metadados de banco de dados distribuído e outros temas. Mais recentemente, surgiram muitas tecnologias novas, que combinam tecnologias distribuídas e de bancos de dados. Essas tecnologias e esses sistemas estão sendo desenvolvidos para lidar com o armazenamento, a análise e a mineração de grandes quantidades de dados que estão sendo produzidos e coletados, e geralmente são conhecidas como tecnologias big data.

Atualmente, surgiram novas tecnologias, como os sistemas de bancos de dados NoSQL, “voltadas para oferecer soluções distribuídas no gerenciamento de grandes quantidades de dados necessárias em aplicações como mídia social, saúde, segurança, para citar apenas algumas” (Elmasri, 2018, p. 758). Por meio desses estudos, vamos compreender que cada um deles possui diferentes modelos de estrutura para armazenamento, sendo baseados em: documentos, chave-valor, colunas e em grafos. Na sequência, é apresentada uma breve explicação sobre cada uma dessas categorias de bancos de dados NoSQL, conforme exposto por Elmasri (2018, p. 799):

1. Sistemas NoSQL baseados em documentos: esses sistemas armazenam dados na forma de documentos usando formatos conhecidos, como JSON (*JavaScript Object Notation*). Os documentos são acessíveis por meio de seu ID de documento, mas também podem ser acessados rapidamente usando outros índices.

2. Armazenamentos de chave-valor do NoSQL: esses sistemas possuem um modelo de dados simples, com base no acesso rápido pela chave ao valor associado a esta chave; o valor pode ser um registro, um objeto, um documento ou até mesmo ter uma estrutura de dados mais complexa.

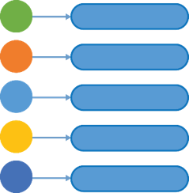
3. Sistemas NoSQL baseados em colunas ou em largura de colunas: esses sistemas particionam uma tabela por coluna em famílias de colunas [...] em que cada família de colunas é armazenada em seus próprios arquivos. Eles também permitem o versionamento dos valores de dados.

4. Sistemas NoSQL baseados em grafos: os dados são representados como gratos e os nós relacionados podem ser encontrados percorrendo suas arestas por meio de expressões de caminho.

Para um melhor entendimento, nos próximos parágrafos serão exploradas cada uma dessas categorias de bancos de dados NoSQL e serão apresentados exemplos, iniciando pelo modelo de armazenamento chave-valor (Key/value).

A**categoria de armazenamento chave-valor** utiliza um modelo de depósito de dados no qual são armazenados os pares (chave e valor) que correspondem a objetos indexados por chaves.

Figura 4 – Armazenamento chave-valor

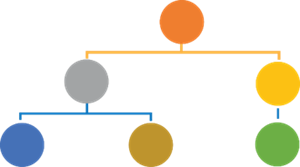


No modelo chave-valor, é possível inserir, consultar e apagar um valor por meio de uma determinada chave (Strauch, 2011). O valor é o termo usado para se referir ao objeto armazenado, o qual corresponde a uma coleção de dados. “Já que depósitos de chave-valor sempre fazem o acesso pela chave primária, eles têm, geralmente um ótimo desempenho e podem ser escaláveis facilmente” (Sadalage; Fowler, 2019, p. 123).

Sistemas de bancos de dados NoSQL que usam esse modelo chave-valor são: DynamoDb, Couchbase, Riak, Azure Table Storage, Redis, entre outros.

**A categoria de armazenamento orientado a documentos** possibilita o armazenamento de dados semiestruturados. Dessa forma, o banco possibilita armazenar documentos sem que haja uma estrutura comum, um esquema, embora ainda façam parte de uma mesma coleção.

Figura 5 – Armazenamento orientado a documentos



Fazendo um comparativo com o sistema gerenciador de banco de dados relacional (SGBDR), o banco de dados NoSQL orientado a documentos utiliza uma “diferente representação de dados; não é a mesma que se utiliza em um SGBDR, em que todas as colunas devem ser definidas e, se não contiverem dados, são marcadas como vazias ou nulas” (Sadalage; Fowler, 2019, p. 135). Nessa categoria, os “documentos são estruturas de dados na forma de árvores hierárquicas e autodescritivas, constituídas de mapas, coleções e valores escalares” (Sadalage; Fowler, 2019, p. 133).  Possibilita armazenar e recuperar documentos em formatos como: XML, JSON, BSON, entre outros.

Sistemas de bancos de dados NoSQL que usam esse modelo orientado a documentos são: MongoDb, CouchDB, RavenDb, entre outros.

A categoria de **armazenamentos de famílias de colunas** “permite que você armazene dados com chaves mapeadas para valores, e os valores são agrupados em múltiplas famílias de colunas, cada uma dessas famílias de colunas funcionando como um mapa de dados” (Sadalage; Fowler, 2019, p. 147).

Figura 6 –Armazenamento de famílias de colunas

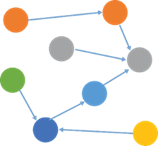


Fazendo um comparativo com um banco de dados relacional, pode-se destacar que esse é um modelo eficiente no processo de escrita dos dados de um registro, já o banco de dados de famílias de colunas é mais eficiente em aplicações em que é preciso otimizar a leitura de dados.

Sistemas de bancos de dados NoSQL que usam esse modelo de famílias de colunas são: Cassanda, Hypertable, Amazon SimpleDB, Hbase, Bigtable, Hadoop, entre outros.

A **categoria de armazenamento de grafos** “permite que você armazene entidades e também relacionamentos entre essas entidades” (Sadalage; Fowler, 2019, p. 161). As entidades são os nodos do grafo e os relacionamentos as arestas, sendo que ambos podem ter propriedades.

Figura 7 – Armazenamento de grafos



“A organização do grafo permite que os dados sejam armazenados uma vez e depois interpretados de formas diferentes baseadas em relacionamentos” (Sadalage; Fowler, 2019, p. 161).

Exemplos de sistemas de bancos de dados NoSQL que usam esse modelo de grafos são: Neo4J, Infinite Graph, OrientDB, entre outros.

Na Tabela 1, é apresentada uma classificação comparativa de classes de bancos de dados NoSQL e banco de dados relacional versus propriedades não funcionais.

Tabela 1 – Comparativo de classes de modelos de bancos de dados versus propriedades não funcionais

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Modelo** | **Performance** | **Escalabilidade** | **Flexibilidade** | **Complexidade** | **Funcionalidade** |
| Chave-valor | Alta | Alta | Alta | Nenhuma | Variável |
| Colunas | Alta | Alta | Moderada | Baixa | Mínima |
| Documentos | Alta | Variável | Alta | Baixa | Variável |
| Grafos | Variável | Variável | Alta | Alta | Teoria de Grafos |
| Relacional | Variável | Variável | Baixa | Moderada | Álgebra Relacional |

Fonte: Elaborado com base em Strauch, 2011, p. 26.

Nos parágrafos anteriores, conhecemos as diferentes categorias de bancos de dados NoSQL. De agora em diante, veremos algumas características dos sistemas NoSQL x Relacional.

**1.1 E AGORA? BANCO DE DADOS RELACIONAL OU NOSQL?**

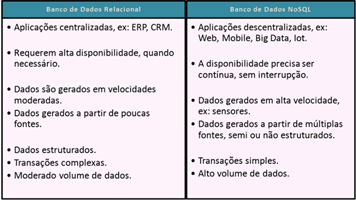
Neste ponto, Sadalage e Fowler (2019, p. 37) enfatizam que é preciso primeiro compreender quais dados queremos armazenar e como queremos manipular. Como resultado dessa análise, observa-se que há uma grande probabilidade de optar por uma combinação de diferentes tecnologias de armazenamento de dados. “Esse ponto de vista é, muitas vezes, chamado de persistência poliglota” (Sadalage; Fowler, 2019, p. 37). Isso se deve ao fato de existirem diferentes características de cada banco de dados em diferentes circunstâncias.

Enquanto os bancos de dados relacionais se destacam por possibilitar consistência dos dados, os bancos NoSQL são recomendados quando se espera por desempenho de processamento, em outras palavras, rápida recuperação, mostrando-se uma solução para o problema de escalabilidade existente nos bancos de dados relacionais. Isso ocorre porque no NoSQL o armazenamento de um conjunto de dados está em um mesmo registro e não depende do processamento de dados armazenados em outras tabelas, como se dá no modelo relacional.

Por exemplo, imagine um fórum em que cada postagem pode receber diversas interações. Ao usarmos um banco de dados relacional para a modelagem deste domínio, teremos a definição das tabelas: postagem e comentários, sendo necessário percorrer todos os comentários para identificar os que se referem a uma determinada postagem. Esse é um exemplo clássico a ser mencionado, no qual os comentários não fazem sentido sem a postagem e é preciso pesquisar milhares de comentários para identificar os que pertencem a uma postagem, podendo, assim, ocasionar um verdadeiro colapso para o servidor (óbvio que aqui colocamos uma pitada de dramatização). Com esse exemplo, podemos compreender que, neste caso, faz muito mais sentido ter os dados de postagens juntamente com os seus respectivos comentários. Assim, quando forem requisitados, a busca será completa e facilitada. “De modo geral, é muito útil poder colocar uma estrutura rica de informações em uma única solicitação ou resposta para reduzir o número de idas e vindas envolvidas nas comunicações remotas” (Sadalage; Fowler, 2019, p. 31).

Para ampliar o entendimento acerca da aplicabilidade e uso de um adequado modelo de banco de dados, no Quadro 1 são apresentados exemplos e tipos de aplicações que melhor condizem com um banco de dados relacional versus tipos de aplicações em que um banco de dados NoSQL melhor se adéqua.

Quadro 1 – Banco de Dados Relacional ou NoSQL – Exemplo de Aplicações



Fonte: Elaborado com base em Pereira, 2020.

Em suma, tanto o banco de dados relacional quanto o NoSQL são recomendados. Assim sendo, é comum ter aplicações com arquitetura híbrida, a qual faz uso de ambos os modelos, aproveitando o que se pode obter de melhor em cada qual, cabendo a escolha da solução ao que for mais adequado ao cenário.

Para concluir essa fundamentação teórica de bancos de dados NoSQL, segundo Sadalage e Fowler (2019, p. 18), “precisamos enfatizar que essa é uma área da computação que está em constante mudança. Aspectos importantes acerca destes tipos de armazenamento modificam-se todos os anos, e surgem novos recursos, novos bancos de dados”.

**TEMA 2 – CARACTERÍSTICAS RELACIONADAS AOS BANCOS DE DADOS DISTRIBUÍDOS E SISTEMAS DISTRIBUÍDOS**

Como relatado por Elmasri (2018, p. 797), essas características estão relacionadas à alta disponibilidade necessária para viabilizar o compartilhamento dos dados e também a escalabilidade, em virtude do contínuo aumento do volume dessas bases e alto desempenho. Para um melhor entendimento, nos próximos parágrafos serão explorados os detalhes a respeito dessas características.

 A escalabilidade de bancos de dados distribuídos pode ser horizontal e vertical. Já nos bancos de dados NoSQL, em linhas gerais, se usa escalabilidade horizontal, “na qual o sistema distribuído é expandido, adicionando mais nodos para armazenamento e processamento de dados à medida que o volume de dados aumenta” (Elmasri, 2018, p. 797).

Os bancos de dados NoSQL têm a necessidade de permanecerem disponíveis, ainda que ocorram falhas em nodos ou até mesmo se algum hardware ou software ficar indisponível. Para isso, “os dados são replicados em dois ou mais nodos de maneira transparente, de modo que, se um nodo falhar, os dados ainda estarão disponíveis em outros nodos” (Elmasri, 2018, p. 797).

Em relação a modelos de replicação, Elmasri (2018, p. 797) afirma que os bancos de dados NoSQL usam dois modelos, chamados de *replicação mestre-escravo*e *ponto a ponto*. “A replicação mestre-escravo exige que uma cópia seja a principal; todas as operações de gravação devem ser aplicadas à cópia principal e, em seguida, propagadas para as cópias escravas” (Elmasri, 2018, p. 797). Já a replicação ponto a ponto consiste em balancear a carga de acesso simultâneo aos registros de um arquivo, não armazenando o arquivo inteiro em apenas um nodo, mas distribuindo para vários nodos, e com isso aumentando a disponibilidade dos dados (Elmasri, 2018, p. 798).

No que diz respeito ao acesso a dados de alto desempenho, Elmasri (2018, p. 798) relata a problemática de encontrar registros ou itens de dados em um emaranhado de registros e itens de dados. Podemos associar isso à “busca por uma agulha no palheiro”. Para atenuar isso, os bancos de dados NoSQL geralmente usam técnicas de *hashing*, ou particionamento.

**2.1 TEOREMA CAP**

As transações de um banco de dados relacional mantêm a integridade dos dados por possuírem propriedades denominadas ACID (atomicidade, consistência, isolamento e durabilidade) (Pereira, 2020). Já os bancos de Dados NoSQL, visto que realizam armazenamento de dados distribuído, possuem propriedades de ambiente distribuído, sendo elas: consistência, disponibilidade e tolerância a partições, denominadas CAP, uma abreviação dos termos em inglês (Browne, 2020). Na sequência, será apresentada uma explicação sobre cada uma destas propriedades:

* **Consistência:** obter um estado consistente após a execução de uma operação. Ao ocorrer uma operação entre uma das cópias compartilhadas do sistema distribuído, sendo cada cópia um nodo, cada modificação de um nodo deve ter seus dados replicados entre as demais cópias, ou seja, a atualização do nodo deve simultaneamente atualizar suas cópias, possibilitando ter dados atualizados em todos os nodos.
* **Disponibilidade:**manter o sistema em operação, ainda que ocorram falhas entre os nodos ou até mesmo indisponibilidade de hardware e software.
* **Tolerância a partições:** manter o sistema em funcionamento mesmo que ocorram partições entre nodos devido a falhas ou a adição/remoção de nodos. O fato é que se falamos de bancos de dados com sistemas distribuídos, falamos de partição de cluster.

O teorema CAP foi introduzido por Brewer (2000) e, embora ele compreenda três propriedades que influenciam na arquitetura e funcionamento do armazenamento dos dados, somente se pode ter a garantia de apenas duas delas. Assim, é preciso optar por quais propriedades se deseja garantir, e isso pode influenciar diretamente na escolha do banco de dados NoSQL a ser utilizado em determinadas aplicações. Dessa forma, considerando a escolha de duas propriedades, se obtém a combinação de três alternativas. Essas combinações se referem às interseções entre consistência e disponibilidade (CA), consistência e tolerância a partições (CP) e disponibilidade e tolerância a partições (AP), destacadas em amarelo na representação do Teorema CAP, como mostra a Figura 8:

Figura 8 – Diagrama do Teorema CAP para distribuição



Fonte: Elaborado com base em Brown, 2020.

A interseção CA se refere à junção das propriedades: consistência e disponibilidade. As aplicações que optam por essa combinação de propriedades necessitam de forte consistência de leitura e escrita de dados e altíssima disponibilidade das aplicações, bastante clássico de aplicações com bancos de dados relacionais (Steppat, 2020; Araujo, 2020). “Nesse modelo, qualquer falha que ocorra em um dos nós, o sistema todo fica indisponível até que o nó que falhou volte ao normal” (Sirqueira, 2018).

A interseção CP se refere à junção das propriedades: consistência e tolerância a partições. Essa combinação de propriedades permite que as aplicações tenham consistência forte dos dados e tolerância a particionamento, permitindo resposta rápida às falhas. “Nesse modelo pode ocorrer que uma operação de escrita gere conflito entre os nós do particionamento de rede, sendo a disponibilidade comprometida até que ocorra um consenso entre os nós” (Sirqueira, 2018), ou seja, o SGBD tem autonomia para aceitar uma escrita ou não. Exemplos de bancos de dados com essas propriedades são: BigTable, Hbase ou MongoDB, Hadoop, entre outros (Steppat, 2020; Araujo, 2020).

A interseção AP se refere à junção das propriedades: disponibilidade e tolerância a partições. Essas propriedades são essenciais para aplicações que precisam estar disponíveis todos os dias e em todos os horários, ou seja, jamais podem ficar offline. Assim, o sistema sempre aceita operações de escrita e a consistência de dados é tratada posteriormente, por meio da sincronização. Tendo em vista esses dois aspectos, o período em que ocorre a persistência de um dado em um determinado nodo e o sincronismo desse nodo com os demais nodos do cluster, salienta-se que “existe uma janela de inconsistência, onde uma operação de leitura em um nodo que ainda não foi atualizado pode retornar dados desatualizados” (Sirqueira, 2018).  Exemplos de bancos de dados com essas propriedades são: Amazon DynamoDB, MongoDB, Cassandra, Riak, Voldemort, entre outros (Steppat, 2020; Araujo, 2020).

“O conhecimento do teorema CAP é fundamental para problemas do mundo real, em que a solução do problema atacado envolve o uso de sistemas distribuídos” (Sirqueira, 2018).

**TEMA 3 – MODELOS DE DADOS E LINGUAGENS DE CONSULTA**

“Sistemas NoSQL enfatizam desempenho e flexibilidade em relação a poder de modelagem e consulta complexa” (Elmasri, 2018, p. 798). Para melhor compreendermos, nos próximos parágrafos serão detalhadas essas características.

Quando criamos um banco de dados relacional, o primeiro passo consiste em criar um esquema. Entretanto, Sadalage e Fowler (2019, p. 61) expõem que é difícil saber antecipadamente o que armazenar. Assim, definir o tipo de armazenamento conforme se conhece mais sobre o projeto permite suportar dados não uniformes, ou seja, quando cada registro pode ser composto por diferentes campos.

Os bancos de dados NoSQL não utilizam esquema. Segundo Elmasri (2018, p. 798), a não exigência de um esquema é uma característica de vários sistemas de bancos de dados NoSQL, o que possibilita armazenar dados semiestruturados, como JSON, XML e autodescritivos. Mas isso não impede que seja especificado um esquema parcial, visando melhorar a eficiência do armazenamento.

De acordo com Sadalage e Fowler (2019, p. 61):

Um armazenamento de chave-valor permite o armazenamento de quaisquer dados sob uma chave. Um banco de dados de documentos faz, efetivamente, o mesmo, uma vez que não tem restrições à estrutura dos documentos armazenados. Bancos de dados de famílias de colunas permitem que sejam armazenados quaisquer dados sob qualquer coluna escolhida. Bancos de dados de grafos permitem que sejam adicionadas, livremente, novas arestas e propriedades aos nodos e às arestas.

Assim, ter um banco de dados sem esquema significa ter um banco de dados que suporta que seus registros sejam compostos por novos ou diferentes dados sem que haja a necessidade de realizar alterações na estrutura do banco.  Isso permite que os registros contenham apenas os dados necessários e facilita a manipulação de dados não uniformes. São esses fatores que possibilitam alta escalabilidade e disponibilidade nos bancos de dados NoSQL.

Apesar disso, “não utilizar esquemas é algo atrativo e, certamente, evita muitos problemas [...] novos problemas podem surgir” (Sadalage; Fowler, 2019, p. 62), visto que não ter um esquema também quer dizer que não há integridade de dados. Assim, para que a estrutura do banco de dados possa ser interpretada, é necessário ter um esquema implícito, ou seja, a interpretação do banco de dados passa para o software aplicativo.

Vogels (2020) utiliza como exemplo o carrinho de compras de uma loja virtual. O aplicativo que gerencia o carrinho de compras de um cliente pode unificar versões conflitantes e gerar um carrinho de compras unificado, ou então, optar por exibir apenas o último carrinho registrado. Perceba que esse gerenciamento não foi atribuído ao banco de dados, como ocorre nos bancos relacionais com a integridade dos dados, mas se trata de uma especificação funcional do aplicativo.

Figura 9 – Carrinho de compras



Crédito: Mariyani Sugianto/Shutterstock.

Outro ponto destacado por Elmasri (2018, p. 799) é que, diferentemente das aplicações com bancos de dados relacionais que usam o SQL, as aplicações com sistemas de bancos de dados NoSQL não podem realizar consultas com tantas condições e restrições. Assim, utiliza-se o termo de consulta menos poderosa, pois nesse modelo, a leitura identifica itens de dados em um único arquivo.

Outra característica importante a ser destacada e que é indispensável em qualquer aplicação é o versionamento. “Alguns sistemas NoSQL fornecem armazenamento de múltiplas versões dos itens de dados” (Elmasri, 2018, p. 799).

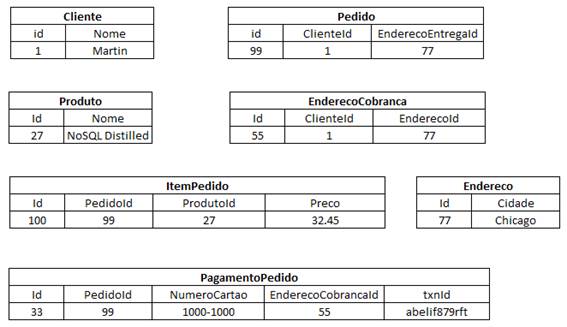
**TEMA 4 – MODELOS DE DADOS AGREGADOS**

Usamos o termo *modelo de dados* para nos referirmos  ao “modelo pelo qual o gerenciador do banco de dados organiza seus dados” (Sadalage; Fowler, 2019, p. 49). “Bancos de dados relacionais não possuem o conceito de agregado em seu modelo de dados, de modo que os chamamos de ‘não agregados’” (Sadalage; Fowler, 2019, p. 49).

Os bancos de dados NoSQL nas categorias chave-valor, documento e família de colunas possuem o conceito de orientação agregada em seu modelo de dados (Sadalage; Fowler, 2019, p. 42). “Bancos de dados de grafos não são agregados” (Sadalage; Fowler, 2019, p. 49). Eles diferem dos agregados em diversos aspectos, exceto por também diferirem dos bancos de dados relacionais e por coincidentemente terem sua ascensão no mesmo período (Sadalage; Fowler, 2019, p. 61).

Para exemplificar a diferença entre o modelo relacional e o modelo agregado, no Quadro 2, é mostrado um modelo relacional elaborado para suportar transações de um website de comércio eletrônico.

Quadro 2 – Modelo relacional para um website de comércio eletrônico



Fonte: Elaborado com base em Sadalage e Fowler, 2019, p. 43.

Ao apresentar esse exemplo, o autor explica o modelo supondo que neste comércio eletrônico “venderemos itens diretamente aos clientes pela web e teremos de armazenar informações sobre os usuários, o nosso catálogo de produtos, os pedidos, as remessas, os endereços de envio, os endereços de cobrança e os dados sobre o pagamento” (Sadalage; Fowler, 2019, p. 61).

Considerando esse cenário, mas adaptando essa modelagem para orientação de dados agregados, na figura a seguir temos a representação dessa modelagem no formato JSON, que é comumente usada para representação de dados NoSQL. Analisando o código apresentado, os comentários “// em clientes” e “// em pedidos” representam o trecho do código que divide essa agregação em duas partes: cliente e pedido. “O cliente contém uma lista de endereços de cobrança; o pedido contém uma lista de itens solicitados, um endereço de envio e pagamentos. O próprio pagamento contém um endereço de cobrança” (Sadalage; Fowler, 2019, p. 46).

Figura 10 – Agregação de dados para um website de comércio eletrônico



Fonte: Sadalage e Fowler, 2019, p. 45.

**TEMA 5 – MODELOS DE DISTRIBUIÇÃO E CONSISTÊNCIA**

Nos temas anteriores desta aula, falamos do armazenamento distribuído de dados dos bancos de dados NoSQL, que propicia escalabilidade à medida que aumenta o volume de dados das aplicações. No Tema 4, falamos sobre o modelo de dados agregado, que é apropriado para um modelo de distribuição de dados. A seguir, vamos conhecer um pouco do comportamento da distribuição de dados.

**5.1 DISTRIBUIÇÃO**

A distribuição de dados pode ser realizada de duas formas: replicação e fragmentação. “A replicação obtém os mesmos dados e os copia em múltiplos nodos. A fragmentação coloca dados diferentes em nodos diferentes” (Sadalage; Fowler, 2019, p. 73). Ambas as técnicas podem ser usadas de forma isolada ou em conjunto.

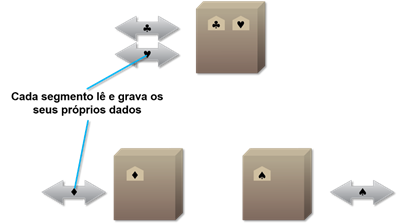
Cabe ressaltar que o mais recomendado é não utilizar distribuição, mantendo o banco de dados em um único servidor. Embora se tenha uma forte associação entre bancos de dados NoSQL com execução em cluster, a opção em um único servidor é mais facilmente gerenciada e menos complexa para os desenvolvedores, por não terem que se preocupar com a distribuição dos dados no desenvolvimento das aplicações (Sadalage; Fowler, 2019, p. 74).

No entanto, “a medida em que o volume de dados cresce, aumenta a necessidade de escalabilidade e melhoria de desempenho” (Monge, 2020). Podemos considerar também outros fatores como, por exemplo, a necessidade de aumentar o processamento e armazenamento das máquinas, assim como o acesso a diferentes partes de dados. Para essa conjuntura, é necessária a utilização de recursos de distribuição e consequentemente um aumento de máquinas que possam realizar o armazenamento e processamento de dados.

**5.1.1 FRAGMENTAÇÃO**

Uma solução é utilizar a técnica de fragmentação, também conhecida por *sharding*, que possibilita escalabilidade horizontal, direcionando o acesso a diferentes partes de dados em diferentes servidores. Cada servidor desempenha o papel de gerenciar um subconjunto de dados, que representa um fragmento, o qual lê e grava seu próprio dado. Na Figura 11, é ilustrada a movimentação dos dados com a fragmentação, que “coloca dados diferentes em nodos separados, cada um desses executando suas próprias leituras e gravações” (Sadalage; Fowler, 2019, p. 75).

Figura 11 – Fragmentação



Fonte: Elaborado com base em Sadalage e Fowler, 2019, p. 75.

Crédito: EgudinKa/Shutterstock.

A complexidade de uso da fragmentação está no modelo de dados, o qual deve ser projetado com a agregação mais adequada possível, de modo que o acesso a um servidor (nodo) possibilite ao usuário encontrar a maior parte dos dados. Consequentemente, isso possibilita que esse conjunto de agregados sejam distribuídos uniformemente pelos nodos (Sadalage; Fowler, 2019, p. 75).

Exemplificando a fragmentação, imagine que a organização dos registros de clientes seja separada em nodos seguindo um modelo de arquivamento de fichários que usa, por exemplo, organização em ordem alfabética.

Nesse caso, cada nodo receberia apenas registros de clientes com a letra alfabética de armazenamento estabelecida, ou seja, um nodo para clientes em que o nome inicia com a letra A, outro com a letra B, assim por diante (Sadalage; Fowler, 2019, p. 76).

Figura 12 – Registros de clientes pelas iniciais do nome



Crédito: hvostik/Shutterstock.

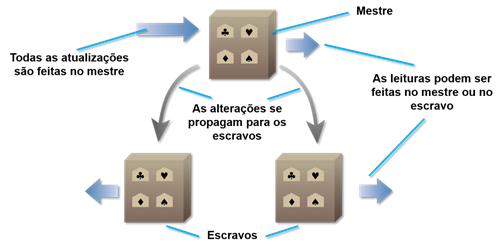
No cenário descrito, a programação do aplicativo é quem deve gerenciar esta organização, de modo que as consultas sejam realizadas no fragmento correspondente. Felizmente, muitos bancos de dados NoSQL já fazem o gerenciamento de alocação e a consulta de dados nos fragmentos automaticamente.

**5.1.2 REPLICAÇÃO**

“A replicação copia os dados para múltiplos servidores, de modo que cada parte dos dados pode ser encontrada em múltiplos lugares” (Sadalage; Fowler, 2019, p. 82). Sendo assim, esta diminui o tempo de recuperação de informações e propicia escalabilidade. Há dois tipos de replicação: mestre-escravo e ponto a ponto (p2p).

Na replicação mestre-escravo, os dados são copiados para múltiplos nodos, sendo um nodo designado como mestre e os demais escravos. O nodo mestre é “a fonte oficial dos dados e, geralmente, fica responsável por processar quaisquer atualizações nesses dados” (Sadalage; Fowler, 2019, p. 77). Os nodos escravos recebem os dados por um processo de sincronização com o mestre e são fontes apenas para leitura, o que proporciona escalabilidade horizontal quando há muitas solicitações de leitura (Sadalage; Fowler, 2019, p. 77). Na Figura 13, é ilustrada a movimentação dos dados nesse tipo de replicação.

Figura 13 – Replicação mestre e escravo



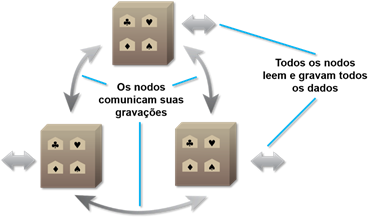
Fonte: Elaborado com base em Sadalage e Fowler, 2019, p. 78.

Crédito: EgudinKa/Shutterstock.

“A replicação mestre-escravo ajuda com a escalabilidade de leitura, mas não com a escalabilidade de gravação” (Sadalage; Fowler, 2019, p. 79). Já na replicação ponto a ponto, não há diferenciação entre os nodos, e todos permitem leitura e gravação de dados, sendo possível “contornar falhas nos nodos sem perder o acesso aos dados. Além disso, você pode “adicionar facilmente nodos para melhorar o seu desempenho” (Sadalage; Fowler, 2019, p. 80).

Na Figura 14, é ilustrada a movimentação dos dados no tipo de replicação ponto a ponto.

Figura 14 – Replicação ponto a ponto



Fonte: Elaborado com base em Sadalage e Fowler, 2019, p. 80.

Crédito: EgudinKa/Shutterstock.

**5.2 CONSISTÊNCIA**

A seção 2.1 abordou os conceitos do Teorema CAP e vimos que os bancos de dados NoSQL possuem propriedades de ambiente distribuído, sendo a tolerância a partições mandatória para atender aos conceitos de distribuição que vimos na seção 5.1. Outro paradigma que difere os bancos de dados NoSQL dos bancos de dados relacionais é a forma como se trata a consistência dos dados, denominada *consistência eventual*, pois é preciso criar recursos programáveis para estabelecer a consistência dos dados.

Com isso, podemos ter conflitos de gravação e de leitura-gravação. O primeiro ocorre, por exemplo, quando temos dois usuários tentando gravar os mesmos dados simultaneamente. Já o segundo ocorre quando um usuário lê um dado inconsistente durante uma gravação que está sendo realizada por outro usuário.

Perante esses conflitos, é possível atacar duas abordagens de controle de concorrência: pessimista e otimista. “Abordagens pessimistas bloqueiam os registros de dados para evitar conflitos. Abordagens otimistas detectam conflitos e os resolvem” (Sadalage; Fowler, 2019, p. 100).

Embora a consistência seja algo importante para a integridade dos dados, às vezes precisamos correr o risco de perdê-la a fim de não sacrificar outras características. Esse balanceamento é contornado estabelecendo recursos programáveis que podem ser definidos no desenvolvimento da aplicação.

**FINALIZANDO**

Por meio desse estudo inicial, adquirimos uma visão geral sobre os elementos que compõem os bancos de dados NoSQL e suas principais características.

Compreendemos também que os bancos de dados NoSQL não vieram para substituir os bancos de dados relacionais, e que tanto o seu comportamento quanto sua finalidade de uso diferem bastante daqueles, de modo que temos aplicações que combinam os dois tipos, aproveitando o que há de melhor em cada um.

Aprendemos alguns conceitos essenciais para iniciarmos nossos projetos usando bancos de dados NoSQL, como o teorema CAP, modelos de dados agregados, distribuição e consistência.

Nas próximas aulas, conheceremos alguns sistemas gerenciadores que aplicam essas propriedades e diferentes modelos, brevemente citados ao longo desta aula.

**REFERÊNCIAS**

ARAUJO, J. R. **Medium**. Disponível em: <https://medium.com/@jrobertoaraujo/teorema-cap-3094645d7249>. Acesso em: 28 jan. 2021.

BREWER, E. A. **Towards Robust Distributed Systems**. Keynote at the ACM Symposium on Principles of Distributed Computing (PODC), 2000. Disponível em: <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>. Acesso em: 28 jan. 2021

BROWNE, J. **Brewer's CAP Theorem**. Disponível em: <http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>.  Acesso em: 28 jan. 2021

BROWN; C. **NoSql Tips and Tricks**. Disponível em: <http://blog.nosqltips.com/search?q=CAP>.  Acesso em: 28 jan. 2021

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

MONGE; W. **Técnicas de Modelagem de Dados (NOSQL)**. Disponível em: <https://medium.com/t%C3%A9cnicas-de-modelagem-de-dados-nosql/banco-de-dados-nosql-58726ce6886c>.  Acesso em: 28 jan. 2021

PEREIRA, N. **Quando Utilizar RDBMS ou NOSQL?**. Disponível em: <http://datascienceacademy.com.br/blog/quando-utilizar-rdbms-ou-nosql/>.  Acesso em: 28 jan. 2021

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

SIRQUEIRA, T.; DALPRA, H. **NoSQL e a Importância da Engenharia de Software e da Engenharia de Dados para o Big Data:** Jornadas de Atualização em Informática. Sociedade Brasileira de Computação, 2018.

STEPPAT, N. **NoSQL – Do teorema CAP para P?(A|C):(C|L)**. Disponível em: <https://blog.caelum.com.br/nosql-do-teorema-cap-para-paccl/>.  Acesso em: 28 jan. 2021

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

VICTORINO, M. **NOSQL**. Disponível em: <http://www.ms.senai.br/transparencia\_senai/uploads/integridade/17-04-2019-01-No-SQL.pdf>.  Acesso em: 28 jan. 2021

VOGELS, W. **All Things Distributed**. Disponível em: <http://www.allthingsdistributed.com/2007/10/amazons\_dynamo.html>.  Acesso em: 28 jan. 2021

**BANCO DE DADOS NOSQL**

AULA 2

Prof. Alex Mateus Porn

**CONVERSA INICIAL**

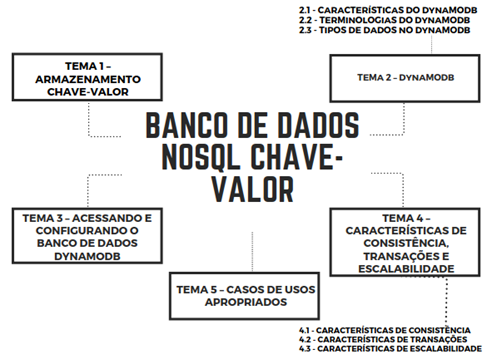
Olá!

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:

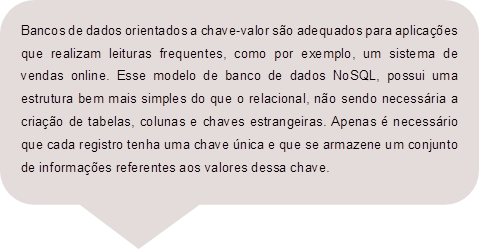


**TEMA 1 – ARMAZENAMENTO CHAVE-VALOR**

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.

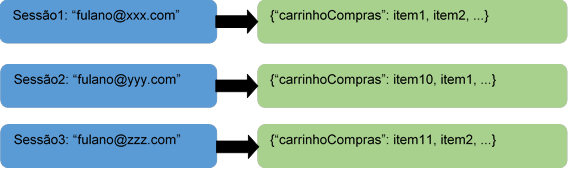
O banco de dados orientado a chave-valor é uma categoria que veio para suprir necessidades presentes, em sua maioria, nas principais aplicações da Web 2.0, principalmente ao que se refere a cache/sessões de aplicações web. Conforme Marquesone (2017, p. 44), o banco de dados orientado a chave-valor pode 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:



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”.

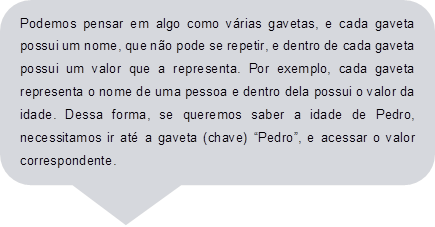
Como um exemplo prático de aplicação para um banco de dados orientado a chave-valor, podemos considerar um carrinho de compras de um site de *e-commerce*, adaptado de Marquesone (2017, p. 45), no qual os clientes acessam o catálogo de produtos e selecionam os itens que desejam, inserindo-os no carrinho, 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):

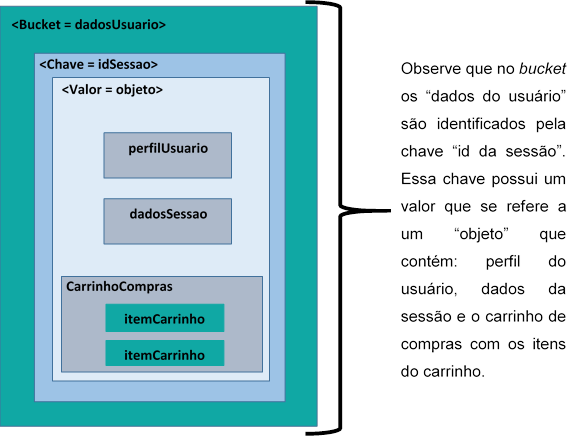


Os bancos de dados orientados a chave-valor têm como foco oferecer flexibilidade, desempenho e escalabilidade no gerenciamento de dados. Por esse motivo, conforme Marquesone (2017, p. 46) esse modelo de banco de dados tende a resolver questões de lentidão para leitura e escrita de dados em grande variedade e volume, podendo otimizar o desempenho da consulta e realizar operações com alta vazão.

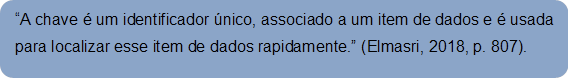
O modelo de depósito dos dados do banco de dados orientado a chave-valor obedece ao formato de uma tabela *hash,*chamada de *tabela* ou*bucket,*dependendo da terminologia do banco de dados*.* Um*bucket*é um contêiner lógico para um conjunto de itens associados, podendo receber itens como pares de chave-valor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

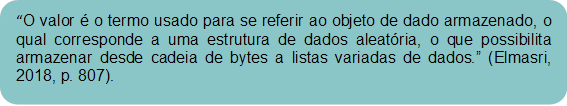
Na Figura 2 os termos *bucket*, chave e valor são ilustrados, mostrando a composição desses elementos no cenário de armazenamento de dados do carrinho de compras.

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



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





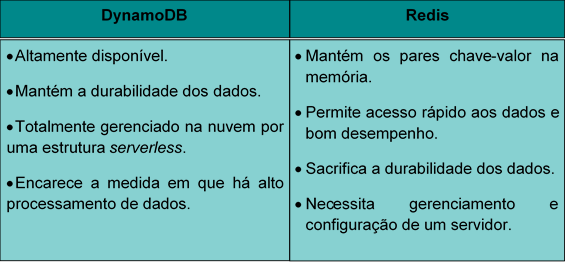
Portanto, não há uma preocupação com os dados a serem armazenados, como ocorre com os bancos de dados relacionais, em que são definidos os tipos de dados. O valor pode receber dados estruturados, semiestruturados ou ainda, não estruturados. Cabe ao aplicativo interpretar os dados recebidos (Sadalage; 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.

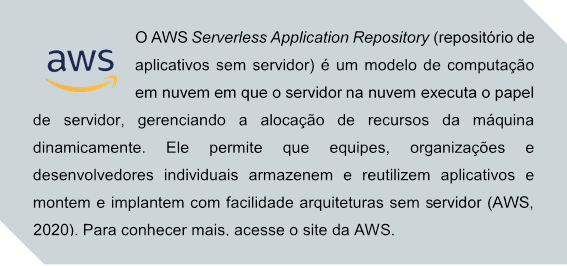
Quadro1 – Breve análise dos bancos de dados NoSQL DynamoDB e Redis.



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:



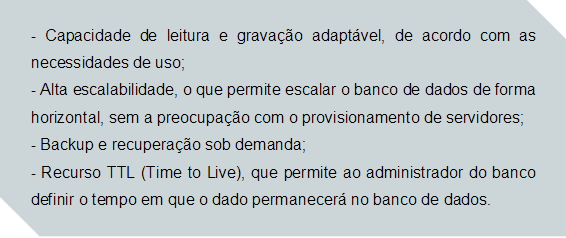
Em relação as suas características, o Dynamo suporta estrutura de dados chave-valor e documentos. Por ter uma estrutura *serverless*, todos os recursos necessários para o funcionamento arquitetural da aplicação são gerenciados pela própria nuvem da AWS, a qual possibilita disponibilidade global da aplicação. 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:



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:

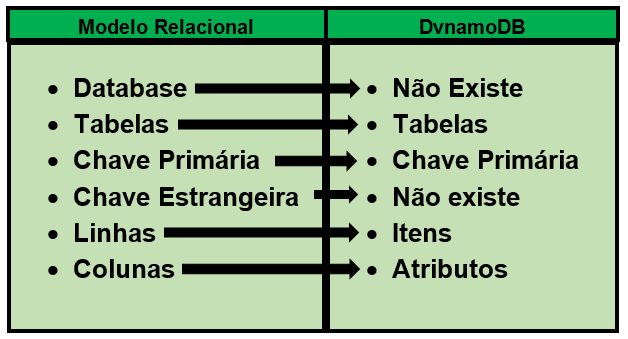
- Acesso pelo AWS Console;
- Acesso pelo AWS CLI (Command Line Interface), ou seja, por linha de comando;
- Acesso pelo AWS SDK, normalmente utilizado pelos desenvolvedores para acesso as APIs do DynamoDB, com alguma linguagem de programação específica, como Python, Ruby, Java etc.)


**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.

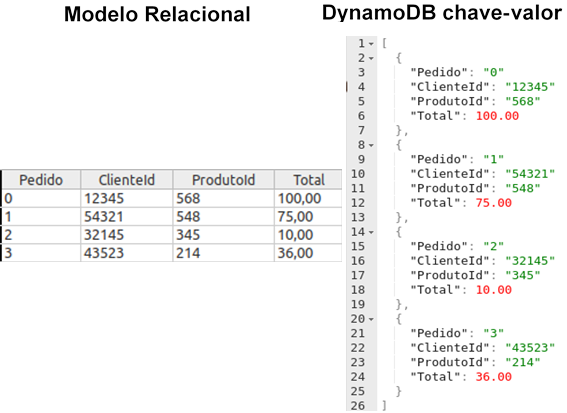
Para melhor compreensão e exemplificação do exposto no parágrafo anterior, no Quadro 2 é apresentada uma breve análise da comparação entre as terminologias do modelo de dados relacional e do DynamoDB.

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

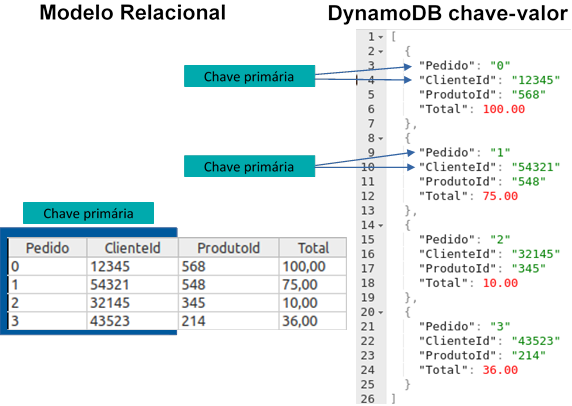


Para uma análise um pouco mais detalhada, a Figura 3 apresenta um exemplo de uma tabela de vendas de um banco de dados relacional de um sistema online de *e-commerce*, com uma simulação dos mesmos dados dessa tabela armazenados no DynamoDB, no formato chave-valor.

Figura 3 – Modelo de tabela relacional *versus* DynamoDB



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.

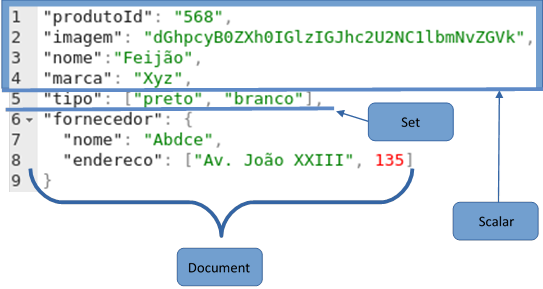
Exemplos de dados suportados pelo tipo Document Type:
- [“Faculdade”, 2020, true, “Curitiba”, 192, false]
- [ ]
- {{“nome”:“Faculdade”}}

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.*

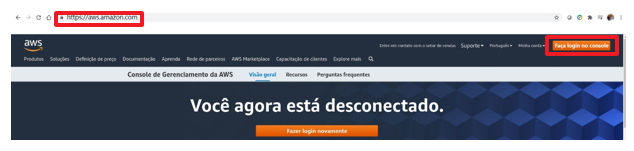
Figura 5 – Exemplo de aplicação dos tipos de dados



**TEMA 3 – ACESSANDO E CONFIGURANDO O BANCO DE DADOS DYNAMODB**

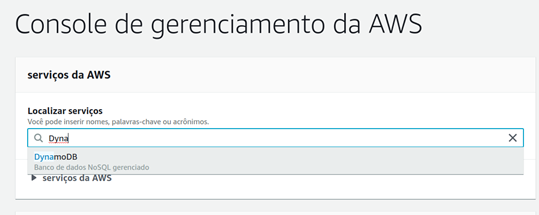
Conforme abordado nos tópicos anteriores, o DynamoDB por ser um banco de dados NoSQL totalmente gerenciado pela AWS, para configurá-lo primeiramente devemos acessar a página da Amazon: https://aws.amazon.com.

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.



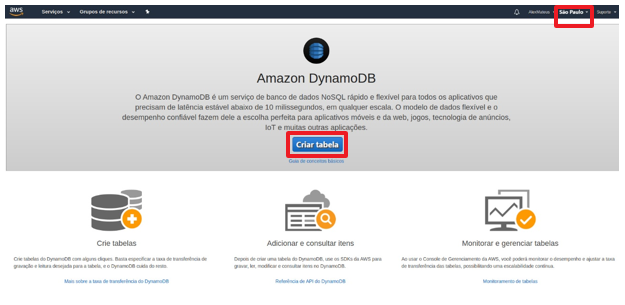
Após a conta criada, basta clicar no botão “**faça login no console**”, conforme destacado em vermelho na Figura 6, informar o login e senha cadastrados e acessar o console da AWS. A Figura 7 apresenta a interface inicial do console do AWS.

Figura 7 – Interface inicial da AWS



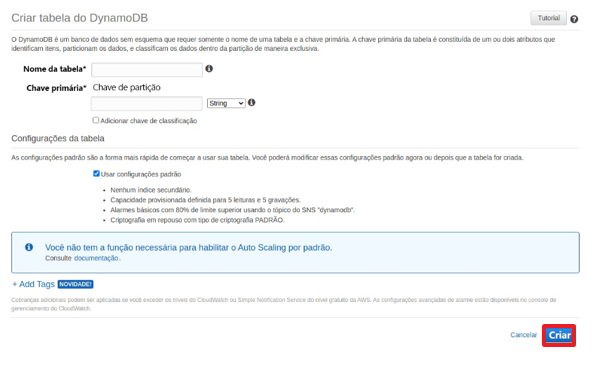
Para acessar o DynamoDB e criar as tabelas, basta digitar na caixa de seleção “Localizar serviços” o nome do banco de dados e selecionar o DynamoDB, conforme mostrado na Figura 7. Antes de criar a primeira tabela, é muito importante definir em qual região geográfica essa tabela será criada, pois em cada 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.

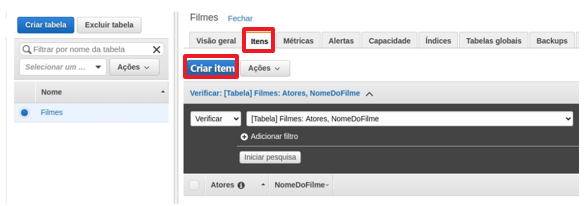
* **Nome da tabela:** Filmes
* **Chave primária:**
  + **Chave de partição:** Atores (tipo de dados String)
  + **Chave de classificação:** NomeDoFilme (tipo de dados String)
* **Usar configurações padrão:**Sim

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.

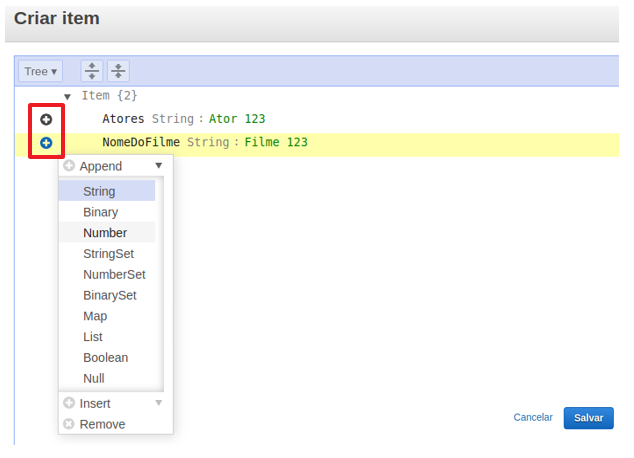
Para inserir registros ou novos atributos na tabela Filmes, basta clicar na aba “Itens” e em seguida no botão “Criar item”, conforme pode ser observado na Figura 10, destacado em vermelho.

Figura 10 – Interface da tabela Filmes no DynamoDB



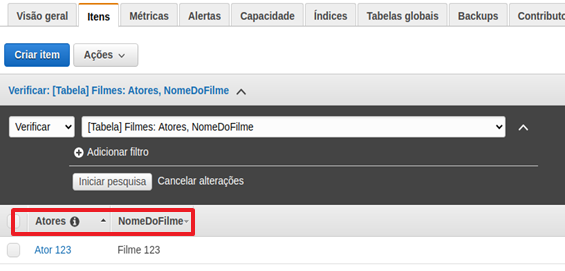
A Figura 11 apresenta a inserção dos dados “Ator 123” para o atributo “Atores” e, os dados “Filme 123” para o atributo “NomeDoFilme” da tabela “Filmes. Para a criação de novos atributos nessa tabela, basta clicar em um dos ícones destacados em vermelho ao lado de um dos atributos, escolher o tipo de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

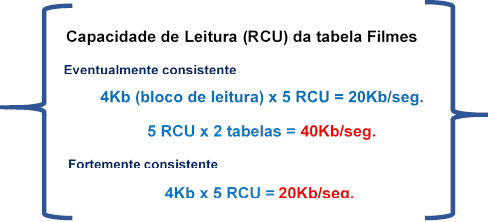
- 1 WCU equivale a uma gravação em uma tabela.

- Cada WCU corresponde a um bloco de 1Kb, ou seja, em cada gravação por segundo, é gravado um bloco de 1Kb de dados.


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.

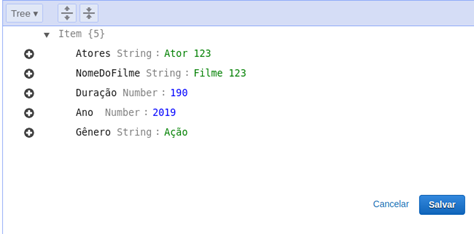


Agora vamos analisar qual é a capacidade máxima de leitura de dados da tabela Filmes, sabendo que ela possui uma capacidade provisionada de 5 RCUs.



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.

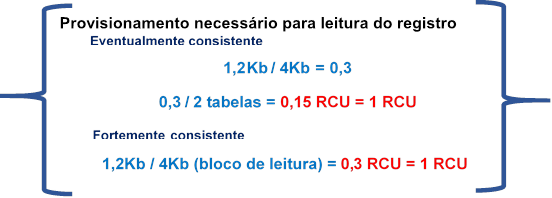
Figura 13 – Atributos da tabela Filmes



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:



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:



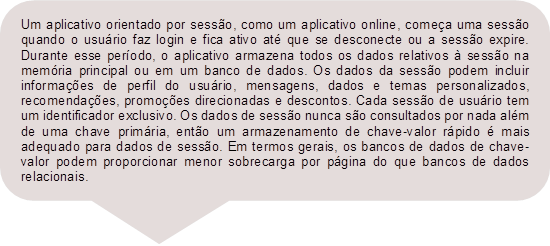
**4.3 CARACTERÍSTICAS DE ESCALABILIDADE**

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):



**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 NotesJ. 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.

**BANCO DE DADOS NOSQL**

AULA 3

Prof. Alex Mateus Porn

**CONVERSA INICIAL**

Olá! Nesta aula estudaremos sobre bancos de dados NoSQL orientados a documentos. Nosso objetivo é caracterizar os principais conceitos sobre o modo de armazenamento e gerenciamento de dados NoSQL orientado a documentos e estudar por meio de exemplos práticos uma ferramenta para criação e gerenciamento desses bancos de dados.

Iniciaremos analisando como são estruturados os bancos de dados orientados a documentos, o formato de um documento e como é o processo de modelagem nesse tipo de banco de dados. O objetivo aqui é compreender a estrutura dos documentos desse tipo de banco de dados para facilitar o entendimento dos exemplos práticos que abordaremos na sequência.

Como ferramenta para criação e gerenciamento dos bancos de dados orientados a documentos, estudaremos a ferramenta MongoDB, que atualmente é uma das mais conhecidas e utilizadas pelos profissionais da área. A aula será finalizada com a apresentação das principais aplicações dos bancos de dados NoSQL orientados a documentos. Vejamos os conteúdos que serão trabalhados:

Texto

Descrição gerada automaticamente com confiança média

**TEMA 1 – SISTEMAS NOSQL BASEADOS EM DOCUMENTOS**

Os bancos de dados NoSQL orientados a documentos podem ser considerados como os mais populares atualmente entre as quatro categorias existentes. Para Hurwitz (2016, p. 91) podem ser encontrados dois tipos de bancos de dados orientados a documentos, os já conhecidos repositórios para conteúdo em estilo de documento completo, como arquivos editores de texto, planilhas eletrônicas, páginas *web*, entre outros e, as bases de dados para armazenar componentes de documentos ou para um conjunto dinâmico de suas partes, as quais abordaremos nesta aula. Conforme Marquesone (2017, p. 47), esse modelo de banco de dados orientado a documentos é considerado uma extensão do banco de dados NoSQL orientado a chave-valor, oferecendo simplicidade e flexibilidade no gerenciamento dos dados e enriquecendo as possibilidades de consultas.

Para Hurwitz (2016, p. 91), bases de dados documentais são muito úteis quando é necessário produzir relatórios que precisam ser montados dinamicamente a partir de elementos que mudam com frequência. Um exemplo é o preenchimento de um documento da área da saúde, em que a composição do conteúdo varia com base no perfil do paciente, como idade, residência, renda, plano de saúde, elegibilidade para programas de governo.

Ao contrário dos bancos de dados relacionais que aplicamos o processo de normalização de dados, para evitar que os dados tenham valores duplicados nos bancos de dados NoSQL orientados a documentos esse processo é desconsiderado, assim como a criação de *joins* e esquemas. Conforme Marquesone (2017, p. 47) e Hurwitz (2016, p. 92), os documentos de um banco de dados orientado a documentos, podem ser definidos como estruturas flexíveis que podem ser obtidas por meio de dados semiestruturados, como os formatos XML (*eXtensible Markup Language*), JSON (*JavaScript Object Notation*) e BSON (*Binary JSON*).

Para nos aproximarmos um pouco mais da compreensão da estrutura de um banco de dados orientado a documentos, imaginemos um documento como sendo uma linha da tabela, e um conjunto de documentos como sendo a tabela com todos os registros. A diferença é que cada documento pode conter variações em sua estrutura (Marquesone, 2017, p. 47). Para compreender melhor esse formato, vamos analisar o exemplo apresentado na Figura 1, que destaca um documento no formato JSON para armazenamento de informações de um cadastro de clientes.

Figura 1 – Documento de cadastro de clientes no formato JSON



Conforme o exemplo, existem dois registros armazenados no documento “clientes”. Enquanto o cliente de nome “Cliente X” possui o atributo “dataNascimento” para armazenar a data completa, o cliente de nome “Cliente Y” possui o atributo “nascimento”, contendo uma lista dos atributos “dia”, “mes” e “ano”, para que os dados da data de nascimento sejam armazenados separadamente. Ambos os registros apresentados na Figura 1 são referentes aos dados dos clientes, porém é possível que cada registro contenha informações diferentes. A mesma situação ocorre para o atributo “celular” do registro do “Cliente X”, sendo que no registro do “Cliente Y”, esse atributo pertence a uma lista de atributos que constituem o atributo “contato”.

Diferentemente dos bancos de dados NoSQL orientados a chave-valor, que somente permitem que as consultas sejam realizadas pelos campos-chaves, de acordo com Marquesone (2017, p. 48) os bancos de dados NoSQL orientados a documentos permitem a criação de consultas e filtros sobre os valores armazenados. Assim, ainda seguindo o exemplo apresentado na Figura 1, podemos criar várias consultas, como por exemplo, listar os clientes por nome, endereço ou data de nascimento.

Desse modo, Marquesone (2017, p. 48-49), define que:

Bancos de dados orientados a documentos são ótimas soluções para armazenamento de registros conter todas as informações relevantes para uma consulta, sem necessitar da criação de joins.com atributos variados. Além disso, esses bancos de dados oferecem grande escalabilidade e velocidade de leitura, pois os dados são armazenados em forma desnormalizada. Por esse motivo, um documento armazenado deve conter todas as informações relevantes para uma consulta, sem necessitar da criação de joins.

Seguindo essa abordagem, Medeiros (2014) menciona que os bancos de dados orientados a documentos têm como características conter todas as informações importantes em um único documento, ser livre de esquemas, possuir identificadores únicos universais, possibilitar a consulta de documentos através de métodos avançados de agrupamento e filtragem (*MapReduce*) e também permitir redundância e inconsistência. Ainda conforme o autor, ao contrário dos bancos de dados relacionais, os bancos de dados orientados a documentos não fornecem relacionamentos entre os documentos, o que mantém seu *design* sem esquemas. Dessa forma, ao invés de armazenar os dados relacionados em uma área de armazenamento separado, os bancos de dados de documentos integram esses dados ao próprio documento. O Quadro 1 apresenta uma breve relação entre o modelo de dados relacional e os bancos de dados orientados a documentos.

Quadro 1 – Relação entre o modelo relacional e orientado a documentos



De acordo com o exemplo do Quadro 1, caso fossemos representar o documento de clientes da Figura 1 em uma tabela no modelo relacional, primeiramente seria necessário especificar um esquema para essa tabela, visto que existem diferenças na representação de alguns atributos para cada registro, como no caso do atributo “dataNascimento”. Supondo que o esquema definido seja conforme a representação de atributos utilizada para o “Cliente X”, ou seja:

* nome : String
* dataNascimento : Date
* endereco : String
* telefone : String
* celular : String

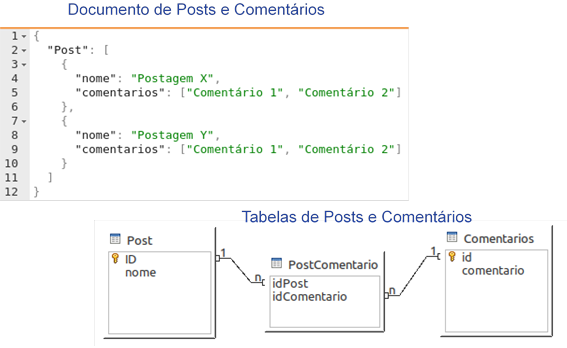
Havendo a necessidade de acrescentar mais colunas à tabela, torna-se necessária a alteração da própria tabela, pois as novas colunas serão adicionadas para todos os registros existentes. A esses registros serão atribuídos o valor nulo. Esse caso pode ser representado ao ser necessário adicionar o atributo “email”, conforme elencado no registro “Cliente Y” da Figura 1. Assim, ao registro “Cliente X” também será atribuído esse atributo, porém com valor nulo. A Figura 2 apresenta o registro de clientes da Figura 1 no modelo relacional.

Figura 2 – Representação de um documento no modelo relacional



Para Lennon (2011), manter os dados integrados ao próprio documento ao invés de armazená-los em uma área de armazenamento separada, funciona muito bem para aplicativos nos quais os dados ficam autocontidos dentro de um documento pai. Como exemplo podem ser citados os posts e comentários em blogs. Os comentários se aplicam somente a um único post, de modo que não faz sentido separá-los dele. Conforme o exemplo proposto pelo autor, no documento dos “posts de um blog” teria um atributo “nome” para registrar o nome do post e outro atributo “comentario” para armazenar os comentários referentes a cada post. Já em um banco de dados relacional, haveria uma tabela para armazenar os comentários, com uma chave primária, uma tabela de *posts* com uma chave primária e, uma tabela de mapeamento intermediária de *posts* e comentários para definir quais comentários pertencem a quais posts. A Figura 3 apresenta esse exemplo de modo gráfico.

Figura 3 – Representação do modelo de dados de documento e relacional



**1.1 MODELAGEM DE DADOS ORIENTADA A DOCUMENTOS**

De acordo com Monteiro (2019), o objetivo de modelar um banco de dados orientado a documentos é minimizar a quantidade de acessos ao banco de dados, para que a aplicação se torne mais rápida. À medida que a quantidade de dados aumenta podemos ter um conjunto de documentos que dará origem a uma coleção e, desse modo, um conjunto de coleções forma o banco de dados. Portanto, basicamente podemos ter dois formatos de modelagem, com documentos referenciados ou incorporados.

* **Modelagem incorporada (embedded)**: a modelagem incorporada refere-se a uma estrutura não normalizada, em que os dados normalmente são acessados juntos, como se fosse um documento dentro do outro (Monteiro, 2019).
* **Modelagem referenciada**:a modelagem referenciada estabelece em ter os documentos separados, mas um dos documentos tem a referência para o outro, de modo a minimizar a quantidade de dados duplicados, aumentar o desempenho nas operações de escrita, porém diminuir o desempenho nas consultas (Monteiro, 2019).

Na modelagem incorporada, a operação principal são as leituras, o que não impede de ser usada em aplicações com um grande volume de escritas. Para Monteiro (2019), no caso de escritas que envolvem muitas alterações nos dados, é preciso tomar cuidado, porque a informação que está sendo alterada pode estar replicada em vários documentos. Assim, a atualização dessa informação pode precisar ser feita em muitos lugares, sob pena de ter dados inconsistentes. A Figura 4 apresenta um exemplo da modelagem de dados orientada a documentos incorporada.

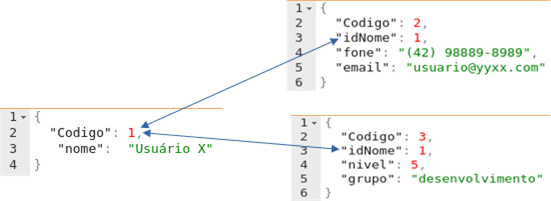
Figura 4 – Modelagem de dados orientada a documentos incorporada



Fonte: Adaptado de Monteiro, 2019.

Já na modelagem referenciada os documentos são armazenados separadamente, fazendo referência à abordagem do modelo relacional. Essa modelagem apresenta maior eficiência quando parte do documento é frequentemente lido/atualizado e a outra parte não, quando os dados não devem ser duplicados e quando um objeto é referenciado em muitos outros. A Figura 5 apesenta um exemplo da modelagem de dados orientada a documentos referenciada.

Figura 5 – Modelagem de dados orientada a documentos referenciada



Fonte: Adaptado de Monteiro, 2019.

**TEMA 2 – MONGODB**

O MongoDB, conforme Hows, Membrey e Plugge (2015, p. 16), é um banco de dados que não tem conceitos de tabelas, esquemas, SQL ou linhas. Não há transações, conformidade com as propriedades ACID de bancos de dados relacionais, *joins* ou chaves estrangeiras. Conforme Lennon (2011), trata-se de um banco de dados orientado a documentos e de software livre que armazena dados em coleções de documentos BSON, um formato semelhante ao JSON, ou seja, uma versão binária do formato JSON.

De acordo com Hows, Membrey e Plugge (2015, p. 72), apesar de o MongoDB ser conhecido como um banco de dados sem esquemas, isso não significa que sua estrutura seja completamente desprovida de esquemas, pois são necessárias as definições de coleções e índices durante a modelagem do banco de dados. Contudo, não é preciso predefinir uma estrutura para nenhum dos documentos que serão adicionados, pois o MongoDB é um banco de dados extraordinariamente dinâmico, de modo que diferentes tipos de documentos podem coexistir em uma mesma coleção. A Figura 6 apresenta um exemplo de dois documentos diferentes pertencentes a mesma coleção.

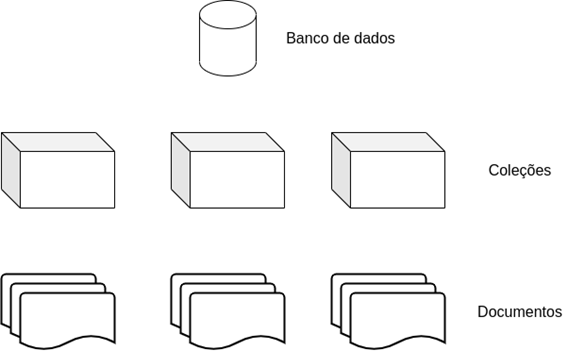
Figura 6 – Documentos diferentes em uma mesma coleção

Fonte: elaborado com base em Hows, Membrey e Plugge, 2015.

Um banco de dados orientado a documentos é constituído por um conjunto de documentos que formam uma coleção, e um conjunto de coleções constituem o banco de dados. Assim, uma coleção pode ser compreendida como um contêiner para armazenamento de documentos. A Figura 7 apresenta o modelo de um banco de dados MongoDB para representar o conceito de documentos, coleções e o banco de dados.

Figura 6 – Modelo de um banco de dados MongoDB



Fonte: elaborado com base em Hows, Membrey e Plugge, 2015.

**2.1 DOCUMENTOS E TIPOS DE DADOS SUPORTADOS**

Os bancos de dados orientados a documentos são considerados uma extensão dos bancos de dados orientados a chave-valor. Desse modo, os documentos são sempre constituídos de pares chave-valor. Relembrando o exemplo apresentado na Figura 6:

**Par: “tipo”: “CD”**

**Chave:** tipo

**Valor:** CD

De acordo com Hows, Membrey e Plugge (2015, p. 74-75), as chaves de um par chave-valor de um documento, são escritas na forma de *strings*, porém os valores podem variar de acordo com tipo de dados que precise ser armazenado. O Quadro 2 apresenta os tipos de dados básicos suportados pelo MongoDB e uma breve descrição de quando são utilizados.

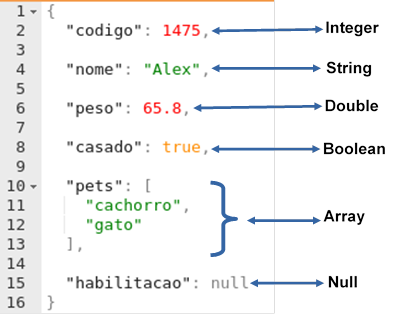
Quadro 2 – Tipos de dados básicos suportados pelo MongoDB

|  |  |
| --- | --- |
| **Tipos de dados** | **Finalidade** |
| String | Utilizado principalmente para armazenar valores textuais. Exemplo: {“universidade”: “Uninter”} |
| Integer | Utilizado para armazenar valores numéricos. Exemplo: {“idade”: 42} |
| Boolean | Utilizado para armazenar os valores “verdadeiro” ou “falso”. Exemplo: {“praticaEsporte”: true} |
| Double | Utilizado para armazenar valores de ponto flutuante. Exemplo: {“peso”: 73.5} |
| Arrays | Utilizado para armazenar arrays. Exemplo: “animais”: [“gato”, “cachorro”, “cavalo”] |
| Null | Utilizado para armazenar um valor nulo. Exemplo: “idade”: null |

Fonte: elaborado com base em Hows, Membrey e Plugge, 2015.

A Figura 8 apresenta um exemplo de um documento com os tipos de dados String, Integer, Boolean, Double, Array e Null.

Figura 8 – Tipos de dados



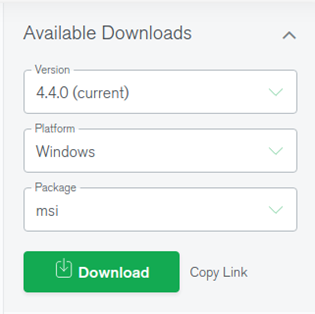
De acordo com Hows, Membrey e Plugge (2015, p. 79), todo documento criado no MongoDB possui um identificador chave único, definido como \_id. Caso essa chave \_id não seja adicionada manualmente com um valor específico ao criar o documento, o MongoDB adiciona automaticamente como sendo o primeiro atributo do documento, definindo um valor binário de 12 bytes.

**2.2 INSTALANDO E ACESSANDO O MONGODB**

Além do banco de dados DynamoDB, o banco de dados MongoDB também disponibiliza uma versão *offline* para *download* e instalação no computador e, uma versão online para criação do banco de dados na nuvem. Nesta aula focaremos na versão online do MongoDB. Porém, realizaremos o acesso e criação do banco de dados pelo *prompt* de comando no Windows ou pelo terminal (*shell*) no Linux, de modo a abordar também a versão *offline*, visto que os comandos são exatamente os mesmos.

Para *download* da versão *offline*, basta acessar a página de *downloads* do MongoDB: [<https://www.mongodb.com/try/download/community>](https://www.mongodb.com/try/download/community). Em seguida, é necessário escolher a versão do MongoDB, o sistema operacional utilizado e o tipo do pacote, conforme pode ser observado na Figura 9.

Figura 9 – Download do MongoDB



Após realizar o *download*, basta executar o arquivo de instalação e seguir o passo a passo do programa de instalação no Windows. Para usuários Linux, somente é necessária a execução do seguinte comando no terminal

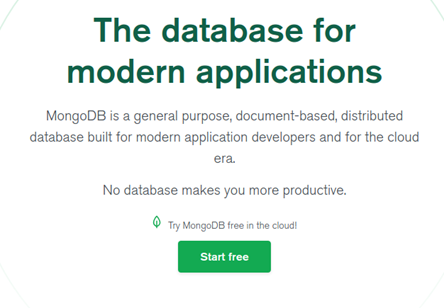
**sudo apt install mongodb**

Após a instalação, para executar o MongoDB basta digitar o comando “mongo” no Terminal do Linux ou, pelo Prompt de Comando do Windows acessar a pasta de instalação do MongoDB e digitar o mesmo comando.

**2.3 CONFIGURANDO O MONGODB**

Conforme mencionado, daremos ênfase à versão *on-line* do MongoDB. Portanto, o primeiro passo é acessar o seguinte endereço no seu navegador de internet: https://www.mongodb.com/. Após carregada a página do MongoDB, o primeiro passo é clicar na opção *Start Free*(iniciar gratuitamente), conforme destacado em verde na Figura 10.

Figura 10 – Página inicial do site do MongoDB

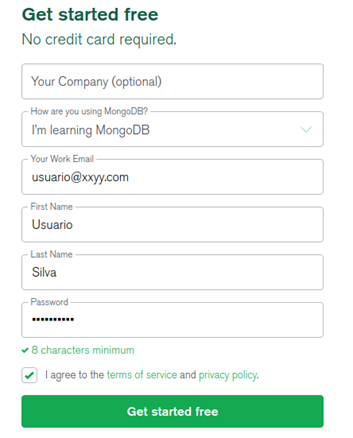


O próximo passo é preencher o formulário com as informações básicas solicitadas para o cadastro, sendo elas:

* Sua empresa (*Your company*): o preenchimento deste campo é opcional;
* Como você usará o MongoDB (*How are you using MongoDB*): selecionar a opção “*I’m learning MongoDB*” (Estou aprendendo o MongoDB);
* Seu e-mail (*Your Work E-mail*): preencher com seu e-mail pessoal;
* Primeiro nome (*First Name*): seu primeiro nome;
* Último nome (*Last Name*): seu sobrenome;
* Senha (*Password*): senha de sua escolha, com no mínimo 8 caracteres.
* Como último passo, deve ser selecionada a caixa de seleção “*I agree to the terms of service and privacy policy” (*Eu concordo com os termos de serviço e política de privacidade*)*.

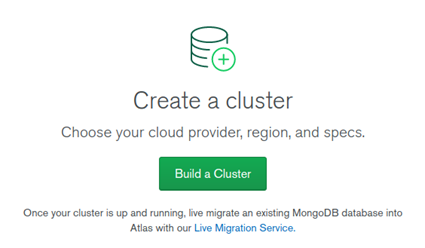
Todas as informações no *site* estão em inglês, não havendo uma versão em português. A Figura 11 apresenta um exemplo desse formulário.

Figura 11 – Formulário para criação da conta no MongoDB



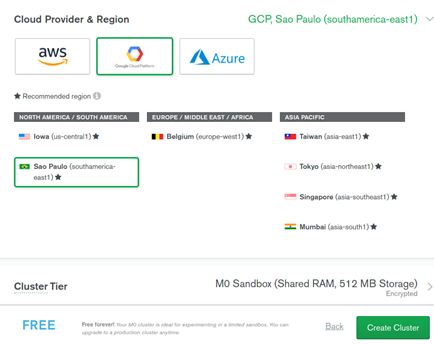
Após preencher o formulário e clicar no botão "*Get started free” (Iniciar*gratuitamente), conforme destacado em verde na Figura 11, será aberta a página da sua conta, conforme apresentado na Figura 12.

Figura 12 – Interface para criação do banco de dados no MongoDB



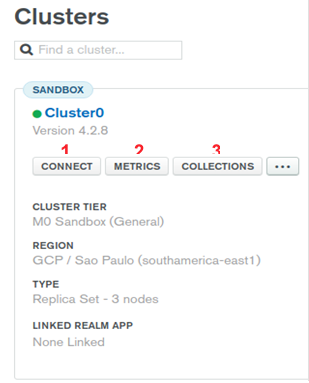
Após criar sua conta, o primeiro passo é clicar no botão “*Build a Cluster”*(Construir um Cluster), conforme destacado em verde na Figura 13. Na página seguinte, escolha a primeira opção, “*free*” (gratuita), para criar seu repositório de bancos de dados sem custos. O próximo passo é selecionar o servidor e a localidade do nosso Cluster e, clicar no botão “*Create Cluster”* (Criar Cluster) destacado em verde na Figura 13.

Figura 13 – Seleção do servidor e localidade do *cluster*



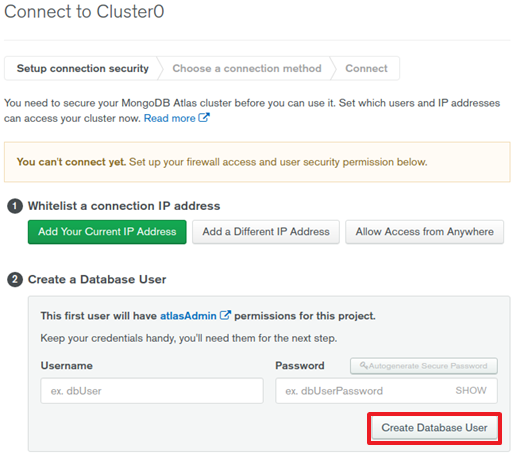
Após alguns minutos será criado o *cluster* e apresentada a interface conforme podemos observar na Figura 14.

Figura 14 – Interface com os dados do novo *cluster*



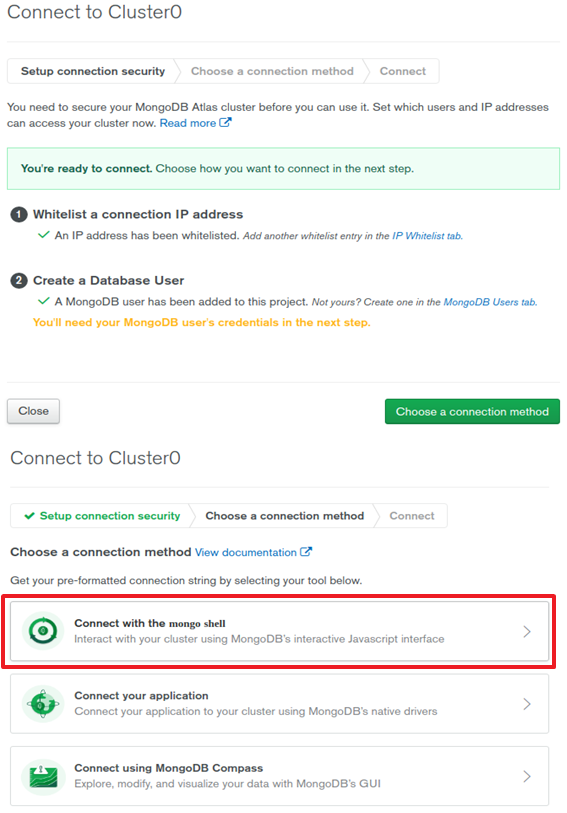
No botão *CONNECT*, indicado com o índice 1 na Figura 14, é possível criar uma conexão remota para o gerenciamento do MongoDB. No botão *METRICS,*indicado com o número 2, são apresentadas as métricas de uso do bando de dados e, no botão *COLLECTIONS*, destacado com o número 3, são listadas todas as coleções criadas para este cluster. Ao clicar no botão *CONNECT* é apresentada a interface mostrada na Figura 15 para configuração dos dados de acesso.

Figura 15 – Configuração dos dados de acesso ao Mongo DB.



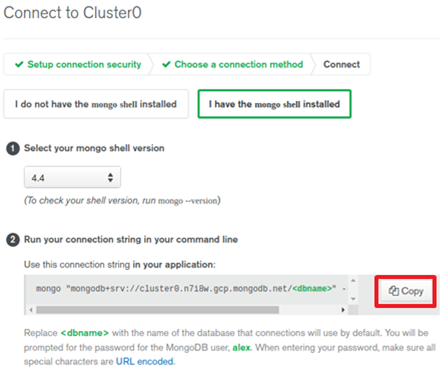
Na opção número 1 da Figura 15, é necessário clicar no botão destacado em verde “*Add Your Current IP Address”*(Adicione Seu Endereço de IP Atual), para adicionar o seu endereço IP de acesso ao MongoDB. Na opção número 2 é necessário informar um nome de usuário e uma senha de acesso ao banco de dados. Por fim, basta clicar no botão destacado em vermelho “*Create Database User”*(Criar Usuário do Banco de Dados) e, em seguida, em “*Choose a connection method”* (Escolher o método de conexão), conforme destacado em verde na Figura 16.

Figura 16 – Escolha do método de acesso



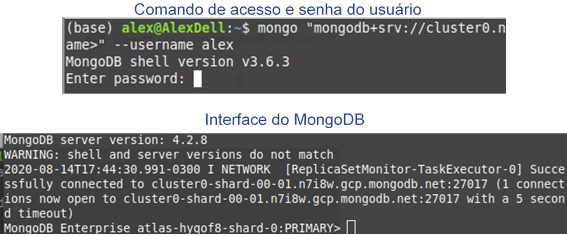
Conforme destacado em vermelho na Figura 16, selecionaremos a primeira opção, “*Connect with the mongo shell” (*Conectar com o Mongo shell). Na Figura 17 podemos observar como proceder.

Figura 17 – Conexão ao MongoDB pelo shell



Como realizamos no tópico 2.2 desta aula a instalação do MongoDB, basta selecionar a opção “*I have the mongo shell installed” (*Eu tenho o mongo shell instalado), destacada em verde na Figura 17, e em seguida clicar no botão “*Copy”*(Copiar), destacado em vermelho, para copiar o endereço de acesso conforme os dados de usuário que criamos no passo anterior. Agora, para acessar o MongoDB e criar nossos bancos de dados, basta abrir o terminal no Linux ou o *prompt* de comando no Windows, colar o endereço copiado, pressionar a tecla *Enter*, digitar a senha do usuário e, na sequência, estaremos conectados no MongoDB *on-line*, conforme pode ser observado na Figura 18.

Figura 18 – Conexão ao MongoDB



**TEMA 3 – OPERAÇÕES CRUD NO MONGODB**

A sigla CRUD refere-se a um acrônimo em inglês para as operações *create*, *read*, *update* e *delete*, ou seja, criar, ler, atualizar e excluir, sendo essas as quatro operações básicas de um banco de dados. O Quadro 3 apesenta uma breve comparação entre essas quatro operações em bancos de dados relacionais e no MongoDB.

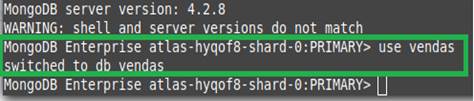
Quadro 3 – Operações CRUD no modelo relacional e MongoDB

Gráfico, Gráfico de mapa de árvore

Descrição gerada automaticamente

Após conectado ao MongoDB, o primeiro passo é criar o novo banco de dados. Para isso, utilizamos o comando **“*use*”** similarmente como fazemos nos bancos de dados relacionais para selecionar um banco de dados existente. Como exemplo, criaremos um banco de dados chamado “vendas”. Para isso, basta digitar o comando**“use vendas”**.

Figura 19 – Criação do banco de dados no MongoDB



Para visualizar o nome do banco de dados que estamos trabalhando atualmente, basta digitarmos o comando**“db.getName()”**. Porém, se em seguida digitarmos o comando **“show dbs”** para listar todos os bancos de dados existentes no MongoDB, nosso banco de dados “vendas” ainda não será listado, pois ele somente será efetivamente criado, após a inserção do primeiro registro.

**3.1 OPERAÇÃO INSERT**

Ao contrário da inserção de dados em tabelas nos bancos de dados relacionais, no MongoDB inserimos os documentos em coleções, conforme vimos no início desta aula. Portanto, o MongoDB oferece os seguintes métodos para inserir documentos em coleções.

**db.coleção.insertOne()**

**db.coleção.insertMany()**

  Para inserir um registro de um cadastro de clientes em uma coleção denominada “clientes”, basta digitarmos o seguinte comando:

**db.clientes.insertOne (**

**{**

**nome: “Alex”,**

**endereco: “Rua das avenidas, 288”,**

**telefone: “42-1111-2222”,**

**idade: 18**

**3.2 OPERAÇÃO FIND**

A operação de leitura, ou seja, consulta aos dados em uma coleção no MongoDB, é realizada pela operação ***find***, similar a operação *select* realizada nos bancos de dados relacionais. Para localizar os dados em uma coleção no MongoDB, digite o comando**db.coleção.find()**. Para listar o documento inserido na coleção clientes no tópico anterior, digite**db.clientes.find()**.

Com o objetivo de filtrar o retorno dos dados, podemos utilizar filtros específicos, como por exemplo, listar todos os clientes com 18 anos**db.clientes.find({idade: 18})**. Para realizar operações lógicas, como listar os clientes com idade maior ou menor do que 18, o MongoDB disponibiliza algumas funções específicas:

* gt (greater than) – maior que;
* gte (greater than or equals) – maior ou igual à;
* lt (less than) – menor que;
* lte (less than or equals) – menor ou igual à.

Então, para localizar os clientes maiores de 18 anos, utilizamos o seguinte comando:**db.clientes.find({idade: {$gt: 18}})**.

**3.3 OPERAÇÃO UPDATE**

Para alterar os dados de um documento, no MongoDB utilizamos a **update**, similar como realizamos nos bancos de dados relacionais. Para editar os dados em uma coleção no MongoDB, digite o seguinte comando:

**db.coleção.updateOne()**

**db.coleção.updateMany()**

Para editar a idade de um cliente na coleção “clientes”, basta digitar o comando**db.clientes.updateOne({nome: “Alex”}, {$set: {idade: 25}})**. No primeiro parâmetro (nome: “Alex”), é indicada a chave de busca para o registro e, o comando **set** define o atributo que será modificado (idade: 25).

**3.4 OPERAÇÃO DELETE**

Para realizar operações de exclusão de documentos, o MongoDB oferece duas opções:

**db.coleção.deleteOne()**

**db.coleção.deleteMany()**

Assim, para excluir os registros de clientes que possuem 25 anos, podemos utilizar o comando**db.clientes.deleteMany({idade: 25})**.

**TEMA 4 – CARACTERÍSTICAS DE CONSISTÊNCIA, TRANSAÇÕES E DISPONIBILIDADE**

Da mesma forma que a AWS está presente em várias regiões do mundo para a disponibilidade do DynamoDB, o MongoDB também está presente em vários lugares garantindo a disponibilidade do serviço através de cada região. Desse modo, conforme ocorre no DynamoDB, os documentos armazenados no MongoDB também são sincronizados entre os data centers da região escolhida.

**4.1 CARACTERÍSTICAS DE TRANSAÇÕES**

No MongoDB, quando é realizada uma operação em um único documento, essa operação é atômica. Conforme MongoDB (2018), como podemos utilizar documentos e matrizes incorporadas para capturar relacionamentos entre dados em uma única estrutura de documento em vez de normalizar em vários documentos e coleções, essa atomicidade de documento único elimina a necessidade de transações de vários documentos para muitos casos de uso práticos. Para situações que exigem atomicidade de leituras e gravações em vários documentos, o MongoDB oferece suporte a transações de vários documentos. Com transações distribuídas, as transações podem ser usadas em várias operações, coleções, bancos de dados e documentos (MongoDB, 2018).

**4.2 CARACTERÍSTICAS DE DISPONIBILIDADE**

Conforme Boaglio (2015, p. 149-150), alta disponibilidade é a possibilidade de ter os dados replicados em diferentes lugares, denominados como nodos, e se um nodo cair, outro assume o seu lugar, evitando assim que a aplicação deixe de funcionar. Essa arquitetura é conhecida como *replica set*, onde um nodo é o primário, de onde os dados são lidos e escritos e, os demais são os nodos secundários, em que os dados são copiados do nodo primário para serem utilizados para consulta.

Ainda de acordo com Boaglio (2015, p. 150), caso o nodo primário falhe, um dos nodos secundários assume para ser o novo nodo primário. Novos nodos secundários podem ser adicionados a qualquer instante, sem interromper o cluster inteiro. Para verificar a disponibilidade de cada nodo, um recurso denominado *heartbeat* é utilizado, fazendo com que a cada dois segundos os nodos conversem para verificar se estão ativos.

**4.3 CARACTERÍSTICAS DE CONSISTÊNCIA**

De acordo com Duarte (2017), o MongoDB garante conformidade com as propriedades ACID como documento. Se os dados estiverem espalhados entre diversos documentos, não há garantia alguma da consistência e integridade entre esses dados. Nesse sentido, a atomicidade também pode ser comprometida caso os dados estejam espalhados entre diversas coleções.

**TEMA 5 – CASOS DE USO APROPRIADOS**

Para Marquesone (2017, p. 49), os bancos de dados NoSQL orientados a documentos são indicados para realizar o armazenamento de conteúdo de páginas *web*, na catalogação de documentos de uma empresa e no gerenciamento de inventário de um *e-commerce*, pois são aplicações que trabalham diretamente com coleções de documentos e, portanto, podem se beneficiar da flexibilidade que o armazenamento orientado a documentos oferece. Além desses cenários, o autor destaca que esse modelo de banco de dados também pode ser muito útil em demais aplicações que utilizam estruturas de dados no formato JSON e que se beneficiam da desnormalização das estruturas dos dados.

Conforme a AWS (2020), os bancos de dados orientados a documentos são ótimas opções para aplicativos de gerenciamento de conteúdo, como *blogs* e plataformas de vídeo, de modo que se o modelo de dados precisar mudar, somente os documentos afetados precisarão ser atualizados e nenhuma atualização de esquema é exigida e nenhum tempo de inatividade de banco de dados é necessário para fazer as alterações.

Esses bancos de dados são eficientes e eficazes para o armazenamento de informações de catálogo. Conforme a AWS (2020), em um aplicativo de comércio eletrônico, diferentes produtos costumam ter números de atributos diferentes, e gerenciar milhares de atributos em bancos de dados relacionais é ineficiente e afeta a performance de leitura. Ao usar um banco de dados de documentos, os atributos de cada produto podem ser descritos em um único documento para gerenciamento fácil e maior velocidade de leitura. Alterar os atributos de um produto não afetará os outros.

**FINALIZANDO**

Nesta aula estudamos os conceitos, implementações e aplicações de uso de bancos de dados NoSQL orientados a documentos. No primeiro tema, vimos como são estruturados esses bancos de dados, como é composto um documento no formato JSON e também fizemos analogia entre os componentes de um banco de dados relacional e o modelo orientado a documentos.

No tema 2, abordamos a ferramenta de construção e gerenciamento de bancos de dados orientados a documentos MongoDB, uma das principais e talvez a ferramenta mais conhecida e utilizada no gerenciamento desses bancos de dados. Pudemos conhecer nesse tópico como criar uma conta na *nuvem* do MongoDB e como criar o nosso primeiro acesso através do terminal, tanto na nuvem como em uma versão instalada localmente em nosso computador.

Já no tema 3 nos aprofundamos nas operações para criação dos bancos de dados, inserção, atualização e remoção de documentos, com alguns exemplos práticos, e sempre tentando fazer comparações com as operações do modelo de dados relacional, para nos aprimorarmos a partir de conhecimentos já consolidados. Encerramos esta aula com uma breve análise das características de transações, disponibilidade e consistência do MongoDB, no tema 4, e com as principais áreas de aplicação dos bancos de dados NoSQL orientados a documentos, no tema 5.

**REFERÊNCIAS**

AWS. **O que é um banco de dados de documentos?** 2020. Disponível em: <https://aws.amazon.com/pt/nosql/document/>. Acesso em: 28 abr. 2021.

BOAGLIO, F. **MongoDB**: Construa novas aplicações com novas tecnologias. São Paulo: Casa do Código, 2015.

DUARTE, L. **Boas práticas com MongoDB**. 22 de set. 2017. Disponível em: <https://blog.umbler.com/br/boas-praticas-com-mongodb/>. Acesso em: 28 de abr. 2021.

HOWS, D.; MEMBREY P.; PLUGGE, E. **Introdução ao MongoDB**. São Paulo: Novatec, 2015.

HURWITZ, J. et al. **Big Data para Leigos**. Rio de Janeiro: Alta Books, 2016.

LENNON, J. **Explore o MongoDB**: Saiba por que esse sistema de gerenciamento de bancos de dados é tão popular. 11 de jul. 2011. Disponível em: <https://www.ibm.com/developerworks/br/library/os-mongodb4/os-mongodb4-pdf.pdf>. Acesso em: 28 abr. 2021.

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

MEDEIROS, H. **Introdução ao MongoDB**. 2014. Disponível em: <https://www.devmedia.com.br/introducao-ao-mongodb/30792#Banco>. Acesso em: 28 abr. 2021.

MONGODB **Manual**. 2018. Disponível em: <https://docs.mongodb.com/manual/introduction/>. Acesso em: 28 abr. 2021.

MONTEIRO, D. **Introdução para modelagem de dados para banco orientado a documentos**. 18 de abr. 2019. Disponível em: <https://imasters.com.br/banco-de-dados/introducao-para-modelagem-de-dados-para-banco-orientado-documentos>. Acesso em: 28 abr. 2021.

**BANCO DE DADOS NOSQL**

AULA 4

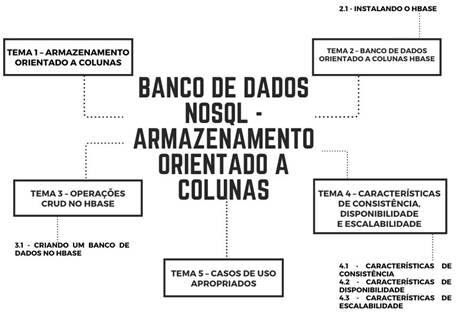
Prof. Alex Mateus Porn

**CONVERSA INICIAL**

Nesta aula abordaremos bancos de dados orientados a colunas, com o objetivo principal de introduzir os principais conceitos sobre o modo de armazenamento e gerenciamento de dados orientados a colunas e apresentar na prática um dos principais sistemas gerenciadores de bancos de dados (SGBD) orientados a colunas, o HBase.

Iniciaremos com uma explanação sobre a estrutura de armazenamento orientada a colunas e as principais definições e conceitos desse tipo de banco de dados. Você aprenderá como os dados são estruturados no tipo orientado a colunas e como são armazenados nas tabelas. Após compreender esse modelo de dados, aplicaremos todos os conceitos de forma prática no SGBD HBase, o que possibilitará compreender todo o processo de criação do banco de dados, inserção, edição e criação de consultas aos dados.

Encerraremos com a apresentação das principais aplicações de bancos de dados NoSQL orientados a colunas e ao longo desta aula trabalharemos os seguintes conteúdos:



**TEMA 1 – ARMAZENAMENTO ORIENTADO A COLUNAS**

O armazenamento de dados orientado a colunas corresponde a uma das quatro principais estruturas de armazenamento de dados NoSQL. Como o próprio nome sugere, ela é composta pela criação de colunas e linhas, possuindo conceitos muito similares aos bancos de dados relacionais. Por outro lado, o armazenamento de dados NoSQL orientado a colunas também pode ser considerado como o mais complexo entre as quatro estruturas.

Para melhor compreender essa estrutura de linhas e colunas dos bancos de dados NoSQL orientados a coluna e as similaridades com os bancos de dados relacionais, vamos fazer uma breve análise e comparação entre essas duas estruturas de armazenamento de dados. Conforme destaca Marquesone (2017, p. 50), para que o armazenamento em um banco de dados relacional ocorra precisamos definir antecipadamente a estrutura da tabela, indicando suas colunas e tipos de dados. Por exemplo, podemos criar uma tabela simples para registro de clientes contendo as seguintes colunas (atributos):

* id\_cliente : int;
* nome : varchar(200);
* data\_nascimento : date;
* telefone : varchar(30);
* cpf : varchar(11)

Veja na Figura 1 a estrutura dessa tabela em um banco de dados relacional.

Figura 1 – Tabela para registro de clientes em um banco de dados relacional

|  |  |
| --- | --- |
| **CLIENTES** | |
| id\_cliente | int |
| nome | varchar(200) |
| nascimento | date |
| telefone | varchar(30) |
| renda | decimal(5,2) |

Fonte: elaborado om base em Marquesone (2017, p. 50).

Marquesone (2017, p. 50) destaca que o seguinte:

Essa estrutura de armazenamento pode trazer diversas limitações. Por exemplo, se esta tabela tem como objetivo armazenar as preferências dos usuários em um aplicativo de compras online, podem haver usuários que gravarão apenas os dados obrigatórios, enquanto outros poderão gravar inúmeras outras informações, como preferência de roupas, cosméticos, sapatos e livros.

A limitação ocorre, pois conforme aborda Marquesone (2017, p. 50), uma vez que definimos essa estrutura, todos os registros de clientes que gravarmos nesse banco deverão conter essas cinco colunas, mesmo que algumas fiquem preenchidas com NULL. O SGBDR armazenará e recuperará os dados uma linha por vez, sempre que realizarmos uma consulta.

Ainda nesse cenário, Marquesone (2017, p. 51) aponta como outro fator de limitação de uma estrutura de banco de dados relacional, o tempo de execução de uma consulta à medida que banco de dados cresce. Conforme o autor, se a quantidade de dados armazenados chega à escala de *terabytes*, mesmo se for realizada uma consulta para buscar um único campo da tabela, o banco de dados relacional precisará passar por todos os registros de todas as linhas para trazer os resultados, impactando o desempenho da consulta. O mesmo impacto ocorre ao ter de reestruturar todos os registros já armazenados na tabela para cada inclusão de um novo campo.

Diante dessa problemática, os bancos de dados orientados a colunas buscam resolver principalmente o problema de escalabilidade e flexibilidade no armazenamento de dados. Assim, Marquesone (2017, p. 51) define que

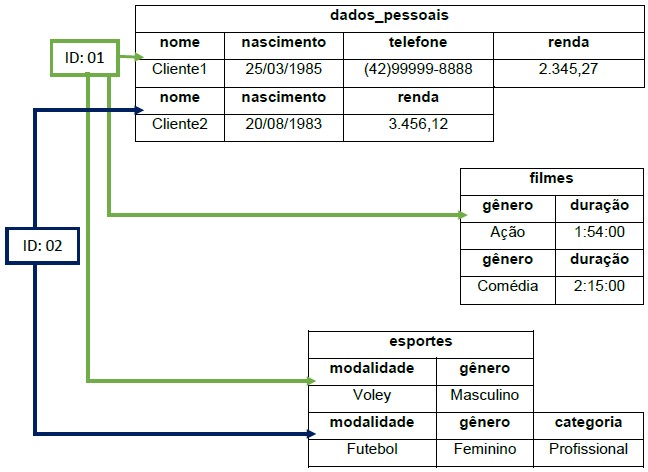
no que se refere à flexibilidade, ao invés de definir antecipadamente as colunas necessárias para armazenar um registro, o responsável pela modelagem de dados define o que é chamado de "famílias de colunas". As famílias de colunas são organizadas em grupos de itens de dados que são frequentemente usados em conjunto em uma aplicação.

Para Marquesone (2017, p. 51), no cenário para armazenamento das preferências do usuário em um aplicativo de compras online, utilizando um banco de dados orientado a colunas, poderia ser definido pelo menos três famílias de colunas:

* dados\_pessoais;
* filmes;
* esportes.

Para cada uma das famílias de colunas propostas, essa estrutura de banco de dados permite a flexibilidade de inserir quantas colunas forem necessárias para cada registro armazenado, sem precisar alterar a estrutura dos dados já armazenados. A Figura 2 apresenta o modelo do banco de dados orientado a colunas para atender ao cenário do armazenamento dos dados das preferências dos usuários de um aplicativo de compras *on-line*.

Figura 2 – Modelo de dados orientado a colunas

Fonte: elaborado com base em Marquenose (2017, p. 52).

Tendo como base a abordagem apresentada na Figura 1, podemos perceber que ao contrário dos bancos de dados relacionais, nos bancos de dados NoSQL orientados a colunas o número de colunas pode ser diferente para cada registro. Nesse contexto, Marquesone (2017, p. 52) afirma que

com essa estratégia de armazenamento por famílias de colunas, além de fornecer flexibilidade, esse modelo oferece também grande escalabilidade. O registro de um item pode ter informações gravadas em diversas famílias de colunas, que podem estar armazenadas em diferentes servidores. Isso é possível pelo fato de que os dados são armazenados fisicamente em uma sequência orientada a colunas e não por linhas.

Destacando ainda Marquesone (2017, p. 52-53), enquanto no banco de dados relacional o registro seria armazenado na sequência: Cliente1, 25/03/1985, (42)99999-8888, ..., no banco de dados orientado a colunas a sequência seria: Cliente1, Cliente2, 25/03/1985, 20/08/1983, .... Para esse último cenário, utilizam-se identificadores de linhas e colunas como chave para consultar os dados. Parafraseando Barroso (2012), uma consulta do tipo ***select avg(renda) from clientes***, em um banco de dados relacional, irá recuperar todas as linhas, carregando todos os campos para executar a operação e retornar a média salarial dos clientes. Já no banco de dados orientado a colunas, apenas a coluna “renda” será avaliada, consumindo assim menos recursos. Porém, uma consulta do tipo ***select \* from people***possivelmente não apresentará benefícios, já que todas as colunas precisarão ser lidas.

**TEMA 2 – BANCO DE DADOS ORIENTADO A COLUNAS HBASE**

O HBase é um banco de dados orientado a colunas utilizado principalmente quando há grande quantidade de dados e tabelas extensas com muitos atributos e dados armazenados. Esse banco de dados foi desenvolvido para funcionar sobre o Hadoop. Por esse motivo, antes de aprender sobre o HBase, precisamos brevemente compreender o que é o Hadoop, que consiste em uma plataforma utilizada para realizar projetos distribuídos que tratam de grandes quantidades de dados. Trata-se de um ecossistema para gerenciamento de bancos de dados distribuídos, composto de:

* Sistema de arquivos – Hadoop File System (HDFS);
* Sistema de processamento de dados – Hadoop MapReduce;
* Sistema de gerenciamento de banco de dados (SGBD) – HBase.

Como podemos observar nessa breve análise, o HBase corresponde à um sistema gerenciador de bancos de dados distribuído orientado a colunas, responsável pelo gerenciamento dos dados no sistema de arquivos Hadoop. Porém, o HBase disponibiliza uma versão para gerenciamento do banco de dados local (*standalone*), não sendo necessária a instalação do Hadoop. Como o escopo dessa disciplina é criação, configuração e gerenciamento de bancos de dados NoSQL, não tendo como foco sistemas distribuídos, abordaremos a versão *standalone* do HBase.

Outra característica do HBase é que ele foi desenvolvido nas versões para Linux e Mac, sendo disponibilizadas atualmente adaptações para o sistema operacional Windows.

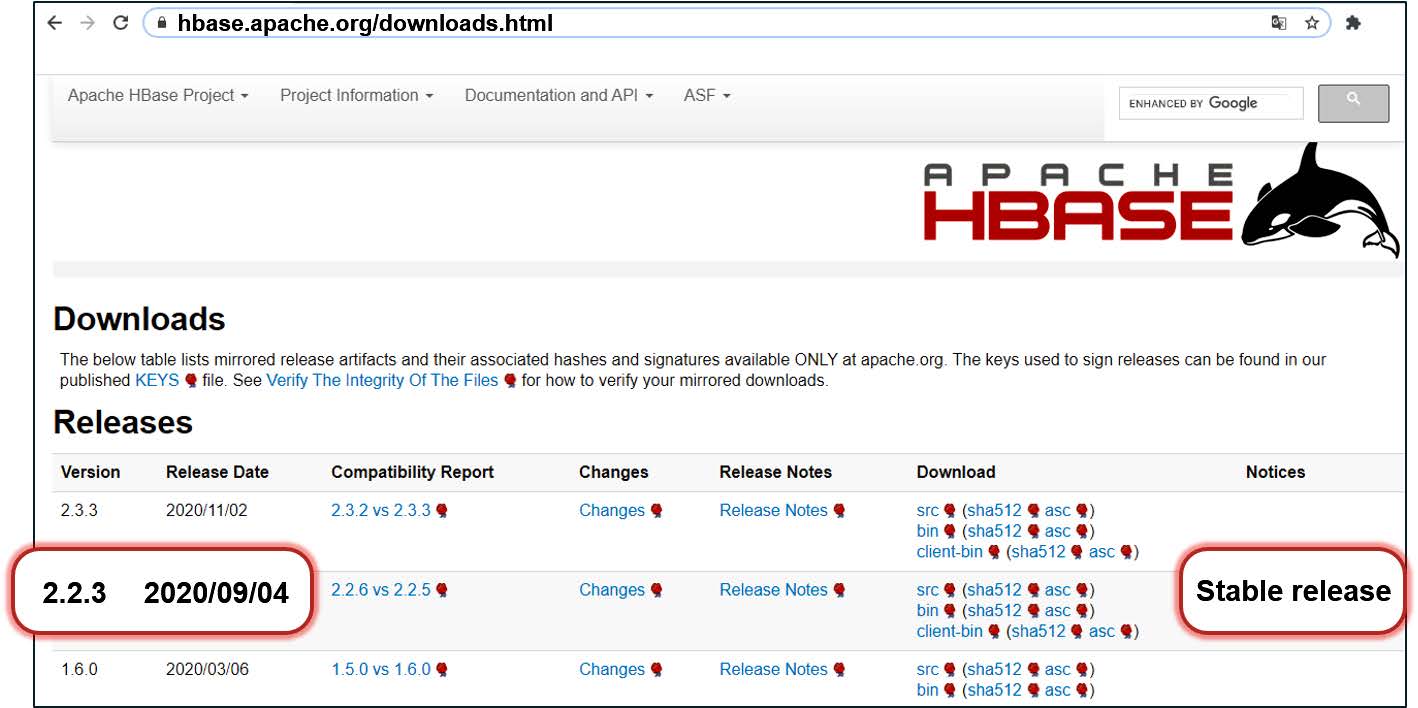
Conforme pode ser consultado em: [<https://www.learntospark.com/2020/08/setup-hbase-in-windows.html>](https://www.learntospark.com/2020/08/setup-hbase-in-windows.html).

Nesta aula abordaremos a instalação e configuração original do HBase para Linux.

**2.1 INSTALANDO O HBASE**

Para instalar o HBase, o primeiro passo consiste em fazer o *download* do SGBD. Para isso, acesse o link [<https://hbase.apache.org/downloads.html>](https://hbase.apache.org/downloads.html) e na página de *downloads* procure pelas versões estáveis (*stable*) do HBase. Atualmente a versão *stable* corresponde à versão 2.2.6.

Figura 3 – Página para download do HBase



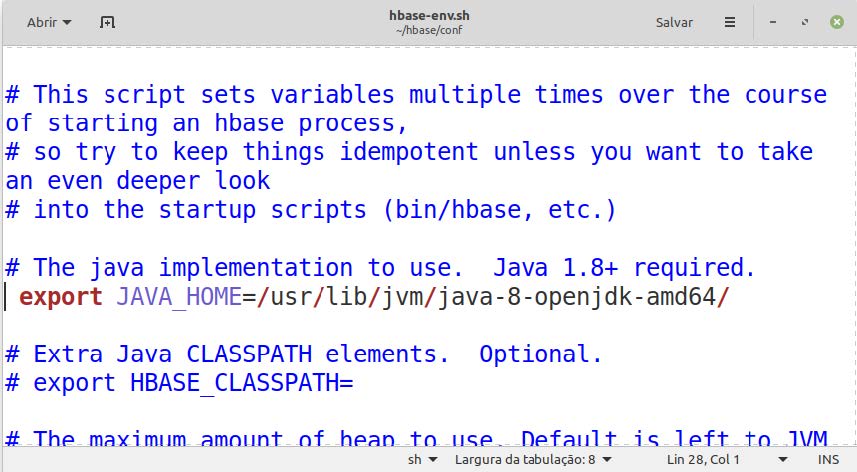
Após o *download* do HBase, devemos descompactar o arquivo baixado em um diretório específico. Recomenda-se criar um diretório denominado hbase dentro do diretório /home do usuário do sistema operacional. Depois de descompactar o arquivo de *download* do HBase dentro do novo diretório, devemos configurar dois arquivos que se encontram dentro da pasta conf, sendo os arquivos hbase-env.sh e hbase-site.xml. Na sequência é apresentado um exemplo do endereço de localização desses dois arquivos:

1. /home/usuário/hbase/conf/hbase-env.sh

2. /home/usuário/hbase/conf/hbase-site.xml

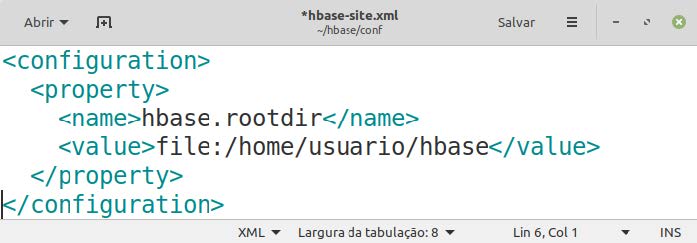
No arquivo hbase-env.sh devemos configurar o endereço da variável de ambiente JAVA\_HOME. Portanto, caso ainda não tenha instalado o Java JDK em seu computador, é necessário acessar a página de *downloads* do JDK em [<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>](https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html) e instalá-lo em seu computador. O endereço da variável de ambiente JAVA\_HOME normalmente se localiza em /usr/lib/jvm/java-versão-ope.jdk. Porém, pode variar de uma distribuição para outra do Linux, sendo necessário uma breve pesquisa para identificar qual a localização do JDK no seu Linux. A Figura 4 apresenta um exemplo da configuração da variável de ambiente JAVA\_HOME no arquivo hbase-env.sh

Figura 4 – Configuração do arquivo hbase-env.sh

Fonte: hbase-env.sh.

O próximo passo consiste na configuração do arquivo hbase-site.xml, em que ficam as configurações do HBase. Como estamos realizando a configuração *standalone*, podemos apagar todo o conteúdo desse arquivo e manter uma configuração básica entre as tags <configuration> e </configuration>, conforme mostrado na Figura 5.

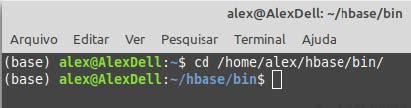
Figura 5 – Configuração do arquivo hbase-site.xml

Fonte: hbase-site.xml.

A configuração entre as tags <property> e </property> corresponde ao local em que serão armazenados os dados. Na configuração *standalone* não é necessário instalar o Hadoop, não sendo, portanto, instalado o sistema de arquivos HDFS. Nesse caso, a linha **<value>file:/home/usuario/hbase</value>**corresponde ao endereço de armazenamento dos dados do HBase no sistema de arquivos local do seu sistema operacional. Lembrando que “usuario” corresponde ao nome do diretório do usuário logado no sistema operacional.

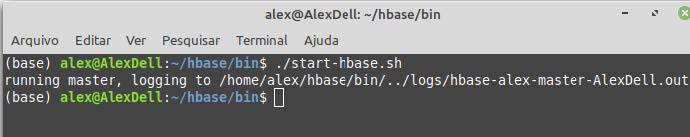
Finalizadas as configurações desses dois arquivos, o próximo passo consiste em inicializar o serviço do HBase e testar se tudo está funcionando corretamente. Para isso, abra o terminal do Linux e navegue até o diretório “bin” localizado dentro da pasta em que foi descompactado o arquivo de download do HBase. Na sequência é apresentado um exemplo dessa tarefa: cd /home/usuario/hbase/bin.

Figura 6 – Acesso ao diretório do HBase pelo terminal



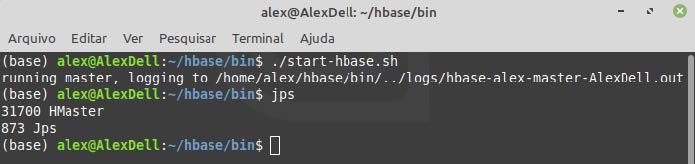
Após acessar a pasta “bin”, basta digitarmos o seguinte comando para inicializarmos o serviço do HBase ./start-hbase.sh. Se tudo foi configurado corretamente conforme explicitado, a seguinte mensagem deve ser apresentada: running máster, logging to /home/usuario/hbase/bin/../logs/hbase-usuario-master-pc.out. A Figura 7 apresenta um exemplo da execução do HBase.

Figura 7 – Inicialização do HBase



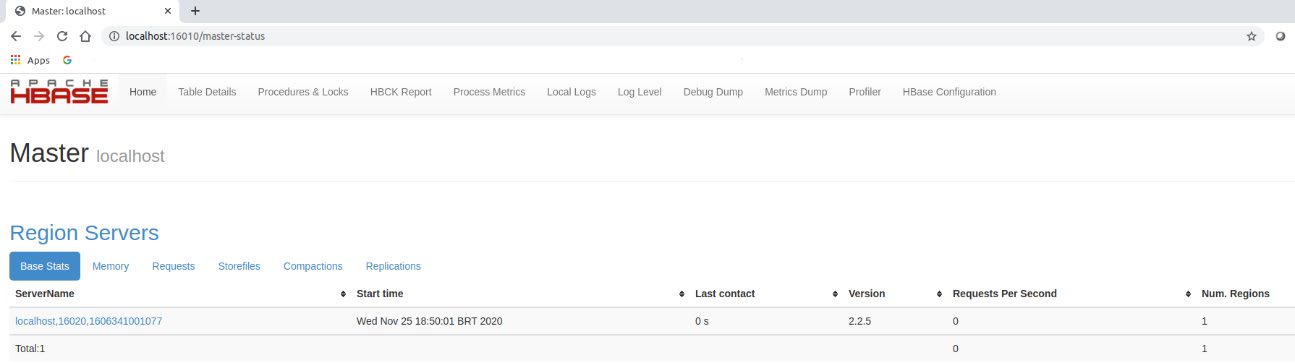
Para conferir se o HBase iniciou corretamente e o servidor máster está ativo, podemos usar o comando jps no terminal, conforme apresentado na Figura 8; deverá ser apresentado o processo HMaster, que corresponde a nossa instância local do HBase.

Figura 8 – Visualização das instâncias do HBase.



Outro modo de conferir se o HBase está executanto perfeitamente e também visualizar os bancos de dados criados, é acessar a interface gráfica pelo navegador web através do seguinte endereço [<http://localhost:16010>](http://localhost:16010/). Se tudo estiver configurado corretamente, deverá ser carregada uma interface gráfica conforme o exemplo mostrado na Figura 9.

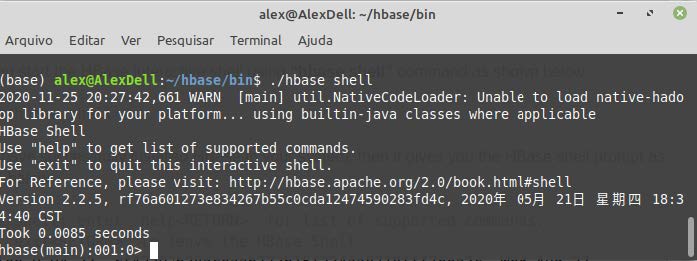
Figura 9 – Interface gráfica do HBase

Fonte: ./hbase shell.

**TEMA 3 – OPERAÇÕES CRUD NO HBASE**

O HBase disponibiliza uma interface de linha de comando (CLI) para criação e gerenciamento dos bancos de dados, que pode ser acessada através da seguinte linha de comando ./hbase shell. A Figura 10 apresenta a interface de linha de comando do HBase.

Figura 10 – Interface de linha de comando do HBase.

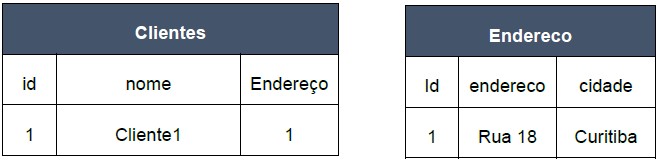


Assim como ocorre com os outros bancos de dados NoSQL, o HBase não suporta operações SQL, como também não suporta:

* Operações entre tabelas;
* Operações entre linhas;
* Agrupamento / Agregação;
* *Joins*;
* Chaves primárias;
* Chaves estrangeiras;
* Restrições.

Todos os dados no HBase precisam descrever uma entidade que deve ser autocontida em sua própria linha. Para melhor compreender essa afirmação, vamos analisar um pequeno exemplo entre um banco de dados relacional com duas tabelas (clientes e endereço) e a mesma estrutura no HBase. A Figura 11 apresenta o modelo de dados relacional.

Figura 11 – Exemplo de um banco de dados relacional



Conforme apresentado, poderíamos facilmente com uma consulta SQL fazendo *join*entre as duas tabelas, descobrir o endereço do cliente de nome Cliente1 ou, por exemplo, quais são todos os clientes que moram no mesmo endereço. Essa mesma estrutura em um banco de dados orientado a colunas, como o HBase, seria representada conforme mostrado na Figura 12.

Figura 12 – Representação do modelo de dados orientado a colunas



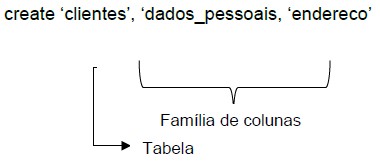
Nesse contexto, conforme os demais bancos de dados NoSQL já estudados, o HBase somente aceita operações CRUD:

* C – Create;
* R – Read;
* U – Update;
* D – Delete.

**3.1 CRIANDO UM BANCO DE DADOS NO HBASE**

Após acessar a interface de linha de comando do HBase, para criar um banco de dados o primeiro passo consiste em elaborar uma tabela e a família de colunas desejada. Para exemplificar essa tarefa usaremos o mesmo exemplo do registro de clientes apresentado nas Figuras 11 e 12 quando da comparação entre um banco de dados relacional e orientado a colunas.

* Criar a tabela Clientes e as famílias de colunas “dados\_pessoais” e “endereco”.

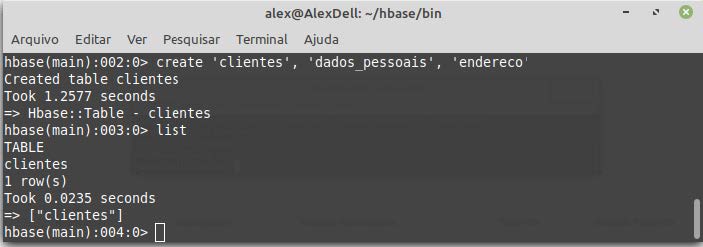


Cada família de colunas pode ter quantos atributos forem necessários. Em nosso exemplo cada família de colunas ficará da seguinte forma:

* dados\_pessoais – nome, idade;
* endereco – rua, cidade.

Ao definir as famílias de colunas de uma tabela no HBase, não é necessário especificar o tipo de dados, pois isso é feito de acordo com o tipo de dados adicionado para cada registro, de modo que uma coluna pode ter diferentes tipos de dados. A Figura 13 apresenta a interface CLI do HBase para a criação da tabela clientes, com o comando list para visualizar as tabelas criadas.

Figura 13 – Criação da tabela clientes no HBase.



* Inserir dados na tabela clientes

put ‘clientes’, ‘1’, ‘dados\_pessoais:nome’, ‘Cliente1’

put ‘clientes’, ‘1’, ‘dados\_pessoais:idade’, ‘35’

put ‘clientes’, ‘1’, ‘endereco:rua’, ‘Rua 18’

put ‘clientes’, ‘1’, ‘dados\_pessoais:cidade’, ‘Curitiba’

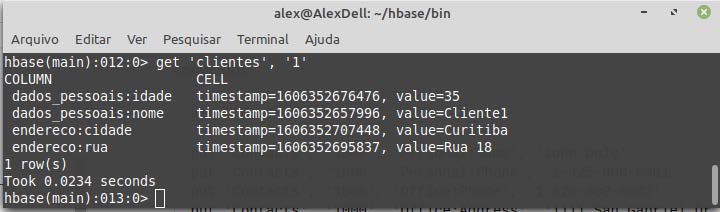
A Figura 14 mostra um exemplo de inserção de dados na tabela clientes.

Figura 14 – Inserção de dados na tabela clientes.



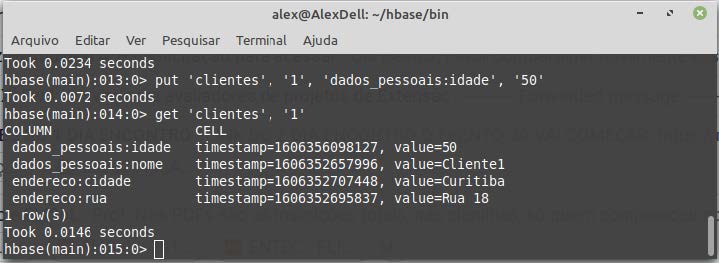
Para visualizar os dados inseridos em uma tabela, usamos o seguinte comando **scan ‘nome\_da\_tabela’**. Para localizar os registros em uma tabela, usamos o comando get da seguinte maneira: **get ‘nome\_da\_tabela’, ‘id\_do\_registro`.**A Figura 15 apresenta um exemplo de uma consulta aos registros da tabela clientes.

Figura 15 – Exemplo de uma consulta na tabela clientes



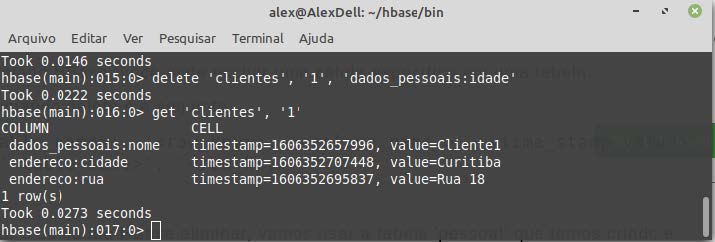
Para atualizar os dados de um registro em uma tabela, usamos o comando put de modo similar para inserir um novo registro, porém, especificamos o id do registro a ser alterado e a coluna a ser atualizada. A Figura 16 apresenta um exemplo de atualização da idade do Cliente1 de 35 para 50 anos.

Figura 16 – Exemplo de atualização de registros



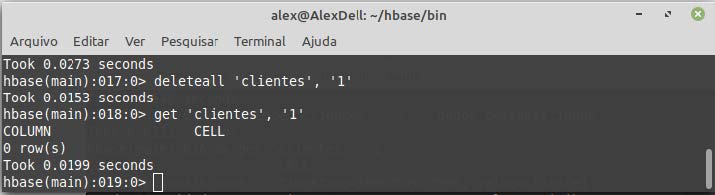
Para excluir os dados de um registro em uma tabela, podemos excluir uma célula de um registro, por exemplo, excluir a idade do Cliente1 ou, excluir um registro completo. Para excluir uma célula de um registro usamos o comando delete ‘clientes’, ‘1’, ‘dados\_pessoais:idade’. A Figura 17 mostra um exemplo de remoção de uma célula de um registro.

Figura 17 – Exemplo de remoção da idade



Para excluir um registro completo usamos o comando deleteall ‘clientes’, ‘1’. A Figura 18 mostra um exemplo de remoção de um registro completo.

Figura 18 – Exemplo de remoção de um registro inteiro



**TEMA 4 – CARACTERÍSTICAS DE CONSISTÊNCIA, DISPONIBILIDADE E ESCALABILIDADE**

Conforme destaca Shon (2014), o HBase é uma solução de banco de dados NoSQL altamente distribuída que pode ser dimensionada para armazenar grandes quantidades de dados esparsos. Para Filipa (2020), o HBase escala linearmente, exigindo que todas as tabelas tenham uma chave. O espaço da chave está dividido em blocos sequenciais que são então atribuídos a uma região. Os servidores de região possuem uma ou mais regiões, de modo que a carga está distribuída uniformemente em todo o *cluster*. Ainda conforme Filipa (2020), os clientes sabem exatamente onde está qualquer informação no HBase e podem entrar em contato diretamente com o servidor de região sem necessidade de um coordenador central.Estas características exigem do HBase diversas garantias, como consistência de dados, disponibilidade e escalabilidade, conforme veremos a seguir.

**4.1 CARACTERÍSTICAS DE CONSISTÊNCIA**

O HBase, de acordo com Claudera (2012), sempre apresentou forte garantia de consistência. Todas as leituras e gravações no banco de dados são roteadas por meio de um único servidor de região, o que garante que todas as gravações no banco ocorram em ordem e todas as leituras acessem os dados confirmados mais recentemente. Ainda conforme o autor, devido a esse roteamento de leituras em um único local, se o servidor ficar indisponível, as regiões das tabelas hospedadas no servidor de região ficarão indisponíveis por algum tempo até que sejam recuperadas. Para Claudera (2012), o HBase garante a consistência na linha do tempo dos dados e de forma forte:

* **Consistência na linha do tempo**: garante a consistência da linha do tempo para todos os dados servidos por servidores de região no modo secundário, o que significa que todos os clientes do HBase visualizam os mesmos dados na mesma ordem, mas esses dados podem estar um pouco desatualizados. Apenas o servidor de região primário tem a garantia de ter os dados mais recentes.
* **Consistência forte**: a consistência forte significa que os dados mais recentes são sempre veiculados. No entanto, a consistência forte dos dados pode aumentar muito a latência no caso de uma falha do servidor de região, porque apenas o servidor de região primário tem garantia de ter os dados mais recentes.

**4.2 CARACTERÍSTICAS DE DISPONIBILIDADE**

Conforme abordado por Filipa (2020), o HBase garante a disponibilidade dos dados de vários modos, tais como:

* Informações de topologia de cluster altamente disponíveis através de implantações de produção com múltiplas instâncias HMaster e ZooKeeper;
* Distribuição de dados em vários nós significa que a perda de um único nó afeta somente os dados armazenados nesse nó;
* HBase permite o armazenamento de dados garantindo que a perda de um único nó não resulte na perda de disponibilidade de dados;
* O formato HFile armazena dados diretamente no HDFS. O HFile pode ser lido ou escrito por diversas tecnologias Apache, permitindo análises profundas no HBase sem movimento de dados.

Para alcançar alta disponibilidade para leituras, conforme destacado em Claudera (2012), o HBase fornece um recurso chamado *replicação de regiã*o. Nesse modelo, para cada região de uma tabela pode haver várias réplicas que são abertas em servidores de região diferentes. Por padrão, a replicação da região é definida como 1, portanto apenas uma única réplica da região é implantada e não há mudanças no modelo original. Se a replicação de região for definida como 2 ou mais, o mestre atribuirá réplicas das regiões da tabela.

**4.3 CARACTERÍSTICAS DE ESCALABILIDADE**

O HBase escala linearmente quando lida com grandes conjuntos de dados formados por bilhões de linhas e milhões de colunas e combina facilmente fontes de dados que utilizam grande variedade de estruturas e esquemas diferentes. O escalonamento possui uma recomendação mínima de 5 nós por cluster Hadoop e permite escalonar com facilidade para centenas de nós de acordo com a demanda.

**TEMA 5 – CASOS DE USO APROPRIADOS**

Em sua abordagem, Shon (2014) destaca duas importantes áreas de aplicação dos bancos de dados orientados a colunas:

* Análise em lote de dados de logs, devido a sua otimização para leituras e varreduras sequenciais;
* Captura de métricas em tempo real de aplicativos, servidores, preferências do usuário, entre outros.

Nesse contexto, Shon (2014) ainda destaca três importantes empresas que fazem uso do HBase:

* Facebook – gerenciamento de mensagens de usuários.
* Pinterest – fornecer *feeds* personalizados aos usuários, capturar dados e potencializar seu processo de recomendações.
* Explorys – capturar bilhões de pontos anônimos de dados clínicos, operacionais e financeiros. A Explorys usa essa plataforma para ajudar seus clientes a obter atendimento de qualidade, minimizar custos e mitigar riscos.

Seguindo essa abordagem, Filipa (2020) cita outros exemplos de aplicação dos bancos de dados orientados a colunas, focando principalmente no uso do HBase. As empresas usam o armazenamento de baixa latência do HBase para cenários que exigem análise em tempo real e dados tabulares para aplicativos de usuários finais. Uma empresa que fornece serviços de segurança na *web* mantém um sistema que aceita bilhões de traços de eventos e registros de atividades dos *desktops* dos seus clientes todos os dias.

**FINALIZANDO**

Nesta aula abordamos de modo geral os conceitos e aplicações de uso sobre os bancos de dados NoSQL orientados a colunas e suas principais características. Conhecemos por meio de várias comparações dessa estrutura de dados com os bancos de dados relacionais, as características de armazenamento e manipulação de dados, dada a similaridade da representação das famílias de colunas em tabelas.

A partir do Tema 2, observamos em profundidade o HBase, um dos principais sistemas gerenciadores de bancos de dados NoSQL orientados a colunas. Abordamos também a instalação e configuração do SGBD, bem como cada uma das operações CRUD para manipulação dos dados no Tema 3. Também compreendemos a partir do Tema 4 algumas características específicas de consistência, disponibilidade e escalabilidade, visto que normalmente essa estrutura de dados é muito utilizada em sistemas distribuídos, principalmente como HBase que focamos nesta aula.

Vimos também que os bancos de dados NoSQL orientados a colunas, assim como os bancos orientados a chave-valor, também são muito úteis para sistemas *on-line* que demandam excessivo número de acessos de leitura, principalmente em sistemas distribuídos, pois oferecem recursos mais rápidos e estáveis do que os bancos de dados relacionais.

**REFERÊNCIAS**

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

BARROSO, I. **Banco de Dados Orientado a Colunas**. Isaías Barroso, 2012. Disponível em: <https://isaiasbarroso.wordpress.com/2012/06/20/banco-de-dados-orientado-a-colunas/>. Acesso em: 29 abr. 2021.

GEORGE, L. **HBase**: The Definitive Guide. Sebastopol: O’Reilly, 2011.

SHON, P. Apache HBase Explained in 5 Minutes or Less. **Credera**, 2014. Disponível em: <https://www.credera.com/insights/apache-hbase-explained-5-minutes-less/>. Acesso em: 29 abr. 2021.

FILIPA, S. Apache HBase: O que é, Conceitos e Definições. **Cetax**, 2020. Disponível em: <https://www.cetax.com.br/blog/o-que-e-o-apache-hbase/>. Acesso em: 29 abr. 2021.

Introduction to HBase High Availability, **Claudera**, c2012-2020. Disponível em: <https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.6.0/bk\_hadoop-high-availability/content/ha-hbase-intro.html#:~:text=HBase%2C%20architecturally%2C%20has%20had%20a,the%20most%20recently%20committed%20data>. Acesso em: 29 abr. 2021.