



BANCO DE DADOS NOSQL

AULA 4

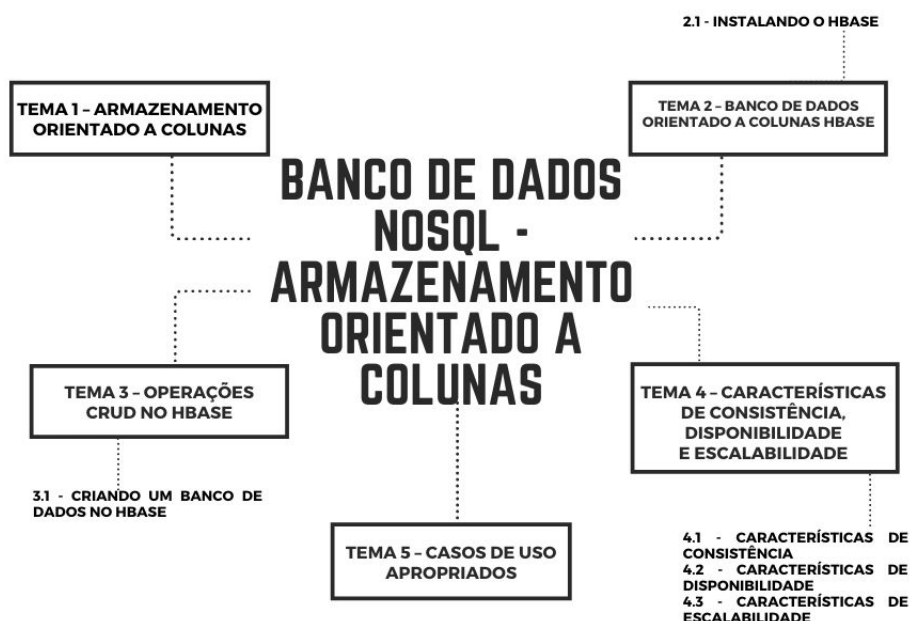


Prof. Alex Mateus Porn

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.

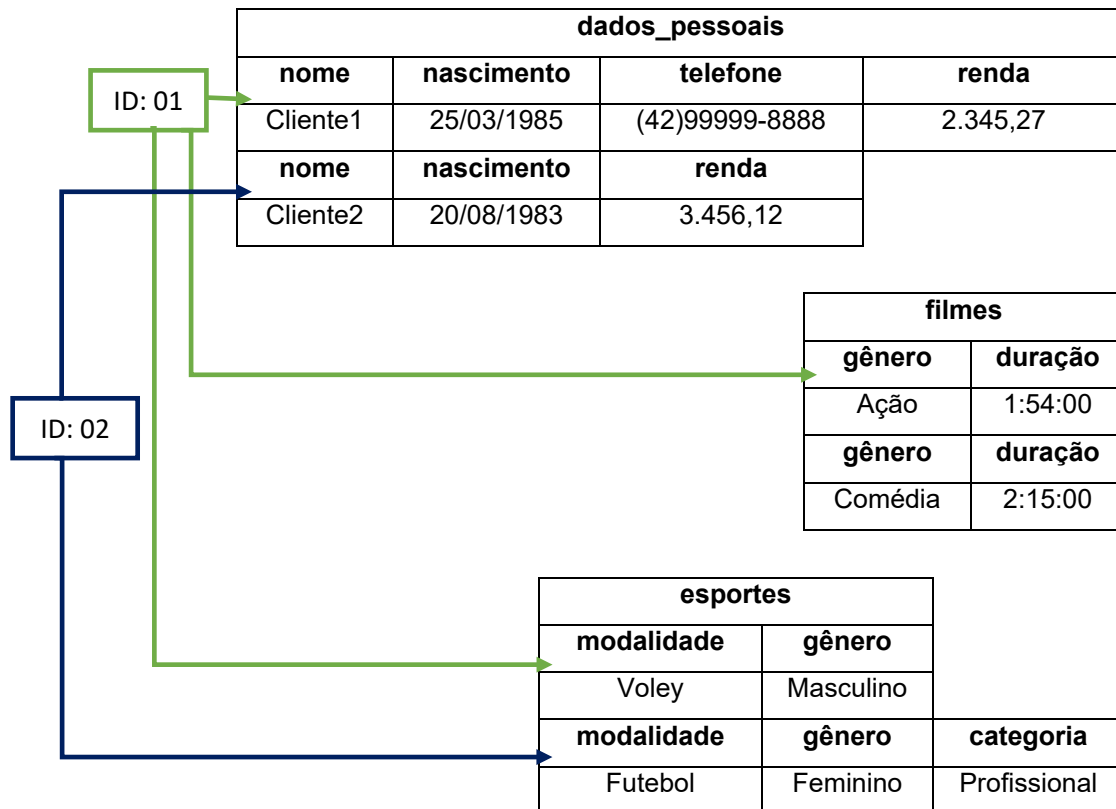
Para Marquesone (2017, p. 51), no cenário para armazenamento das preferências do usuário em um aplicativo de compras online, utilizando um banco de dados orientado a colunas, poderia ser definido pelo menos três famílias de colunas:

- 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>.

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>> 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

The screenshot shows the Apache HBase Downloads page. The table lists releases with the following data:

Version	Release Date	Compatibility Report	Changes	Release Notes	Download	Notices
2.3.3	2020/11/02	2.3.2 vs 2.3.3	Changes	Release Notes	src (sha512 asc) bin (sha512 asc) client-bin (sha512 asc)	
2.2.3	2020/09/04	2.2.6 vs 2.2.5	Changes	Release Notes	src (sha512 asc) bin (sha512 asc) client-bin (sha512 asc)	Stable release
1.6.0	2020/03/06	1.5.0 vs 1.6.0	Changes	Release Notes	src (sha512 asc) bin (sha512 asc)	

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:

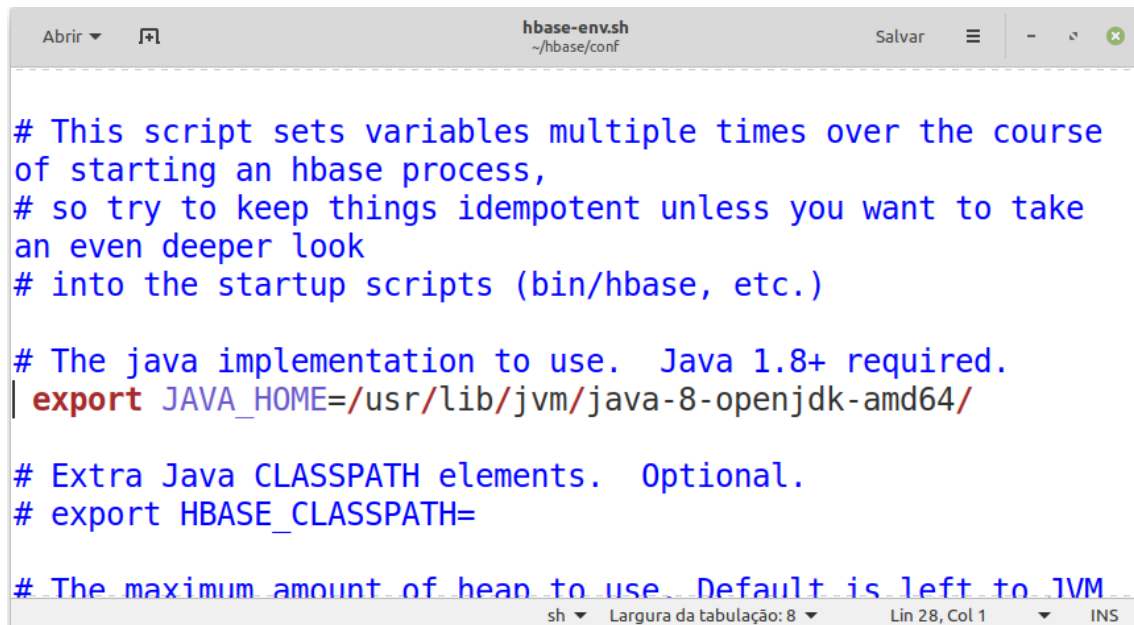
1. */home/usuário/hbase/conf/hbase-env.sh*
2. */home/usuário/hbase/conf/hbase-site.xml*

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>> 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



```
# This script sets variables multiple times over the course
of starting an hbase process,
# so try to keep things idempotent unless you want to take
an even deeper look
# into the startup scripts (bin/hbase, etc.)

# The java implementation to use.  Java 1.8+ required.
| export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/

# Extra Java CLASSPATH elements.  Optional.
# export HBASE_CLASSPATH=

# The maximum amount of heap to use. Default is left to JVM
```

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



```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:/home/usuario/hbase</value>
  </property>
</configuration>
```

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

```
alex@AlexDell: ~/hbase/bin
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
(base) alex@AlexDell:~$ cd /home/alex/hbase/bin/
(base) alex@AlexDell:~/hbase/bin$
```

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

```
alex@AlexDell: ~/hbase/bin
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
(base) alex@AlexDell:~/hbase/bin$ ./start-hbase.sh
running master, logging to /home/alex/hbase/bin/../logs/hbase-alex-master-AlexDell.out
(base) alex@AlexDell:~/hbase/bin$
```

Para conferir se o HBase iniciou corretamente e o servidor máster está ativo, podemos usar o comando `jps` no terminal, conforme apresentado na



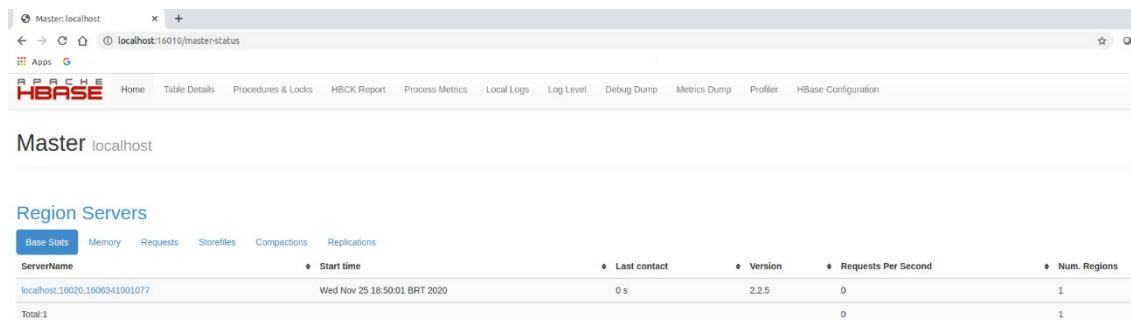
Figura 8; deverá ser apresentado o processo HMaster, que corresponde a nossa instância local do HBase.

```
alex@AlexDell: ~/hbase/bin
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
(base) alex@AlexDell:~/hbase/bin$ ./start-hbase.sh
running master, logging to /home/alex/hbase/bin/../logs/hbase-alex-master-AlexDell.out
(base) alex@AlexDell:~/hbase/bin$ jps
31700 HMaster
873 Jps
(base) alex@AlexDell:~/hbase/bin$
```

Figura 8 – Visualização das instâncias do HBase.

Outro modo de conferir se o HBase está executando perfeitamente e também visualizar os bancos de dados criados, é acessar a interface gráfica pelo navegador web através do seguinte endereço <<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.

```
alex@AlexDell: ~/hbase/bin
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda

(base) alex@AlexDell:~/hbase/bin$ ./hbase shell
2020-11-25 20:27:42,661 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop
op library for your platform... using builtin-java classes where applicable
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.2.5, rf76a601273e834267b55c0cda12474590283fd4c, 2020年 05月 21日 星期四 18:3
4:40 CST
Took 0.0085 seconds
hbase(main):001:0>
```

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

Clientes			Endereco		
id	nome	Endereço	Id	endereco	cidade
1	Cliente1	1	1	Rua 18	Curitiba

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

id	nome	endereço
1	Cliente1	<STRUCT>

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

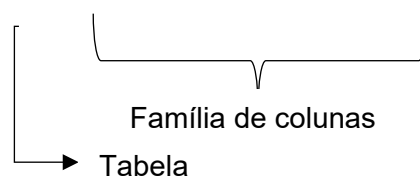
- C – Create;
- R – Read;
- U – Update;
- D – Delete.

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 “endereço”.

```
create 'clientes', 'dados_pessoais', 'endereço'
```



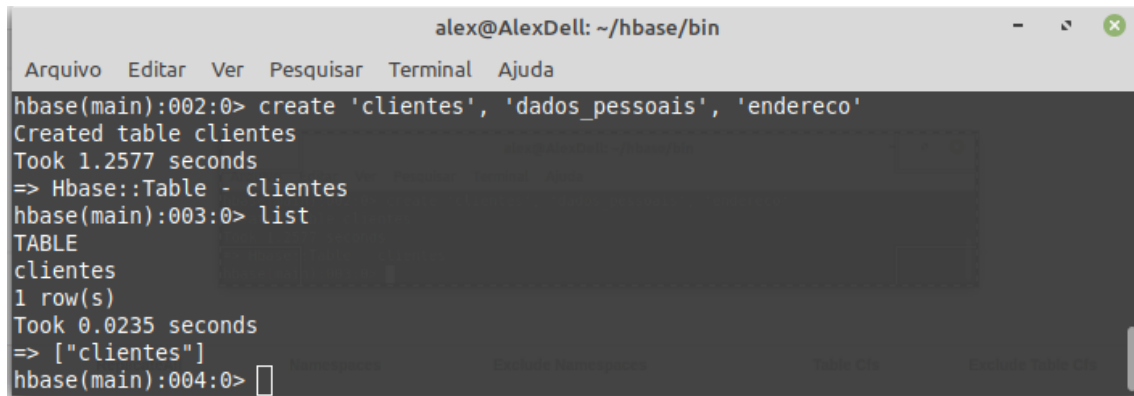
Cada família de colunas pode ter quantos atributos forem necessários. Em nosso exemplo cada família de colunas ficará da seguinte forma:

- dados_pessoais – nome, idade;
- endereço – 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.



```
alex@AlexDell: ~/hbase/bin
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
hbase(main):002:0> create 'clientes', 'dados_pessoais', 'endereco'
Created table clientes
Took 1.2577 seconds
=> Hbase::Table - clientes
hbase(main):003:0> list
TABLE
clientes
1 row(s)
Took 0.0235 seconds
=> ["clientes"]
hbase(main):004:0> 
```

- 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.



```
alex@AlexDell: ~/hbase/bin
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Took 0.0235 seconds
=> ["clientes"]
hbase(main):004:0> put 'clientes', '1', 'dados_pessoais:nome', 'Cliente1'
Took 0.2210 seconds
hbase(main):005:0> put 'clientes', '1', 'dados_pessoais:idade', '35'
Took 0.0037 seconds
hbase(main):006:0> put 'clientes', '1', 'endereco:rua', 'Rua 18'
Took 0.0062 seconds
hbase(main):007:0> put 'clientes', '1', 'endereco:cidade', 'Curitiba'
Took 0.0035 seconds
hbase(main):008:0> 
```

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

```
alex@AlexDell: ~/hbase/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
hbase(main):012:0> get 'clientes', '1'
COLUMN CELL
dados_pessoais:idade timestamp=1606352676476, value=35
dados_pessoais:nome timestamp=1606352657996, value=Cliente1
endereco:cidade timestamp=1606352707448, value=Curitiba
endereco:rua timestamp=1606352695837, value=Rua 18
1 row(s)
Took 0.0234 seconds
hbase(main):013:0> 
```

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

```
alex@AlexDell: ~/hbase/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
Took 0.0234 seconds
hbase(main):013:0> put 'clientes', '1', 'dados_pessoais:idade', '50'
Took 0.0072 seconds
hbase(main):014:0> get 'clientes', '1'
COLUMN CELL
dados_pessoais:idade timestamp=1606356098127, value=50
dados_pessoais:nome timestamp=1606352657996, value=Cliente1
endereco:cidade timestamp=1606352707448, value=Curitiba
endereco:rua timestamp=1606352695837, value=Rua 18
1 row(s)
Took 0.0146 seconds
hbase(main):015:0> 
```

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.

```
alex@AlexDell: ~/hbase/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
Took 0.0146 seconds
hbase(main):015:0> delete 'clientes', '1', 'dados_pessoais:idade'
Took 0.0222 seconds
hbase(main):016:0> get 'clientes', '1'
COLUMN CELL
dados_pessoais:nome timestamp=1606352657996, value=Cliente1
endereco:cidade timestamp=1606352707448, value=Curitiba
endereco:rua timestamp=1606352695837, value=Rua 18
1 row(s)
Took 0.0273 seconds
hbase(main):017:0> 
```

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

```
alex@AlexDell: ~/hbase/bin
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Took 0.0273 seconds
hbase(main):017:0> deleteall 'clientes', '1'
Took 0.0153 seconds
hbase(main):018:0> get 'clientes', '1'
COLUMN          CELL
0 row(s)
Took 0.0199 seconds
hbase(main):019:0> 
```

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 Cludera (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 Cludera (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 Cloudera (2012), o HBase fornece um recurso chamado *replicação de região*. 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**. Isaias 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, **Cloudera**, 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.