



# Apache HBase

**Beca - Data&Analytics 2021**  
**Technology - Valdir Sevaio**

## AGENDA



- 01 ► Introdução NoSQL e por quê utilizamos no Bigdata?
- 02 ► Introdução Apache HBase
- 03 ► Comandos de definição de dados
- 04 ► Comandos de manipulação de dados
- 05 ► Comandos de segurança de dados
- 06 ► Cenários de Utilização do HBase
- 07 ► Operação Bulk Insert
- 08 ► Integração Hive e HBase
- 09 ► Próximos passos



# Introdução NoSQL

Beca Data&Analytics 2021

## Significados:

- NoSQL = “No SQL”, “Não SQL” ou “Não Relacional”
- NoSQL = “Not Only SQL”
- Termo genérico para representar os bancos de dados não relacionais.
- O NoSQL emergiu como uma alternativa de banco de dados não relacionado, normalmente evitando operações de “*join*”, é distribuído, *open-source*, escalável na horizontal, livre de modelagens ou schema (não é necessário fixar modelos para as tabelas), suporta replicação, acesso via API de operações e eventualmente consistente.

## Apache HBase

## Banco de dados Relacional vs Bancos de dados NoSQL – Parte 1

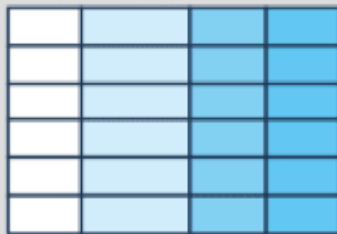
Quando considerar NoSQL	Quando considerar Relacional
Carga de trabalho de alto volume que exigem grande escala	Carga de trabalho é consistente e requer escala média para grande
Carga de trabalho não exigem garantias do ACID	Garantias de ACID são necessárias
Os dados são dinâmicos e frequentemente alterados	Dados são previsíveis e altamente estruturados
Os dados podem ser expressos sem relações (joins)	Os dados são expressos de maneira relacional
Alto velocidade de gravação e a segurança de gravação não é crítica	A garantia de gravação é um requisito
Consulta de dados é simples e tende a ser simples	Consultas e relatórios complexos
Dados exigem uma ampla distribuição geográfica	Usuários são mais centralizados

Apache  
HBase

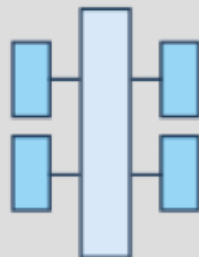
## Banco de dados Relacional vs Bancos de dados NoSQL – Parte 2

### SQL

#### Relational

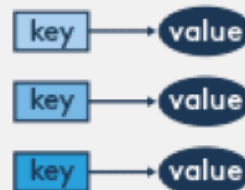


#### Analytical (OLAP)



### NoSQL

#### Key-Value



#### Column-Family



#### Graph

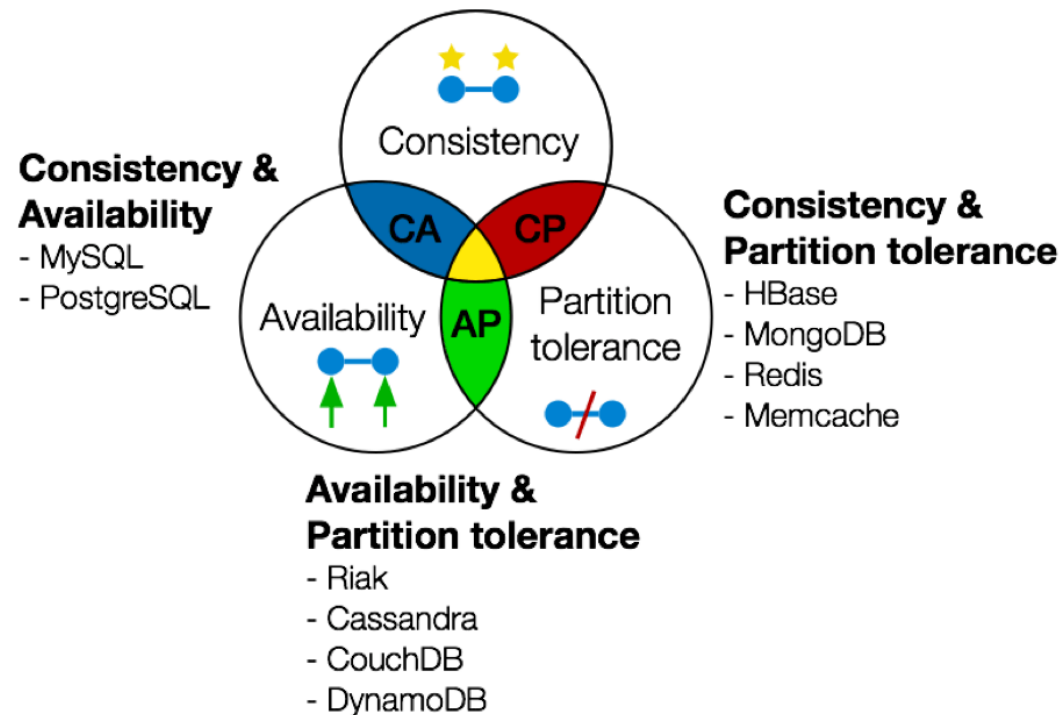


#### Document



O teorema CAP ou *teorema de Brewer* indica que o armazenamento de dados distribuídos só podem atender **dois dos três atributos**: Consistência, Disponibilidade, Partição Tolerante a Falhas.

**Tanto Hadoop e HBase atendem CP**, porque possuem um ponto de falha que é respectivamente o NamedNode e HMaster que não possuem redundância dos dados dos próprios serviços para todos os nós do cluster.



## Limitações do Hadoop

- Hadoop pode executar apenas processamento batch e os dados são acessados de forma sequencial, isso significa que é necessário percorrer todo o conjunto de arquivos (*scan search*) mesmo para os jobs mais simples.

## Cenário

- Um grande conjunto de dados (*dataset*) quando processado com um outro conjunto de dados, ambos serão processados de maneira sequencial, nesse momento **uma nova solução é necessária para acessar qualquer ponto do *dataset* e levando um tempo menor**
- Aplicações como HBase, Cassandra, Dynamo e MongoDB, etc, são banco dados que armazenam grandes quantidade de dados e os acessos à esses dados são realizados de forma **aleatória em termos de posição do registro e do tempo.**





# Introdução Apache HBase

Beca Data&Analytics 2021

**HBase** é um banco de dados ***distribuído*** e **orientado a colunas** (*Column Family ou Wide Column*).

Uma definição mais técnica:

O armazenamento do HBase é um esparso, distribuído, persistente, multidimensional e ordenado ***Map***.

**Map** é indexado por uma linha chave (**row key**), coluna chave (**column key**), e um coluna **timestamp**.

Cada valor no **Map** é interpretado como um vetor de bytes (**array of bytes**).

O **array of bytes** nos permite gravar portanto qualquer informação se for necessário, inclusive documentos, arquivos JSON, CSV, etc.

## O que é Apache Hbase?

O HBase usa um modelo de dados aonde os usuários armazenam linhas em tabelas nomeadas.

Um linha da tabela tem uma chave ordenável (**row key**) e um conjunto de columns.

A tabela é armazenada de maneira esparsa que permite a definição de uma grande quantidade de colunas se for necessário.

Portanto podemos entender que o núcleo de dados do HBase é um **Map**.

Na maioria das linguagens de programação essa abstração de estrutura de dados existe e pode ser representada como um conjunto de chaves e conjunto de valores.

**Cada chave é associada à um valor.**

```
{  
  "zzzzzz" : "Olá",  
  "xyz" : "hello",  
  "aaaab" : "world Hbase!",  
  "1" : "x",  
  "aaaaa" : "y"  
}
```

## Persistente

Significa que o dado que é colocado nesse **map**, é persistido depois que o programa que criou ou acessou finalizou.

## Distribuído

HBase foi construído em cima dos sistemas de arquivos distribuídos HDFS (*Hadoop's Distributed File System*) ou Amazon S3.

Portanto os dados são replicados para um conjunto de nós e há essa camada de proteção quando um nó entra em falha.

## Esparso

Uma linha pode conter qualquer quantidade de colunas para cada *Column Family* ou nenhum coluna preenchida.

## Ordenado

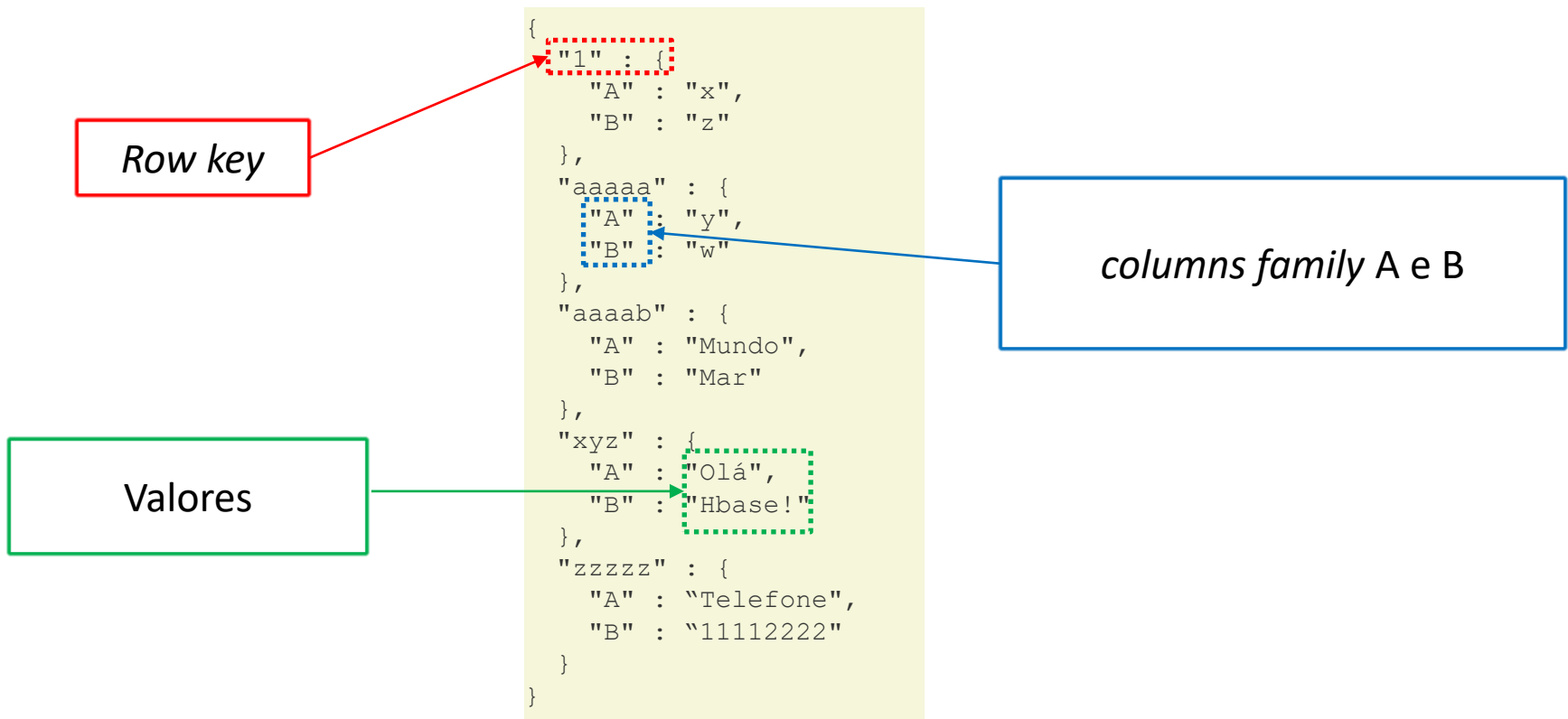
Diferente da maioria das implementações da abstração da estrutura **Map**, as chaves de cada linha são rigidamente **ordenadas por ordem alfabética**.

```
{  
  "1" : "x",  
  "aaaaa" : "y",  
  "aaaab" : "world Hbase!",  
  "zzzzz" : "Olá",  
  "xyz" : "hello"  
}
```

**IMPORTANTE:** Só as chaves são indexadas automaticamente e a convenção de nomes para chaves determinará a proximidades dos registros

## Multidimensional e Ordenado

Esquecendo o conceito tradicional de linhas e colunas do mundo relacional, pense no multidimensional como um **Map** que contém **Maps**.





## ***Column Family (CF)***

O mapeamento de colunas se dá na criação da *column family* que é a definição de conjuntos de colunas que representam determinado domínio da informação.

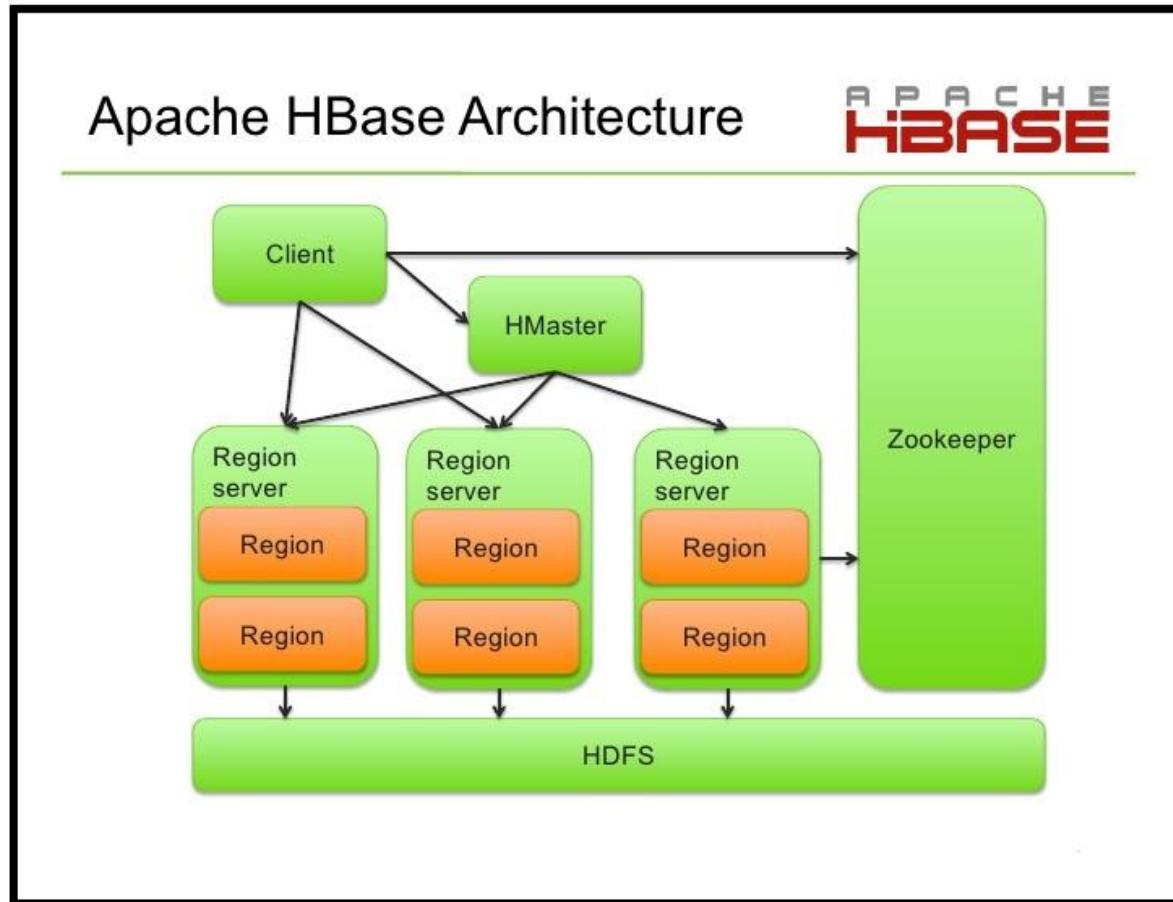
Em versões antigas do Hbase, a definição da *Column Family* é realizada normalmente na **criação da tabela** e de difícil alteração posterior. Pode se tornar caro adicionar novas *column family*, portanto é uma boa pensar em todas as possíveis que podem ser geradas no início da criação.

Um ponto importante, embora a definição da *column family* é estática, as colunas não são portanto se necessário pode ser adicionado novas a qualquer momento.

Como cada linha pode conter quantidades diferentes de coluna, não é possível consultar todas as colunas de todas as linhas sem fazer um *full-scan* em todo o conteúdo da tabela.

## Apache HBase

### Arquitetura alto nível de componentes do Apache HBase



## HBase Client

Responsável por encontrar o *RegionServers* que estão atendendo as linhas em particular que estão sendo utilizadas, para isso é consultado uma base de dados de metadados interna do Hbase, **hbase:meta**.

Depois de localizado o *RegionServer*, o *client* se comunica para solicitar requisição de leitura/gravação do registro, isso até versão 2.x.y.

Apartir da versão 3.x.y. o HMaster controla as requisições leitura/gravação.

Caso ocorra erro na comunicação com o *RegionServer* por alguma falha será atribuído pelo **load balancer** do HMaster um novo *RegionServer*.

## Apache HBase

## Arquitetura alto nível de componentes do Apache HBase

### HMaster

Responsável por monitorar todas as instâncias de *RegionServer* no cluster.  
É meio para todas as solicitações de mudanças de metadados.

Em um ambiente distribuído de produção esse serviço é executado no NamedNode do Hadoop.

É possível ter vários nós de um ambiente clusterizado atuar como *master*, porém só um pode ficar ativo, o restante fica passivo, caso ocorra alguma falha no master principal.

## Apache HBase

### Arquitetura alto nível de componentes do Apache HBase

#### **RegionServer**

Responsável por servir e gerenciar os nós que atuam como regiões. No ambiente distribuído produtivo esse serviço é executado no DataNode (nós) de um ambiente Hadoop.

Lida com as operações nos dados (get, put, delete, next, etc)

Operações da Region (splitRegion, compactRegion, etc)

Cada região tem responsabilidade sobre operações de controle como: controle de memória, compactação de dados, atribuição cache automático.

## Apache HBase

## Arquitetura alto nível de componentes do Apache HBase

### Regions

Representam os elementos básicos de disponibilidade e distribuição das tabelas e incluem o armazenamento de cada *column family*.

### Hierarquia dos Objetos no HBase

- Table (Tabela HBase)

  - Region (Regiões da tabela)

    - Store (Unidade de armazenamento por ColumnFamily para cada região da tabela)

      - MemStore (MemStore para cada armazenamento em cada região da tabela)

      - StoreFile (StoreFiles para cada armazenamento em cada região da tabela)

        - Block (Arquivos que servem como blocos dentro do StoreFile dentro do armazenamento em cada região de uma tabela)

## Apache HBase

## Arquitetura alto nível de componentes do Apache HBase

### Apache Zookeeper

O tipo de instalação distribuída do Hbase é necessário que o Apache Zookeeper esteja funcionando no cluster.

O Apache Zookeeper é o responsável por dar visibilidade a todos os nós de serviços do HBase de quem é o master atual, são os servidores que atendem ao papel RegionServers, Region. Toda configuração que é padrão para cada um dos papéis são armazenados no Zookeeper, portanto ele tem um papel fundamental de manter sincronizado toda a parametrização em comum para todos os nós.

O cliente HBase se comunica com os RegionServer através do Zookeeper.

## Apache HBase

Arquitetura alto nível de componentes do  
Apache HBase

### Apache HDFS

É o sistema de arquivos do ecossistema do Hadoop. Cada arquivo está armazenado em múltiplos blocos e mantém tolerância a falhas, os blocos são replicados pelos cluster Hadoop.

HDFS é utilizado pelo componentes do HBase.

Os arquivos gerenciados pelo HBase são criados dentro do HDFS.

#### HBASE

#### HDFS com MR

Baixa latência nas operações	Alta latência nas operações
Acesso a leitura e gravação aleatória	Grava única e leitura muita vezes
Acessado através de comandos shell, cliente API em Java, REST, Avro ou Thrift	Acessado primeiramente através dos Jobs do MR(Map Reduce)



A background image showing two people, a man and a woman, smiling and looking at each other in a library setting with bookshelves in the background. The image has a purple tint.

# Comandos Gerais no HBase

Beca Data&Analytics 2021

Apache  
HBase

## Comandos no HBase

**hbase shell**

No console do HBase é possível utilizar todos os comandos de manipulação de informação e comandos gerais.

Atualmente o HBase até a versão 3.x.y não suporta uma query de consulta (SQL).

Os comandos mais básicos de manipulação das informação são: put, get, scan, drop, disable, etc.

```
[hadoop@dataserver conf]$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hbase/lib/client-facing-thirdparty/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
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 2.2.0, rUnknown, Tue Jun 11 04:30:30 UTC 2019
Took 0.0055 seconds
hbase(main):001:0> █
```

## Apache HBase

## Comandos no HBase

### help

Exibe informações adicionais de uso para cada comando existente.

#### Sintaxe:

```
help '<comando>'
```

#### Comando:

```
help 'create'
```

```
help 'describe'
```

```
help 'get'
```

```
help 'put'
```

## status

Comando dá detalhes sobre o sistema como o número de servidores presente, quantidade de servidores ativos, média de carga, quantidade de Stored ativos, é possível passar qualquer parâmetro dependendo em qual detalhe seja necessário saber sobre o sistema.

```
hbase(main):001:0> status  
1 active master, 0 backup masters, 1 servers, 0 dead, 3.0000 average load  
Took 0.5941 seconds
```

## Variações

**status 'simple'**

**status 'summary'**

**status 'detailed'**

Apache  
HBase

## Comandos no HBase

**version**

Exibe a versão do HBase.

```
hbase(main):008:0> version
2.2.0, rUnknown, Tue Jun 11 04:30:30 UTC 2019
Took 0.0002 seconds_
```

**table\_help**

Comando que auxilia como utilizar comandos que se referenciam a uma tabela.

É possível pegar a referencia de uma tabela cria e utilizar como variável para próximas operações.

```
hbase(main):009:0> table help
Help for table-reference commands.
```

You can either create a table via 'create' and then manipulate the table via commands like 'put', 'get', etc. See the standard help information for how to use each of these commands.

However, as of 0.96, you can also get a reference to a table, on which you can invoke commands. For instance, you can get create a table and keep around a reference to it via:

```
hbase> t = create 't', 'cf'
```

Or, if you have already created the table, you can get a reference to it:

```
hbase> t = get_table 't'
```

You can do things like call 'put' on the table:

```
hbase> t.put 'r', 'cf:q', 'v'
```

## Apache HBase

## Comandos no HBase

### whoami

Exibe o usuário logado e a qual grupo de permissão pertence.

```
hbase(main):012:0> whoami
hadoop (auth:SIMPLE)
  groups: hadoop
Took 0.0125 seconds_
```



# Comandos Definição de dados

Beca Data&Analytics 2021

Os comandos abaixo são utilizados para operar **as tabelas** no HBase.

**create** – Cria uma tabela.

**list** – Lista todas as tabelas no HBase independente do *namespace*.

**disable** – Desabilita uma tabela.

**is\_disabled** – Checa se uma tabela está desabilitada.

**enable** – Habilita uma tabela.

**is\_enabled** – Checa se uma tabela está habilitada.

**describe** – Exibe informações de definição de uma tabela.

**alter** – Realiza alterações em uma tabela.

**exists** – Verifica se uma tabela existe.

**drop** – Exclui um tabela do HBase.

**drop\_all** – Exclue todas as que se aplicam a um padrão de nomes via regra de Regex.

**IMPORTANTE:** Todas as operações lista acima estão disponíveis via API Java de fácil utilização sob o pacote `org.apache.hadoop.hbase.client` com os objetos `HBaseAdmin` e `HTableDescriptor`.



Apache  
HBaseComandos de Definições no Hbase – CREATE  
TABLE**CREATE**

Permite criar uma tabela no HBase.

Sintaxe:

```
create '<nome tabela>', '<nome da column family>' {PROPRIEDADES}
```

Exemplo:

Imagine uma tabela de funcionarios

Row Key	Dados Pessoais	Dados Profissional

## Apache HBase

### Comandos de Definições no Hbase – CREATE TABLE

#### Exemplo criação da tabela de funcionários

Comando:

```
create 'funcionario', 'pessoais', 'profissionais'
```

```
create 'funcionario', {NAME=>'pessoais', VERSIONS=>5}, {NAME=>'profissionais', VERSIONS=>4}
```

Liste todas as tabelas

**list**

Com o comando *list* é possível visualizar todas as tabelas presentes e criados no HBase. É possível passar expressões regulares para realizar buscas mais personalizadas.

## DESCRIBE

O comando exibe informações sobre as *column families* presentes na tabela mencionada, também traz informações sobre os filtros associados, versões, se a tabela está em memória, etc.

### Sintaxe:

describe '<nome da tabela>'

### Comando:

**describe 'funcionario'**

## DISABLE

Desabilita a tabela mencionada, caso seja necessário que uma tabela seja deletada ou excluída, primeiro é necessário desabilita-la.

Após desabilitada ainda é possível lista-la (*list*) em conjunto com as demais tabelas e checar sua existência com comando *exists*, porém não é possível mais escanear (*scan*).

### Sintaxe:

```
disable '<nome da tabela>'
```

### Comando:

```
disable 'funcionario'
```

## **DISABLE\_ALL**

Desabilita as tabelas que atendem dentro do critério de expressão regular.

Para evitar interrupções importantes esse comando tem uma confirmação manual (Sim ou Não) antes de efetivar a desabilitação.

### Sintaxe:

disable '<nome da tabela>'

### Comando:

**disable 'funcionario'**

## ENABLE

Habilita a tabela mencionada no comando. Se a tabela está desabilitada no primeiro momento e não foi deletada ou excluída, e desejamos reutilizar a tabela, então primeiro nos precisamos habilita-la novamente.

### Sintaxe:

enable '<nome da tabela>'

### Comando:

**enable 'funcionario'**

## DROP

Para deletar ou dropar uma tabela presente no HBase, primeiro é necessário desabilitá-la com o comando *disable*.

### Sintaxe:

drop '<nome da tabela>'

### Comando:

drop 'funcionario'

## DROP\_ALL

Esse comando excluirá todas as tabelas que estiverem com o nome dentro da regra da expressão regular.

Todas as tabelas que serão excluídas, precisam estar na situação de desabilitadas, portanto ter passado pelo comando *disable\_all*.

### Sintaxe:

`drop_all '<expressão regular>'`

### Comando:

`drop_all 'func.*'`



## Apache HBase

### Comandos de Definições no Hbase – IS\_ENABLED

#### IS\_ENABLED

Esse comando só **verificará** se a tabela está habilitada ou não. Caso esteja desabilitada é necessário acionar o outro comando **enable**.

#### Sintaxe:

is\_enabled '<nome da tabela>'

#### Comando:

is\_enabled 'funcionario'

**ALTER (Acréscendo *column family* e limitando versões da coluna)**

Esse comando alterará a definição (*schema*) da família de colunas (*column family*) de uma tabela especificada.

- Alterar uma única ou múltiplas *column family*.
- Deletar *column family*
- Diversas operações são permitidas utilizando atributos específicos de definição de uma tabela.

**Sintaxe:**

```
alter '<nome da tabela>', NAME=><column familyname>, VERSIONS => 5
```

Exemplo comando abaixo para limitar armazenamento até 5 versões da column family

```
alter 'funcionario', NAME => 'hobby', VERSIONS => 5
```

## **ALTER (Dropando *column family*)**

### Sintaxe:

alter '<nome da tabela>', 'delete'=>'column family'

### Comando:

**alter 'funcionario', 'delete'=>'pessoais'**

## ALTER\_STATUS

Com esse comando é possível acompanhar com o status das alterações realizada de uma tabela para todos nós *RegionServer*.

### Sintaxe:

alter\_status '<nome da tabela>'

### Comando:

alter\_status 'funcionario'



# Comandos Manipulação de dados

Beca Data&Analytics 2021

Os comandos abaixo são utilizados para operar **os dados** no HBase.

**put** – Insere/Atualiza um valor em uma determinada célula de uma específica linha de uma tabela.

**get** – Consulta todo o conteúdo de uma linha ou célula em uma tabela.

**delete** – Exclui um valor de uma célula em uma tabela.

**deleteall** – Exclui todas as células de uma linha em específico.

**scan** – Varre toda a tabela retornando os dados contidos.

**count** – Conta e retorna o número de linhas em uma tabela.

**truncate** – Desabilita, exclui e recria uma tabela em específico.

**IMPORTANTE:** Todas as operações de CRUD lista acima estão disponíveis via API Java de fácil utilização sob o pacote `org.apache.hadoop.hbase.client` com os objetos Htable Put e Get.

Apache  
HBaseComandos de manipulação de dados no  
HBase - PUT**PUT**

Comando para inserir um determinado valor em uma célula na coluna ou linha. Também utilizado para atualizar um determinado valor de uma célula para um determinado *rowKey* e *ColumnFamily:colname* já existente.

Sintaxe:

```
put '<nome da tabela>', '<rowkey>', '<columnfamily:colname>', '<valor>'
```

Comando:

```
put 'funcionario', '1', 'pessoais:nome', 'Maria'  
put 'funcionario', '1', 'pessoais:cidade', 'São Paulo'  
put 'funcionario', '1', 'pessoais:cidade', 'Belo Horizonte'  
put 'funcionario', '2', 'profissionais:empresa', 'Everis'
```

*É acionado atualização da chave/valor.*

```
scan 'funcionario', {VERSIONS => 3}
```

## Apache HBase

### Comandos de manipulação de dados no HBase - GET

#### GET

Comando retorna um determinado valor em uma célula na coluna ou linha inteira do *rowKey*.

#### Sintaxe:

get '<nome da tabela>', '<rowkey>', [parâmetros opcionais]

#### Comando:

get 'funcionario', '1'

get 'funcionario', '1', {COLUMN => 'pessoais:cidade'}

get 'funcionario', '1' , {COLUMN => 'pessoais:cidade', VERSIONS=> 3}



## Apache HBase

## Comandos de manipulação de dados no HBase - COUNT

### COUNT

Comando recupera a quantidade de linhas de uma determinada tabela.  
O Intervalo de contagem de linhas pode ser especificado de forma opcional.  
A contagem de linhas é exibida a cada 1000 linhas.

#### Sintaxe:

```
count '<nome da tabela>', CACHE => 1000
```

#### Comando:

```
count 'funcionario', CACHE=> 1000
```

Apache  
HBaseComandos de manipulação de dados no  
HBase - DELETE**DELETE**

Comando remove um determinado valor em uma célula na coluna informadas, também possível remover todas as células de uma rowKey especificada.

Sintaxe:

```
delete '<nome da tabela>', '<rowkey>', '<column name>'
delete '<nome da tabela>', '<rowkey>', '<column name>', '<timestamp>'
deleteall '<nome da tabela>', '<rowkey>'
```

Comando:

```
delete 'funcionario', '1', 'pessoais:cidade'
delete 'funcionario', '1', 'pessoais:cidade', 129019101
deleteall 'funcionario', '1'
```

## Apache HBase

## Comandos de manipulação de dados no HBase - SCAN

### SCAN

Comando remove um determinado valor em uma célula na coluna informadas, também possível remover todas as células de uma rowKey especificada.

#### Sintaxe:

scan '<nome da tabela>', '[parâmetros opcionais]'

#### Comando:

Exibe as ultimas versões de cada rowKey e respectivas colunas

**scan 'funcionario'**

Exibe todas as últimas 10 versões de cada rowKey e respectivas colunas

**scan 'funcionario', {RAW=>true, version=>10}**

## Apache HBase

## Comandos de manipulação de dados no HBase - TRUNCATE

### TRUNCATE

Comando remove todas as linhas e colunas presentes na tabela.  
Internamente o comando realizada a desabilitação da tabela, drop a tabela se ainda está presente, e recria.

#### Sintaxe:

`truncate '<nome da tabela>'`

#### Comando:

`truncate 'funcionario'`

## Colunas com propriedade TTL – Time To Live

É um mecanismo que permite configurar de forma opcional o tempo de permanência de registros em uma tabela ou especificamente de uma coluna.

Configurável em segundos, quantos tempo o registro ficará disponível para consulta.

Os registros são deletados após esse período.

Esse comportamento é feito para todas as versões de cada linha e também válido para quando o HBase é utilizado como intermediários no fluxo de dados ou seja para dados de transição.

A deleção do registro pode ocorrer também como versionamento da linha, como se fosse uma deleção lógica mas os dados vão continuar nas versões anteriores.

## Exemplo TTL – Time To Live

Criar uma tabela com registros que ficam 20 segundos na base.

```
create 'ttl_exemplo', { 'NAME' => 'cf', 'TTL' => 20 }
```

```
put 'ttl_exemplo', 'linha123', 'cf:desc', 'TTL Exemplo'
```

```
get 'ttl_exemplo', 'linha123', 'cf:desc'
```

Aguarde 20 segundos e consulte a linha novamente.

```
get 'ttl_exemplo', 'linha123', 'cf:desc'
```



# Comandos Segurança dos dados

Beca Data&Analytics 2021

O Hbase suporta o controle de acesso às informações das tabelas criadas.

## GRANT

Concede acesso aos tipos de privilégios. R = leitura, W = Gravação, X = Execução, C=criação, A = admin

### Sintaxe:

```
grant <user> <permissions> [ @<namespace> [ <table>[ <column family>[ <column qualifier> ] ] ] ]
```

### Comando:

```
grant 'usrcarga', 'RWXCA'
```

```
grant 'dev001', 'R'
```

```
grant 'dev001', 'R', 'funcionario'
```



## Apache HBase

## Comandos para Segurança dos dados - REVOKE

### REVOKE

Revoga os direitos de acesso de uma tabela para o usuário especificado.

#### Sintaxe:

```
revoke <usuario> [ @<namespace> [ <nome da tabela> [ <column family> [ <column qualifier> ] ] ]
```

#### Comando:

```
revoke 'dev001'  
revoke 'dev001', 'funcionario'
```

Apache  
HBase

## Comandos para Segurança dos dados – USER\_PERMISSION

### USER\_PERMISSION

Lista todas as permissões do usuário em uma tabela.


Sintaxe:

```
user_permission '<nome da tabela>'
```

Comando

**user\_permission**

**user\_permission 'funcionario'**



# Exercícios sobre Comandos

Beca Data&Analytics 2021

## Exercício

**1. Criar uma tabela que representa lista de Cidades e que permite armazenar até 5 versões na Column Family com os seguintes campos:**

Código da cidade como rowKey

Column Family=info

Nome da Cidade

Data de Fundação

Column Family=responsaveis

Nome Prefeito

Data de Posse do Prefeito

Nome Vice prefeito

Column Family=estatisticas

Data da última Eleição

Quantidade de moradores

Quantidade de eleitores


Ano de fundação

## Exercício

2. Inserir 10 cidades na tabela criada de cidades.
3. Realizar uma contagem de linhas na tabela.
4. Consultar só o código e nome da cidade.
5. Escolha uma cidade, consulte os dados dessa cidade em específico antes do próximo passo.
6. Altere para a cidade escolhida os dados de Prefeito, Vice Prefeito e nova data de Posse.
7. Consulte os dados da cidade alterada.
8. Consulte todas as versões dos dados da cidade alterada.

## Exercício

9. Exclua as três cidades com menor quantidade de habitantes e quantidade de eleitores.
10. Liste todas as cidades novamente.
11. Adicione na ColumnFamily “estatísticas”, duas novas colunas de “quantidade de partidos políticos” e “Valor em Reais à partidos” para as 2 cidades mais populosas cadastradas.
12. Liste novamente todas as cidades.



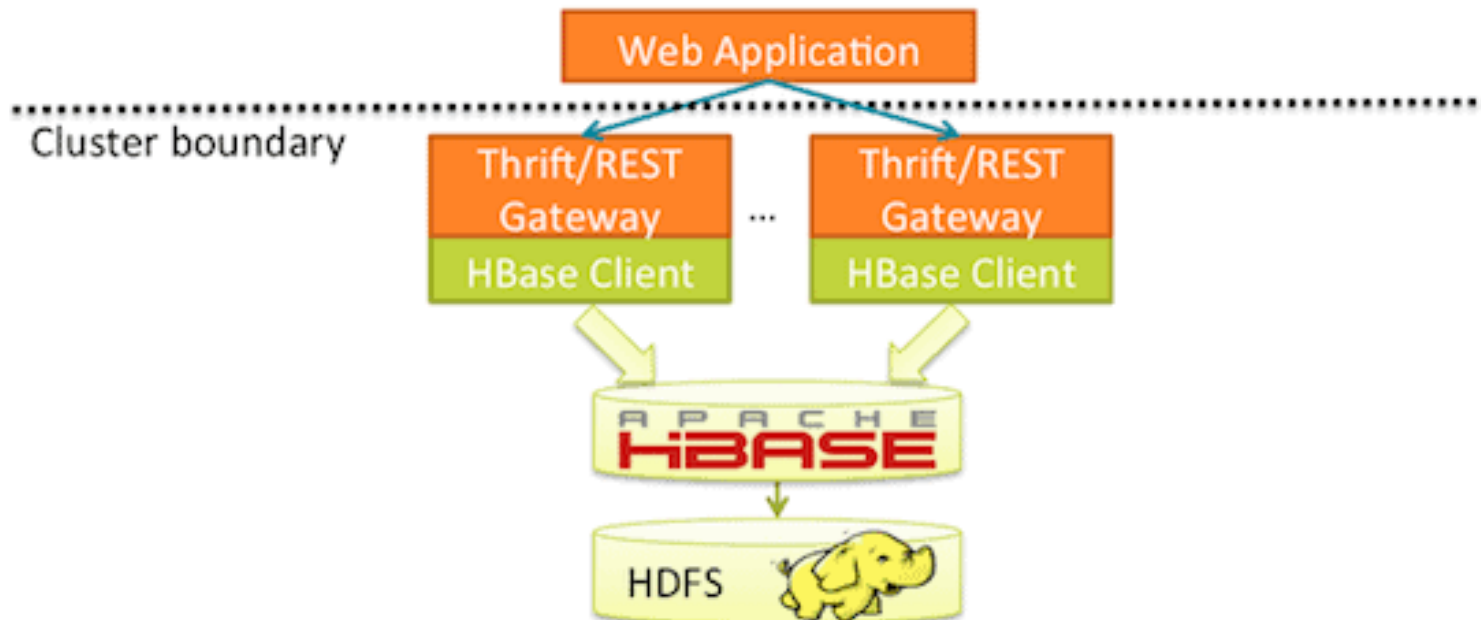
## Cenários de Utilização do HBase

Beca Data&Analytics 2021

## Apache HBase

### Cenário 1 de Utilização do HBase

- Utilizado como banco de dados para aplicações Web e aplicativos móveis.
- Por meio de integrações REST é possível fazer essa integração com o HBase.

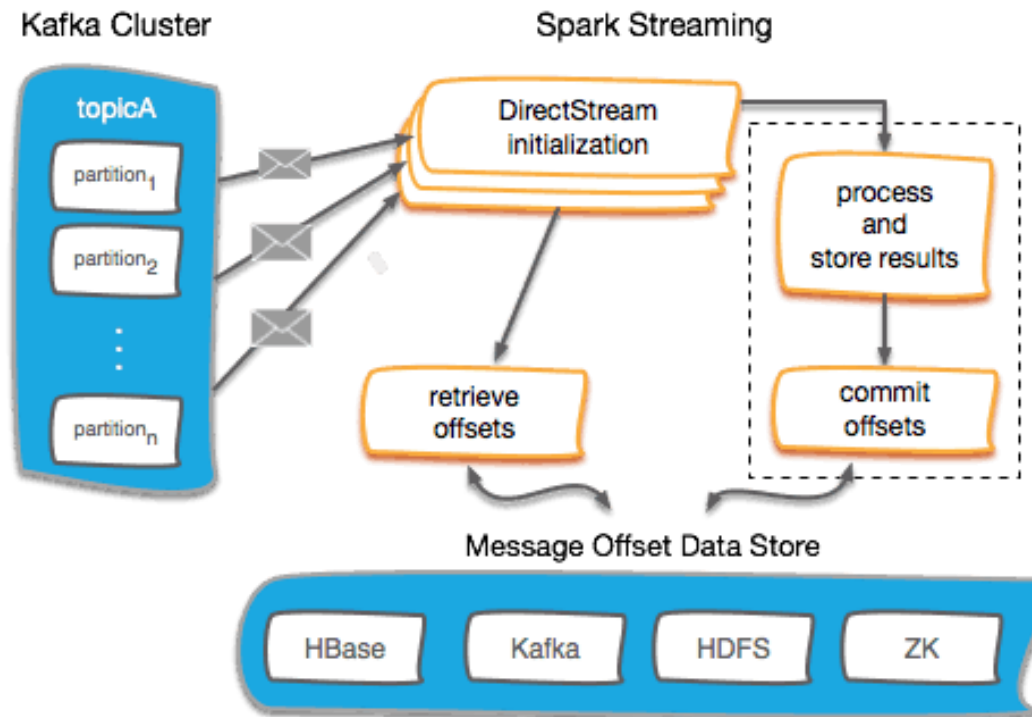




Apache  
HBase

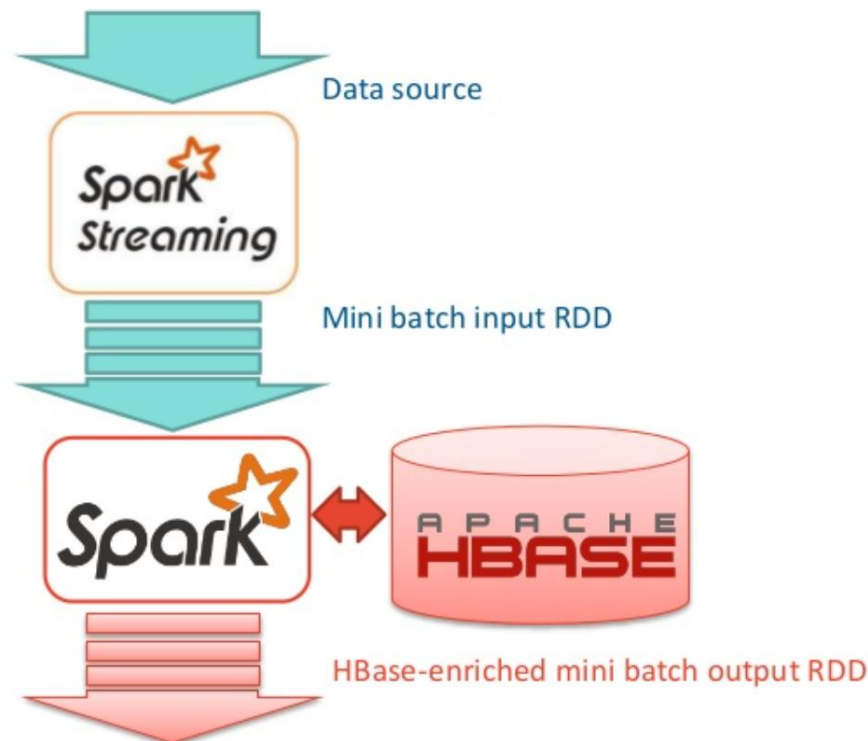
## Cenário 2 de Utilização do Hbase

- Utilizado como respositório para guardar uma cópia / replicador dos dados vindo de serviços de eventos antes da consolidação final.



## Cenário 3 de Utilização do HBase

- Enriquecimento dos dados finais com dados armazenados no HBase. Por exemplo: *lookups*.
- Como há o versionamento padrão do Hbase, é possível ter o rastro das informações alteradas para cada *rowKey*.



A background image showing two people, a man and a woman, smiling and looking at each other in a library setting with bookshelves in the background. The image has a purple tint.

# Operação *Bulk Insert*

Beca Data&Analytics 2021

Nos banco de dados essa é a uma operação que permite fazer a carga massiva de dados.

Normalmente isso ocorre por acionamento de um programa utilitário do banco de dados.

No HBase é possível realizar essa carga por esses meios mais comuns:

- Classe Utilitária do HBase [org.apache.hadoop.hbase.mapreduce.ImportTsv].
- **Intermediários em script com Sqoop, aplicações em Spark, Apache Phoenix, e etc, que implementam indiretamente a Hbase API.**
- External Table do Hive utilizando StorageHandler para consultar dos dados no HBase.
- Hbase API Client.

## Apache HBase

### Exemplo de Operação *Bulk Insert* no HBase

1. Entrar no  
**hbase shell**

2. Criar a tabela employees com o seguinte comando:  
**create 'employees','employee\_data'**

3. Utilizar o arquivo de massa employees.csv e aplicar o comando:  
**sed -i 's/\r\$//' \*.csv**

4. Criar uma pasta no HDFS pelo shell do Linux  
**hadoop fs -mkdir /main\_stage**

Copia os arquivos exportados para o HDFS pelo shell do Linux

**hadoop fs -copyFromLocal <caminho local>/employees.csv /main\_stage/employees.csv**

5. Executar a importação no shell do Linux

**hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=';' -  
Dimporttsv.columns=HBASE\_ROW\_KEY,employee\_data:birth\_date,employee\_data:first\_name,  
employee\_data:last\_name,employee\_data:gender,employee\_data:hire\_date employees  
/main\_stage/employees.csv**

6. Confirmação no Hbase que a importação *Bulk Insert* foi realizada com sucesso.

**hbase shell**

7. Dentro do Shell do HBase confirmar que todas as linhas foram carregadas e a quantidade bate com o número de linhas do arquivo .csv

**count 'employees'**

8. Exibir um registro para confirmar se o mapeamento foi correto

**get 'employees', '10001'**

A background image showing a man and a woman smiling and looking at each other in a library setting, with bookshelves visible in the background. The image has a purple tint.

# Integração HBase com Hive

Beca Data&Analytics 2021

É possível realizar essa integração utilizando a implementação da interface ***StorageHandler*** com a classe ***org.apache.hadoop.hive.hbase.HBaseStorageHandler*** que o Hbase disponibiliza.

No Hive a interface ***StorageHandler*** permite que outras aplicações externas ao Hive (i.e Cassandra, Azure Table, JDBC (MySQL), MongoDB, ElasticSearch, e etc) implementem e disponibilizem operações dessas estruturas e dados armazenados ao Apache Hive.

A idéia é que o Hive tenha visibilidade do metadados quando a tabela é criado no Hive mas os dados e o controle de armazenamento esteja externo.

É uma evolução do conceito de tabela gerenciada (managed) e externa (external) do Hive.

Saber mais sobre StorageHandlers, [saiba mais](#).

Para saber mais detalhes da Integração do Hive e HBase e quais operações estão suportadas, [saiba mais](#).




Utilizando a tabela **employee** utilizada nos exercícios anteriores, vamos dar visibilidade ao Hive dessa estrutura do HBase.

1. Vamos criar um novo schema no Hive.

```
CREATE DATABASE tabelas_hbases;
```

2. Criar a tabela employee no Hive.

```
CREATE EXTERNAL TABLE tabelas_hbases.employee (  
  emp_no INT,  
  birth_date string,  
  first_name string,  
  last_name string,  
  gender string,  
  hire_date string)  
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  
WITH SERDEPROPERTIES("hbase.columns.mapping"=":key, employee_data:birth_date,  
employee_data:first_name,  
employee_data:last_name,employee_data:gender,employee_data:hire_date")  
TBLPROPERTIES("hbase.table.name"="employees",  
"hbase.mapred.output.outputtable"="employees");
```



# Exercício Integração Hive/HBase

Beca Data&Analytics 2021


1. Criar a tabela **salaries** no Hive importando o arquivo salaries.csv para o HDFS.

Obs: Aplicar o comando no arquivo:

```
sed -i 's/\r$//' *.csv
```

2. Criar uma tabela externa no Hive representando a tabela employee do Hbase (Usar o exemplo anterior)

3. Consulte com a tabela **employee** e **salaries**, traga o código, nome do funcionario e salários dos cinco funcionários mais bem pagos.



# Próximos Passos

Beca Data&Analytics 2021

- Aplicação **Apache Phoenix** que habilita capacidades de execução consultas SQL, analítico e OLTP ao HBase.  
As duas distribuições mais conhecidas do ecossistema do Hadoop *on-premise* Cloudera e Horton suportam e em algumas instalações podem permitir a utilização ou já utilizam.  
<https://phoenix.apache.org/>
- Caso queira aprofundar o conhecimento em NoSQL, olhar outros produtos como **Apache Cassandra (versão da DATASTAX)** que tem pontos técnicos mais desenvolvidos e melhorados que o HBase e que não precisa do ecossistema Hadoop (HDFS e Zookeeper) para funcionar. Diferente do Hbase o Cassandra tem um sistema de armazenamento distribuído classificado como AP no teorema CAP. <https://www.datastax.com/>
- Utilizar o Apache Spark para manipular as informações vindo de eventos (Kafka) ou gravando em tópicos as informações e replicando os dados no HBase.
- Dependendo da necessidade de processar mensagens/tabelas/estruturas com **schema** complexos seja o momento de olhar para formatos serializados **Apache Avro, Apache Thrift e Google Protobuf (Protocol Buffers)** que são bem utilizados. É importante ter a clareza em quando modelar tabelas estruturadas/semi-estruturados ou formatos serializados como Avro/Thrift e etc.

A background image showing two young people, a man and a woman, smiling and looking at each other in a library setting. Bookshelves filled with books are visible in the background. The image has a purple tint.

## Referências Utilizadas

Beca Data&Analytics 2021

## Apache HBase

## Referências utilizadas

[Apache HBase Reference Guide](#)

[Secondary Indexing on Hbase](#)

[Cloudera Apache HBase Rest Interface](#)

[Hbase as Offset Managment for Apache Kafka with Apache Spark Streaming](#)

[Ccomo-se-aplica-el-teorema-cap-y-el-reto-de-la-escalabilidad-en-las-rdbms-y-nosql](#)

[NoSQL do teorema CAP para PACCL](#)



Obrigado!



Consulting, Transformation, Technology and Operations