|  |
| --- |
| Para Atualizar docker: apt-get Update & apt-get install apt-file |

[1. Instalação do docker e docker-compose](https://academy.semantix.ai/courses/204/modules/items/10614)

Docker: https://docs.docker.com/get-docker/

Docker-compose: https://docs.docker.com/compose/install/

2. Executar os seguintes comandos, para baixar as imagens do Cluster de Big Data:

* git clone https://github.com/rodrigo-reboucas/docker-bigdata.git
* cd docker-bigdata
* docker-compose pull

3. Iniciar o cluster Hadoop através do docker-compose

* docker-compose up -d

4. Listas os containers em execução

5. Verificar os logs dos containers do docker-compose em execução

6. Verificar os logs do container namenode

7.  Acessar o container namenode

8. Listar  os diretórios do container namenode

9. Parar os containers do Cluster de Big Data

10. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

[Aula 3](https://academy.semantix.ai/courses/204/modules/items/10614)

1. Iniciar o cluster de Big Data

* cd docker-bigdata
* docker-compose up -d

2. Baixar os dados dos exercícios do treinamento

* cd input
* sudo git clone https://github.com/rodrigo-reboucas/exercises-data.git

3. Acessar o container do namenode → ***docker exec -it namenode bash***

4. Criar a estrutura de pastas apresentada a baixo pelo comando: $ hdfs dfs -ls -R /

user/aluno/ ***→ hdfs dfs -mkdir -p/user/aluno/aurelio/data***

<nome>

data

recover → **hdfs dfs -mkdir *user/aluno/aurelio/recover***

delete → **hdfs dfs -mkdir *user/aluno/aurelio/delete***

5. Enviar a pasta “/input/exercises-data/escola” e o arquivo “/input/exercises-data/entrada1.txt” para  data

→ ***hdfs dfs -put /input/exercises-data/escola/ /user/aluno/aurelio/data***

***→ hdfs dfs -put /input/exercises-data/entrada1.txt/ /user/aluno/aurelio/data***

6. Mover o arquivo “entrada1.txt” para recover

→ ***hdfs dfs -mv user/aluno/aurelio/data/entrada1.txt useraluno/rodrigo/recover***

7. Baixar o arquivo do hdfs “escola/alunos.json” para o sistema local /

→

8. Deletar a pasta recover

→ ***hdfs dfs -rm -R /user/aluno/aurelio/recover***

9. Deletar permanentemente o delete

→ **hdfs dfs -rm -skipTrash -R /user/aluno/aurelio/delete (Muito cuidado ao usar esse comando!!!!)**

10. Procurar o arquivo “alunos-modelo” dentro do /user

→ ***hdfs dfs -find /user -name alunos.csv***

11. Mostrar o último 1KB do arquivo “alunos-modelo”

***→ hdfs dfs -tail /user/aluno/aurelio/data/escola/alunos.csv***

12. Mostrar as 2 primeiras linhas do arquivo “alunos-modelo”

→ ***hdfs dfs -cat /user/aluno/aurelio/data/escola/alunos.csv | head -n2***

13. Verificação de soma das informações do arquivo “alunos.cvs”

→ **hdfs dfs -checksum *user/aluno/aurelio/data/escola/alunos.csv***

14. Criar um arquivo em branco com o nome de “teste” no data

→ ***hdfs dfs -touchz /user/aluno/aurelio/data/test***

15. Alterar o fator de replicação do arquivo “teste” para 2

→ ***hdfs dfs -setrep 2 /user/aluno/aurelio/data/test***

16. Ver as informações do arquivo “alunos.cvs”

→ ***hdfs dfs -stat %r /user/aluno/aurelio/data/escola/alunos.csv***

17. Exibir o espaço livre do data e o uso do disco

→ ***hdfs dfs -df /user/aluno/aurelio/data/***

18. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

[Aula](https://academy.semantix.ai/courses/204/modules/items/10614) 4

Introdução a Hive

**Hive - Criação de Tabela Raw**

1. Enviar o arquivo local “/input/exercises-data/populacaoLA/populacaoLA.csv” para o diretório no HDFS “/user/aluno/<nome>/data/populacao”

→ *~/01 Programação/05 Docker/docker-bigdata$* **docker exec -it namenode bash**

→ *root@namenode:/#* **ls /input/exercises-data/populacaoLA/**

→ **hdfs dfs -mkdir /user/aluno/aurelio/data/populacao** (Criar pasta população)

**→ hdfs dfs -put /input/exercises-data/populacaoLA/populacaoLA.csv /user/aluno/aurelio/data/populacao** (Envia o arquivo)

**→ hdfs dfs -ls /user/aluno/aurelio/data/populacao (**conferir se criou)

2. Listar os bancos de dados no Hive

→ **Crtl D**

→ **docker exec -it hive-server bash**

→ **beeline -u jdbc:hive2://localhost:10000 (antes pode ser dado um beeline --help)**

→ **show databases;**

3. Criar o banco de dados <nome>

→ **create database aurelio;**

4. Criar a Tabela Hive no BD <nome>

→ use aurelio;

→ **create table pop(**

**zip\_code int,**

**total\_population int,**

**media\_age float,**

**total\_males int,**

**total\_females int,**

**total\_households int,**

**average\_households\_size float**

**)**

**row format delimited**

**fields terminated by ','**

**lines terminated ‘\n’**

**stored as textfile**

**tblproperties("skip.header.line.count"="1");**

* Tabela interna: pop
* Campos:
  + zip\_code - int
  + total\_population - int
  + median\_age - float
  + total\_males - int
  + total\_females - int
  + total\_households - int
  + average\_household\_size - float
* Propriedades
  + Delimitadores: Campo ‘,’ | Linha ‘\n’
  + Sem Partição
  + Tipo do arquivo: Texto
  + tblproperties("skip.header.line.count"="1")’

5. Visualizar a descrição da tabela pop

→ **desc pop;**

→ **desc formatted pop;**

6. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**Hive - Inserir Dados na Tabela Raw**

1.Visualizar a descrição da tabela pop do banco de dados <nome> (antes tem que iniciar → **docker exec -it hive-server bash)**

→ s**how databases;**

→ **use aurelio;**

→ **desc pop; ou desc formatted pop;**

2.Selecionar os 10 primeiros registros da tabela pop

→ **select \* from pop limit 10;**

3.Carregar o arquivo do HDFS “/user/aluno/<nome>/data/população/populacaoLA.csv” para a tabela Hive pop

→ **load data inpath '/user/aluno/aurelio/data/populacao' overwrite into table pop;** (antes tem que carregar o arquivo)

4.Selecionar os 10 primeiros registros da tabela pop

→ **select \* from pop limit 10;**

5.Contar a quantidade de registros da tabela pop

→ **select count(\*) from pop; (essa função é um mapreduce)**

6. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Hive - Criação de Tabela Particionada**

1. Criar o diretório “/user/aluno/<nome>/data/nascimento” no HDFS

→ docker exec -it namenode hdfs dfs -ls /

→ **hdfs dfs -mkdir /user/aluno/aurelio/data/nascimento**

**ou**

**→ docker exec -it namenode hdfs dfs -mkdir /*user/*aluno/aurelio/data/nascimento**

2. Criar e usar o Banco de dados <nome>

→ **docker exec -it hive-server bash**

→ **beeline -u jdbc:hive2://localhost:10000**

→ **create database aurelio;**

→ **use aurelio**;

3. Criar uma tabela externa no Hive com os parâmetros:

→ **create external table nascimento(**

**nome string,**

**sexo string,**

**frequencia int**

**)**

**partitioned by (ano int)**

**row format delimited**

**fields terminated by ','**

**lines terminated by '\n'**

**stored as textfile**

**location'/user/aluno/aurelio/data/nascimento';**

a) Tabela: nascimento

b) Campos: nome (String), sexo (String) e frequencia (int)

c) Partição: ano

d) Delimitadores:

i) Campo ‘,’

ii)  Linha ‘\n’

e) Salvar

i) Tipo do arquivo: texto

ii) HDFS: '/user/aluno/<nome>/data/nascimento’

4.Adicionar partição ano=2015

→ **alter table nascimento add partition(ano=2015);**

5.Enviar o arquivo local “input/exercises-data/names/yob2015.txt” para o HDFS no diretório /user/aluno/<nome>/data/nascimento/ano=2015

→  **hdfs dfs -put /input/exercises-data/names/yob2015.txt /user/aluno/aurelio/data/nascimento/ano=2015**

6.Selecionar os 10 primeiros registros da tabela nascimento no Hive

→ **select \* from aurelio.nascimento; (ver tudo)**

**→ select \* from aurelio.nascimento limit 10;**

7.Repita o processo do 4 ao 6 para os anos de 2016 e 2017.

**→ use aurelio;**

**→ alter table nascimento add partition(ano=2016);**

**→ alter table nascimento add partition(ano=2017);**

**→ hdfs dfs -ls /user/aluno/aurelio/data/nascimento**

**→ docker exec -it hive-server bash**

**→ beeline -u jdbc:hive2://localhost:10000**

**→ select \* from aurelio.nascimento where ano=2017 limit 10;**

**→ select \* from aurelio.nascimento where ano=2016 limit 10;**

**→ select \* from aurelio.nascimento where ano=2015 limit 10; (ver limite de 10 primeiros)**

**→ select \* from aurelio.nascimento where sexo='M'; (por sexo)**

8. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Hive - Seleção de Tabelas**

1. Selecionar os 10 primeiros registros da tabela nascimento pelo ano de 2016

→ **select \* from aurelio.nascimento where ano=2016 limit 10;**

**→ select \* from nascimento where ano=2016 limit 10;**

2. Contar a quantidade de nomes de crianças nascidas em 2017

→ **select count(nome) as qtd from nascimento where ano=2017;**

3. Contar a quantidade de crianças nascidas em 2017

→ **select sum(frequencia) as qtd from nascimento where ano=2017;**

4. Contar a quantidade de crianças nascidas por sexo no ano de 2015

→ **select sexo, sum(frequencia) as qtd from nascimento where ano=2015 group by sexo;**

5. Mostrar por ordem de ano decrescente a quantidade de crianças nascidas por sexo

**→ select ano, sexo, sum(frequencia) as qtd from nascimento group by ano, sexo; (ordem crescente)**

**→ select ano, sexo, sum(frequencia) as qtd from nascimento group by ano, sexo order by ano desc; (ordem decrescente)**

6. Mostrar por ordem de ano decrescente a quantidade de crianças nascidas por sexo com o nome iniciado com ‘A’

→ **select ano, sexo, sum(frequencia) as qtd from nascimento where nome like ‘A%’ group by ano, sexo order by ano desc;**

7. Qual nome e quantidade das 5 crianças mais nascidas em 2016

→ **select nome, max(frequencia) as qtd from nascimento where ano=2016 group by nome order by qtd desc limit 5;**

8. Qual nome e quantidade das 5 crianças mais nascidas em 2016 do sexo masculino e feminino

→ **select nome, max(frequencia) as qtd, sexo from nascimento where ano=2016 group by nome,sexo order by qtd desc limit 5;**

9. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Hive - Criação de Tabelas Otimizadas**

1. Usar o banco de dados <nome>

→ **use aurelio;**

2. Selecionar os 10 primeiros registros da tabela pop

→ **select \* from pop limit 10;**

3. Criar a tabela pop\_parquet no formato parquet para ler os dados da tabela pop

**→ create table pop\_parquet(**

**zip\_code int,**

**total\_population int,**

**median\_age float,**

**total\_males int,**

**total\_females int,**

**total\_households int,**

**average\_household\_size float**

**) stored as parquet;**

4. Inserir os dados da tabela pop na pop\_parquet

**→ insert into pop\_parquet select \* from pop;**

5. Contar os registros da tabela pop\_parquet

**→ select count(\*) from pop\_parquet;**

6. Selecionar os 10 primeiros registros da tabela pop\_parquet

**→ select \* from pop\_parquet limit 10;**

7. Criar a tabela pop\_parquet\_snappy no formato parquet com compressão Snappy para ler os dados da tabela pop

**→ create table pop\_parquet\_snappy(**

**zip\_code int,**

**total\_population int,**

**median\_age float,**

**total\_males int,**

**total\_females int,**

**total\_households int,**

**average\_household\_size float)**

**stored as parquet**

**tblproperties('parquet.compress'='SNAPPY');**

8. Inserir os dados da tabela pop na pop\_parquet\_snappy

→ **insert into pop\_parquet\_snappy select \* from pop;**

9. Contar os registros da tabela pop\_parquet\_snappy

→ **select count(\*) from pop\_parquet\_snappy;**

10. Selecionar os 10 primeiros registros da tabela pop\_parquet\_snappy

**→ select \* from pop\_parquet\_snappy limit 10;**

11. Comparar as tabelas pop, pop\_parquet e pop\_parquet\_snappy no HDFS.

**→ hdfs dfs -ls /user/hive/warehouse/aurelio.db**

**→ hdfs dfs -du -h /user/hive/warehouse/aurelio.db**

12. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**MySQL - Verificar e Instalar os Banco de Dados de testes**

1. Copiar os dados do local para o contêiner database

→ **ls input/exercises-data/db-sql/**

$ **docker cp input/exercises-data/db-sql/ database:/**

2. Acessar o contêiner database

→ docker exec -it database bash

$ docker exec -it database bash

3. Instalar Banco de Dados de testes

→ **mysql -h localhost -u root -psecret (para entra no mysql)**

* Diretório /db-sql - BD employees (Já existe)

**$ cd /db-sql**

**$ mysql -psecret < employees.sql ou mysql -psecret ou mysql -p’secret’**

* **Diretório /db-sql/sakila - BD sakila**

**$ cd /db-sql/sakila/**

**$ mysql -psecret < sakila-mv-schema.sql  
$ mysql -psecret < sakila-mv-data.sql (< para executa o script)**

**→ use sakila;**

**→ select \* from rental limit 10;**

4. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**no namenode para importar sqoop hdfs.**

**→ sqoop import database -> hdfs(datanode)**

como acessar **sqoop**

**→ docker exec -it namenode bash**

**→ sqoop version → sqoop help → sqoop help import**

**→**

Listar Banco de dados

**Sqoop -  Pesquisa e Criação de Tabelas**

Todos os comandos precisam ser executados pelo Sqoop. ( esta no namenode)

→ **docker exec -it namenode bash**

1. Mostrar todos os databases

**→ sqoop list-databases –-connect jdbc:mysql://database --username root --password secret**

2. Mostrar todas as tabelas do bd employees

**→ sqoop list-tables --connect jdbc:mysql://database/employees --username root --password secret**

3. Inserir os valores ('d010', 'BI') na tabela departments do bd employees

Primeiro realizar uma consulta ou pesquisa → **sqoop eval –-connect jdbc:mysql://database/employees --username root --password secret –-query ”select \* from departments”**

para inserir:

→ **sqoop eval –-connect jdbc:mysql://database/employees --username root --password secret –-query ”insert into departments values(‘d010’,’BI’)”**

4. Pesquisar todos os registros da tabela departments

→ **sqoop eval –-connect jdbc:mysql://database/employees --username root --password secret –-query ”select \* from departments”**

5. Criar a tabela benefits(cod int(2)  AUTO\_INCREMENT PRIMARY KEY, name varchar(30)) no bd employees

→ **sqoop eval --connect jdbc:mysql://database/employees --username root --password secret --query "create table benefits(cod int(2) AUTO\_INCREMENT PRIMARY KEY, name varchar(30))”**

6. Inserir os valores (null,'food vale') na tabela benefits

→ **sqoop eval --connect jdbc:mysql://database/employees --username root --password secret --query "insert into benefits values(null,'food vale')"**

7. Pesquisar todos os registros da tabela benefits

→ **sqoop eval –-connect jdbc:mysql://database/employees --username root --password secret –-query ”select \* from benefits”**

8. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Sqoop - Importação BD Employees**

1. Pesquisar os 10 primeiros registros da tabela employees do banco de dados employees

→ **sqoop eval --connect jdbc:mysql://database/employees --username root --password secret --query "select \* from employees limit 10"**

2. Realizar as importações referentes a tabela employees e para validar cada questão,  é necessário visualizar no HDFS\*

1. Importar a tabela employees, no warehouse  /user/hive/warehouse/db\_test\_a

→  **sqoop import --table employees --connect jdbc:mysql://database/employees --username root --password secret --warehouse-dir /user/hive/warehouse/db\_test\_a**

**→ hdfs dfs -ls /user/hive/warehouse/db\_test\_a/employees (para listar)**

**→ hdfs dfs -cat /user/hive/warehouse/db\_test\_a/employees/part-m-00000 | head -n3 (ver os 3 primeiros)**

1. Importar todos os funcionários do gênero masculino, no warehouse  /user/hive/warehouse/db\_test\_b

→ **sqoop import --table employees --connect jdbc:mysql://database/employees --username root --password secret --where "gender='M'" --warehouse-dir /user/hive/warehouse/db\_test\_b**

**→ hdfs dfs -ls -h /user/hive/warehouse/db\_test\_b/employees (para verificar)**

**→ hdfs dfs -cat /user/hive/warehouse/db\_test\_b/employees/part-m-00000 | head -n3 (Leitura dos dados)**

1. importar o primeiro e o último nome dos funcionários com os campos separados por tabulação, no warehouse  /user/hive/warehouse/db\_test\_c

→ **sqoop import --table employees --connect jdbc:mysql://database/employees --username root --password secret --columns "first\_name,last\_name" --fields-terminated-by '\t' --warehouse-dir /user/hive/warehouse/db\_test\_c**

**→ hdfs dfs -cat /user/hive/warehouse/db\_test\_c/employees/part-m-00000 | head -n 10**

1. Importar o primeiro e o último nome dos funcionários com as linhas separadas por “ : ” e salvar no mesmo diretório da questão 2.C

→ sqoop import --table employees --connect jdbc:mysql://database/employees --username root --password secret --columns "first\_name,last\_name" --lines-terminated-by ':' --warehouse-dir /user/hive/warehouse/db\_test\_c -delete-target-dir

\* Dica para visualizar no HDFS:

$ hdfs dfs -cat /.../db\_test/nomeTabela/part-m-00000 | head -n 5

3. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Sqoop - Importação BD Employees- Otimização**

Realizar com uso do **MySQL**

**→ docker exec -it database bash**

**→ mysql -h localhost -u root -psecret ou mysql -psecret**

**→ use employees;**

**→ select \* from titles limit 10;**

1. Criar a tabela cp\_titles\_date, contendo a cópia da tabela titles com os campos title e to\_date

→ **create table cp\_titles\_date select title, to\_date from titles;**

**→ select \* from cp\_titles\_date limit 10;**

2. Pesquisar os 15 primeiros registros da tabela cp\_titles\_date

→ **select \* from cp\_titles\_date limit 15;**

3. Alterar os registros do campo to\_date para null da tabela cp\_titles\_date, quando o título for igual a Staff

→ **update cp\_titles\_date set to\_date=NULL where title='staff';**

Realizar com uso do **Sqoop**- Importações no warehouse **/user/hive/warehouse/db\_test\_<numero\_questao>** e visualizar no HDFS

4. Importar a tabela titles com 8 mapeadores no formato parquet

→ **sqoop import --table titles --connect jdbc:mysql://database/employees --username root --password secret -m 8 --as-parquetfile --warehouse-dir /user/hive/warehouse/db\_test2\_4**

**→ hdfs dfs -ls -h /user/hive/warehouse/db\_test2\_4**

**→ hdfs dfs -ls -h /user/hive/warehouse/db\_test2\_4/titles**

5. Importar a tabela titles com 8 mapeadores no formato parquet e compressão snappy

→ **sqoop import --table titles --connect jdbc:mysql://database/employees --username root --password secret -m 8 --as-parquetfile --warehouse-dir /user/hive/warehouse/db\_test2\_5 --compress --compression-codec org.apache.hadoop.io.compress.SnappyCodec**

**→ hdfs dfs -ls -h /user/hive/warehouse/db\_test2\_5/titles**

6. Importar a tabela cp\_titles\_date com 4 mapeadores (erro)

→ **sqoop import --table cp\_titles\_date --connect jdbc:mysql://database/employees --username root --password secret -m 4 --warehouse-dir /user/hive/warehouse/db\_test2\_6**

**→ sqoop list-tables --connect jdbc:mysql://database/employees --username root --password secret (para listar tabelas)**

Importar a tabela cp\_titles\_date com 4 mapeadores divididos pelo campo título no warehouse **/user/hive/warehouse/db\_test2\_title**

**→ sqoop import -Dorg.apache.sqoop.splitter.allow\_text\_splitter=true --table cp\_titles\_date --connect jdbc:mysql://database/employees --username root --password secret -m 4 --warehouse-dir /user/hive/warehouse/db\_test2\_title --split-by title**

Importar a tabela cp\_titles\_date com 4 mapeadores divididos pelo campo data no warehouse **/user/hive/warehouse/db\_test2\_date**

**→ sqoop import -Dorg.apache.sqoop.splitter.allow\_text\_splitter=true --table cp\_titles\_date --connect jdbc:mysql://database/employees --username root --password secret -m 4 --warehouse-dir /user/hive/warehouse/db\_test2\_date --split-by to\_date**

Qual a diferença dos registros nulos entre as duas importações?

→ **hdfs dfs -count -h /user/hive/warehouse/db\_test2\_date**

**→ hdfs dfs -ls -h -R /user/hive/warehouse/db\_test2\_date**

**→ hdfs dfs -count -h /user/hive/warehouse/db\_test2\_title**

**→ hdfs dfs -ls -h -R /user/hive/warehouse/db\_test2\_title**

7. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**Sqoop - Importação BD Employees – Formato e Compressão**

Este exercício não é obrigatório, é apenas um bônus, pois sua resolução exige muito tempo, por isso não irei fazer a correção

Importar a tabela employees com 1 mapeador no warehouse /user/hive/warehouse/db\_format, afim de completar a tabela abaixo.

* Formato

o --as-textfile (padrão)

o --as-parquetfile

o ---as-avrodatafile

o --as-sequencefile

* Compactação

o sem compactação (padrão)

o --compress --compression-codec=snappy

o --compress ou --compression --compression-codec=gzip

o --compress --compression-codec=bzip2

o --compress --compression-codec=deflate

o --compress --compression-codec=lz4

|  |  |  |
| --- | --- | --- |
| Formato de armazenamento | Compressão | Tamanho da tabela |
| Texto | - |  |
| Texto | Snappy |  |
| Texto | Gzip |  |
| Texto | lz4 |  |
| Avro | - |  |
| Avro | Snappy |  |
| Avro | Gzip |  |
| Avro | lz4 |  |
| Parquet | - |  |
| Parquet | Snappy |  |
| Parquet | Gzip |  |
| Parquet | lz4 |  |

Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Sqoop - Importação BD Sakila – Carga Incremental**

Realizar com uso do **MySQL**

**→ docker exec -it database bash**

**→ mysql -psecret**

**→ show tables;**

→ **use sakila;**

1. Criar a tabela cp\_rental\_append, contendo a cópia da tabela rental com os campos rental\_id e rental\_date

→ **create table cp\_rental\_append select rental\_id, rental\_date from rental;**

**→ select \* from cp\_rental\_append limit 15 ;**

2 .Criar a tabela cp\_rental\_id e cp\_rental\_date, contendo a cópia da tabela cp\_rental\_append

**→ create table cp\_rental\_id select \* from cp\_rental\_append;**

**→ create table cp\_rental\_date select \* from cp\_rental\_append;**

**→ select \* from cp\_rental\_id limit 5 ;**

**→ select \* from cp\_rental\_date limit 5 ;**

Realizar com uso do **Sqoop**- Importações no warehouse /user/hive/warehouse/db\_test3 e visualizar no HDFS

→ docker exec -it namenode bash

3. Importar as tabelas cp\_rental\_append, cp\_rental\_id e cp\_rental\_date com 1 mapeador

**→ sqoop import --connect jdbc:mysql://database/sakila --username root --password secret –-warehouse-dir /*user*/hive/warehouse/db\_test3 -m 1 –-table cp\_rental\_append**

**→ sqoop import --connect jdbc:mysql://database/sakila --username root --password secret –-warehouse-dir /*user*/hive/warehouse/db\_test3 -m 1 –-table cp\_rental\_id**

**→ sqoop import --connect jdbc:mysql://database/sakila --username root --password secret –-warehouse-dir /*user*/hive/warehouse/db\_test3 -m 1 –-table cp\_rental\_date**

**→ hdfs dfs -ls -R /user/hive/warehouse/db\_test3/cp\_rental\_append (Conferir)**

**→ hdfs dfs -ls -R /user/hive/warehouse/db\_test3/cp\_rental\_id (Conferir)**

**→ hdfs dfs -ls -R /user/hive/warehouse/db\_test3/cp\_rental\_date (Conferir)**

**→ hdfs dfs -tail /user/hive/warehouse/db\_test3/cp\_rental\_append/part-m-00000**

**→ hdfs dfs -tail /user/hive/warehouse/db\_test3/cp\_rental\_data/part-m-00000**

Realizar com uso do **MySQL**

4. Executar o sql /db-sql/sakila/insert\_rental.sql no container do database

**$ docker exec -it database bash**

**$ cd /db-sql/sakila**

**$ mysql -psecret < insert\_rental.sql**

**→ cat insert\_rental.sql**

 → **root@database:/db-sql/sakila# mysql -psecret < insert\_rental.sql**

Realizar com uso do **Sqoop**- Importações no warehouse /user/hive/warehouse/db\_test3 e visualizar no HDFS

5. Atualizar a tabela cp\_rental\_append no HDFS anexando os novos arquivos

→ **sqoop import --table cp\_rental\_append --connect jdbc:mysql://database/sakila --username root --password secret --warehouse-dir /user/hive/warehouse/db\_test3 -m 1 --append**

6. Atualizar a tabela cp\_rental\_id no HDFS de acordo com o último registro de rental\_id, adicionando apenas os novos dados.

→ **sqoop eval --connect jdbc:mysql://database/sakila --username root --password secret --query "select \* from cp\_rental\_append order by rental\_id desc limit 5" (para ver o ultimo colocar 1)**

**→ sqoop import --table cp\_rental\_id --connect jdbc:mysql://database/sakila --username root --password secret --warehouse-dir /user/hive/warehouse/db\_test3 -m 1 --incremental append --check-column rental\_id --last-value 16049 (16049 o ultimo valor)**

7. Atualizar a tabela cp\_rental\_date no HDFS de acordo com o último registro de rental\_date, atualizando os registros a partir desta data.

→ **sqoop import --table cp\_rental\_date --connect jdbc:mysql://database/sakila --username root --password secret --warehouse-dir /user/hive/warehouse/db\_test3 -m 1 --incremental lastmodified --merge-key rental\_id --check-column rental\_date --last-value '2005-08-23 22:50:12.0'**

**→ hdfs dfs -ls -h -R /user/hive/warehouse/db\_test3 (para visualizar)**

8. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**Sqoop - Importação para o Hive e Exportação - BD Employees**

1. Importar a tabela employees.titles do MySQL para o diretório /user/aluno/<nome>/data com 1 mapeador.

→ **sqoop import --table titles --connect jdbc:mysql://database/employees --username root --password secret --warehouse-dir /user/aluno/aurelio/data -m 1**

**→ hdfs dfs -ls /user/aluno/aurelio/data/titles/**

2. Importar a tabela employees.titles do MySQL para uma **tabela Hive**no banco de dados seu nome com 1 mapeador.

→ **sqoop import --table titles --connect jdbc:mysql://database/employees --username root --password secret -m 1 --hive-import --hive-table aurelio.titles**

**→ hdfs dfs -ls /user/hive/warehouse/aurelio.db**

3. Selecionar os 10 primeiros registros da tabela titles no **Hive**.

→ **hdfs dfs -cat /user/hive/warehouse/aurelio.db/titles/part-m-00000 | head -n 10**

ou

→ **docker exec -it hive-server bash**

**→ beeline -u jdbc:hive2://localhost:10000**

**→ use aure lio;**

**→ select \* from titles limit 10;**

4. Deletar os registros da tabela employees.titles do MySQL e verificar se foram apagados, através do Sqoop

**→ docker exec -it namenode bash**

**→ sqoop eval --connect jdbc:mysql://database/employees --username root --password secret --query "select \* from titles limit 10"**

para deletar:

**→ sqoop eval --connect jdbc:mysql://database/employees --username root --password secret --query "truncate table titles"**

**→ sqoop eval --connect jdbc:mysql://database/employees --username root --password secret --query "select \* from titles limit 10"**

5. Exportar os dados do diretório /user/hive/warehouse/<nome>.db/data/titles para a tabela do MySQL  employees.titles.

→ **sqoop export --table titles --connect jdbc:mysql://database/employees --username root --password secret --export-dir /user/aluno/aurelio/data/titles**

6. Selecionar os 10 primeiros registros registros da tabela employees.titles do MySQL.

→ **sqoop eval --connect jdbc:mysql://database/employees --username root --password secret --query "select \* from titles limit 10"**

7. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**Acessar HBASE:**

**verificar containers**

**→ docker ps**

**acessar Hbase bash:**

**→ docker exec -it hbase-master bash**

**→ hbase shell**

**se está ativo → status**

**HBase - Exercícios**

1. Criar a tabela ‘controle’ com os dados:

* Chave: id
* Família de Coluna: produto e fornecedor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | produto | | fornecedor | |
| nome | qtd | nome | estado |
| 1 | ram | 100 | TI Comp | SP |
| 2 | hd | 50 | Peças PC | MG |
| 3 | mouse | 150 | Inf Tec | SP |

→ create 'controle', 'produto','fornecedor' ou create 'controle',{NAME=> 'produto'},{NAME=>'fornecedor'}

inserir dados;

**→ put ‘controle’, ‘1’,’produto:nome’,’ram’**

**→ put 'controle', '2','produto:nome','hd'**

**→ put 'controle', '3','produto:nome','mouse'**

**→ put 'controle', '1','produto:qtd','100'**

**→ put 'controle', '2','produto:qtd','50'**

**→ put 'controle', '3','produto:qtd','150**

**→ put 'controle', '1','fornecedor:nome','TI Comp’**

**→ put 'controle', '2','fornecedor:nome','Peças pc'**

**→ put 'controle', '3','fornecedor:nome','Inf Tec'**

**→ put 'controle', '1','fornecedor:estado','SP'**

**→ put 'controle', '3','fornecedor:estado','SP'**

**→ put 'controle', '2','fornecedor:estado','MG'**

2. Listar as tabelas e verificar a estrutura da tabela ‘controle’

**→ list**

**→ describe ‘controle’**

3. Contar o número de registros da tabela ‘controle’

→ **count ‘controle’**

4. Alterar  a família de coluna produto para 3 versões

→ **alter 'controle',{NAME=>'produto',VERSIONS=>3}**

5. Alterar a quantidade para 200 do id 2

→ **put 'controle','2', 'produto:qtd','200'**

6. Pesquisar as versões do id 2  da coluna quantidade

**→ get 'controle','2',{COLUMNS=>'produto',VERSIONS=>2}**

**→ get 'controle','2',{COLUMNS=>'produto:qtd',VERSIONS=>2}**

**→ get 'controle','2',{COLUMNS=>['produto:qtd','fornecedor:nome'],VERSIONS=>2}**

7. Excluir os id do estado de SP

→ **scan 'controle', {COLUMNS=> 'fornecedor:estado',LIMIT => 5, FILTER=> "ValueFilter(=,'binary:SP')"}**

→ **deleteall 'controle','1'**

**→ deleteall 'controle','3'**

8. Deletar a coluna estado da chave 2

**→ delete ‘controle’, ‘2’, ‘fornecedor:estado’**

9. Pesquisar toda a tabela controle

→ **scan ‘controle’**

10. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**Introdução Spark**

# Acessar o Spark

Adicionar o jar para salvar tabelas Hive:

→ **curl -O** **https://repo1.maven.org/maven2/com/twitter/parquet-hadoop-bundle/1.6.0/parquet-hadoop-bundle-1.6.0.jar**

**→ docker cp parquet-hadoop-bundle-1.6.0.jar spark:/opt/spark/jars**

**para acessar:**

**→ docker exec -it spark bash**

**→ ls *opt/*spark/jars**

**para abrir spark:**

**→ spark-shell**

**Comandos spark:**

**Spark - Exercícios de DataFrame**

1. Enviar o diretório local “/input/exercises-data/juros\_selic” para o HDFS em “/user/aluno/<nome>/data”

**→ docker exec -it namenode bash**

**→ hdfs dfs -put /input/exercises-data/juros\_selic/ /user/aluno/aurelio/data**

2. Criar o DataFrame (spark)jurosDF para ler o arquivo no HDFS “/user/aluno/<nome>/data/juros\_selic/juros\_selic.json”

**→ docker exec -it spark bash**

**→ spark-shell**

**→val jurosDF = spark.read.json("/user/aluno/aurelio/data/juros\_selic/juros\_selic.json")**

3. Visualizar o Schema do jurosDF

→ **jurosDF.printSchema**

4. Mostrar os 5 primeiros registros do jutosDF

→ **jurosDF.show(5,false)**

5. Contar a quantidade de registros do jurosDF

→ **jurosDF.count**

6. Criar o DataFrame jurosDF10 para filtrar apenas os registros com o campo “valor” maior que 10

→ **val jurosDF10 = jurosDF.where("valor > 10")**

**tomar cuidado:**

→ **val jurosDF10 = jurosDF.where("valor > 10").show(10) //** deixa de ser dataframe e vira dataset

7. Salvar o DataFrame jurosDF10  como tabela Hive “<nome>.tab\_juros\_selic”

→ **jurosDF10.write.saveAsTable("aurelio.tab\_juros\_selic")**

8. Criar o DataFrame jurosHiveDF para ler a tabela “<nome>.tab\_juros\_selic”

→ **spark.read.table("aurelio.tab\_juros\_selic")**

9. Visualizar o Schema do jurosHiveDF

→ **jurosHiveDF.printSchema**

10. Mostrar os 5 primeiros registros do jurosHiveDF

→ **jurosHiveDF.show(5)**

11. Salvar o DataFrame jurosHiveDF no HDFS no diretório “/user/aluno/nome/data/save\_juros” no formato parquet

→ **jurosHiveDF.write.save("/user/aluno/aurelio/data/save\_juros")**

12. Visualizar o save\_juros no HDFS

**→ docker exec -it namenode bash**

**→ hdfs dfs -ls /user/aluno/aurelio/data/save\_juros**

13. Criar o DataFrame jurosHDFS para ler o diretório do “save\_juros” da questão 8

**→ val jurosHDFS = spark.read.parquet("/user/aluno/aurelio/data/save\_juros")**

**ou**

**→ val jurosHDFS = spark.read.load("/user/aluno/aurelio/data/save\_juros")**

**ou**

**→ val jurosHDFS = spark.read.parquet("hdfs://namenode:8020/user/aluno/aurelio/data/save\_juros")**

14. Visualizar o Schema do jurosHDFS

→ **jurosHDFS.printSchema**

15. Mostrar os 5 primeiros registros do jurosHDFS

→ **jurosHDFS.show(5)**

16. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Spark - Exercícios de Esquema e Join**

1. Criar o DataFrame **alunosDF** para ler o arquivo no hdfs “/user/aluno/<nome>/data/escola/alunos.csv” sem usar as “option”

→  **val alunosDF = spark.read.csv("/user/aluno/aurelio/data/escola/alunos.csv")**

2. Visualizar o esquema do alunosDF

→ **alunosDF.printSchema**

3. Criar o DataFrame **alunosDF** para ler o arquivo “/user/aluno/<nome>/data/escola/alunos.csv” com a opção de Incluir o cabeçalho

→ **val alunosDF = spark.read.option(“header”,true).csv(“/user/aluno/aurelio/data/escola/alunos.csv”)**

**→ val alunosDF = spark.read.csv("/user/aluno/aurelio/data/escola/alunos.csv").show(3)**

4. Visualizar o esquema do alunosDF

→ **alunosDF.printSchema**

5. Criar o DataFrame **alunosDF** para ler o arquivo “/user/aluno/<nome>/data/escola/alunos.csv” com a opção de Incluir o cabeçalho e inferir o esquema

→ **val alunosDF = spark.read.option("header",true).option("inferSchema","true").csv("/user/aluno/aurelio/data/escola/alunos.csv")**

6. Visualizar o esquema do alunosDF

→ **alunosDF.printSchema**

7. Salvar o DaraFrame alunosDF como tabela Hive “tab\_alunos” no banco de dados <nome>

→ **alunosDF.write.saveAsTable("aurelio.tab\_alunos")**

8. Criar o DataFrame **cursosDF** para ler o arquivo “/user/aluno/<nome>/data/escola/cursos.csv” com a opção de Incluir o cabeçalho e inferir o esquema

→ **val cursosDF = spark.read.option("header",true).option("inferSchema","true").csv("/user/aluno/aurelio/data/escola/cursos.csv")**

9. Criar o DataFrame **alunos\_cursosDF** com o inner join do alunosDF e cursosDF quando o id\_curso dos 2 forem o mesmo

→ **val alunos\_cursosDF = alunosDF.join(cursosDF,"id\_curso")**

10. Visualizar os dados, o esquema e a quantidade de registros do alunos\_cursosDF

→ **alunos\_cursosDF.show(10)**

11. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Spark - Exercícios da API Catalog**

Realizar os exercícios usando a API Catalog.

1. Visualizar todos os banco de dados

→ **spark.catalog.listDatabases.show(false)**

2. Definir o banco de dados “seu-nome” como principal

→ **spark.catalog.setCurrentDatabase(“aurelio”)**

3. Visualizar todas as tabelas do banco de dados “seu-nome”

→ **spark.catalog.listTables.show**

→

4. Visualizar as colunas da tabela tab\_alunos

→ **spark.catalog.listColumns(“tab\_alunos”).show**

5.  Visualizar os 10 primeiros registos da tabela "tab\_alunos" com uso do spark.sql

→ **spark.read.table(“tab\_alunos”).show(10)**

→ **spark.sql(“select \* from tab\_alunos limit 10”).show()**

6. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **ok**

**Spark - Exercícios de SQL Queries vs Operações de DataFrame**

Realizar as seguintes consultas usando SQL queries e transformações de DataFrame na tabela “tab\_alunos” no banco de dados <nome>

1. Visualizar o id e nome dos 5 primeiros registros

→ **spark.catalog.listColumns("tab\_alunos").show**

**→ spark.sql("select id\_discente,nome from tab\_alunos limit 5").show**

**→ spark.read.table("tab\_alunos").show(5)**

**→ val alunosHiveDF = spark.read.table("tab\_alunos")**

**→ alunosHiveDF.select("id\_discente","nome").limit(5).show**

2. Visualizar o id, nome e ano quando o ano de ingresso for maior ou igual a 2018

**→ spark.sql("select id\_discente,nome,ano\_ingresso from tab\_alunos where ano\_ingresso >=2018").show**

**→alunosHiveDF.select("id\_discente","nome","ano\_ingresso").where("ano\_ingresso >= 2018").show**

3. Visualizar por ordem alfabética do nome o id, nome e ano quando o ano de ingresso for maior ou igual a 2018

→ **spark.sql("select id\_discente,nome,ano\_ingresso from tab\_alunos where ano\_ingresso >=2018 order by nome desc").show**

**→ alunosHiveDF.select("id\_discente","nome","ano\_ingresso").where("ano\_ingresso >= 2018").orderBy($"nome".desc).show**

**ou**

**→ alunosHiveDF.select("id\_discente","nome","ano\_ingresso").where("ano\_ingresso >= 2018").orderBy(alunosHiveDF("nome").desc).show**

4. Contar a quantidade de registros do item anterior

**→ alunosHiveDF.select("id\_discente","nome","ano\_ingresso").where("ano\_ingresso >= 2018").orderBy(alunosHiveDF("nome").desc).count**

**→ spark.sql("select count(id\_discente) from tab\_alunos where ano\_ingresso >= 2018").show**

5. Clicar no botão de **Enviar Tarefa**, e enviar o texto: **o.k.**

**https://web.dio.me/track/orange-tech-backend**

