TREINAMENTO DATABRICKS Delta Lake

Apostila com os comandos das videoaulas Notebooks

Utilização dos comandos
Apostila criada por Grimaldo Oliveira

Curso DATABRICKS DELTA LAKE
Prof. Grimaldo Oliveira
grimaldo_lopes@hotmail.com

Conheça o Professor Grimaldo Oliveira

Sou professor das pós-graduações das universidades **UNIFACS, CATÓLICA DO SALVADOR** e **ISL Wyden**. **Mestre** pela **Universidade do Estado da Bahia** (UNEB) no Curso de Mestrado Profissional Gestão e Tecnologias Aplicadas à Educação (GESTEC). Possuo Especialização em Análise de Sistemas pela Faculdade Visconde de Cairu e **Bacharelado** em **Estatística** pela Universidade Federal da Bahia. Atuo profissionalmente como consultor há mais de **15 anos nas áreas de Data Warehouse, Mineração de Dados, Ferramentas de Tomada de Decisão e Estatística**.

BI PRO

Idealizador do treinamento online **BI PRO** com + de 10 módulos contendo todas as disciplinas para formação completa na área de dados. Quem participa do **BI PRO** tem acesso gratuito: todos os meus cursos de dados da Udemy, + ebook BI COMO DEVE SER - O Guia Definitivo, espaço de mentoria para retirada de dúvidas, respostas das atividades. Acesse www.bipro.com.br

Autor do eBook **BI COMO DEVE SER - O Guia Definitivo,** com ele você poderá entender os conceitos e técnicas utilizados para o desenvolvimento de uma solução BI, tudo isso de forma objetiva e prática, com linguagem acessível tanto para técnicos quanto gestores e analista de negócio. Acesse www.bicomodeveser.com.br

Site de **cupons** do prof. Grimaldo, com desconto de todos os seus cursos de dados da Udemy, atualizado diariamente com diversas promoções, incluindo cursos gratuitos. Acesse https://is.gd/CUPOMCURSOSPROFGRIMALDO

Curso DATABRICKS DELTA LAKE
Prof. Grimaldo Oliveira
grimaldo_lopes@hotmail.com

Idealizador do Blog **BI COM VATAPÁ** reúne informações diversas sobre a área de dados com detalhes sobre o mundo de Business Intelligence, Big Data, Ciência de dados, Mineração de dados e muitos outros. Acesse http://bicomvatapa.blogspot.com/

Aula – comandos

Esta apostila contém os comandos utilizados nos NOTEBOOKS para interação com o DATABRICKS DELTA LAKE

- **Notebook - Script de carga das tabelas – Script 01**

```
%md
#######Primeiro Script , trabalhando com carga de dados e ajuste de
registros no Delta Lake
##Carrega os dados do arquivo Json

%python
dfcliente =
spark.read.json("/FileStore/tables/cliente/clientes.json");
dfcliente.printSchema()
dfcliente.show()

%md
##Cria a tabela temporária com os dos do arquivo json em memória

%python
dfcliente.createOrReplaceTempView("compras_view");
saida =spark.sql("SELECT * FROM compras_view")
saida.show()

%md
##Carrega os dados no Delta Lake gerando uma tabela chamada compras,
note USING DELTA

%scala
val scrisql = "CREATE OR REPLACE TABLE compras (id STRING, date_order
STRING,customer STRING,product STRING,unit INTEGER,price DOUBLE) USING
DELTA PARTITIONED BY (date_order) ";
spark.sql(scrisql);

%md
##Lista os dados do Delta Lake, que estará vazia

%scala
spark.sql("select * from compras").show()

%md
##Criando um merge para carregar os dados da tabela temporário no
Delta Lake

%scala
val  mergedados = "Merge into compras " +
        "using compras_view as cmp_view " +
        "ON compras.id = cmp_view.id " +
        "WHEN MATCHED THEN " +
        "UPDATE SET compras.product = cmp_view.product," +
        "compras.price = cmp_view.price " +
        "WHEN NOT MATCHED THEN INSERT * ";

spark.sql(mergedados);
```

```
%md
## Exibe os dados que foram carregados com o merge

%scala
spark.sql("select * from compras").show();

%md
##Atualiza os dados do id=4 com o comando update

%scala
val atualiza_dados = "update compras " +
        "set product = 'Geladeira' " +
        "where id =4";
spark.sql(atualiza_dados);

%md
## Exibe os dados que foram carregados, note a atualização no id=4

%scala
spark.sql("select * from compras").show();

%md
## Eliminação do registro cujo o id=4

%scala
val deletaregistro = "delete from compras where id = 1";
spark.sql(deletaregistro);

%md
## Exibe os dados que foram carregados

%scala
spark.sql("select * from compras").show();
```

- **Notebook – Comandos utilitários Delta Lake – Script 02**

```
%md
### Mostrando o histórico das transações no Delta Lake - tabela
Compras

%sql
DESCRIBE HISTORY '/user/hive/warehouse/compras'

%md
### Mostrando os dados de criação da tabela compras

%sql
DESCRIBE DETAIL '/user/hive/warehouse/compras'

%md
## Retornando(exibindo) a versão dos dados a posição do merge dos
dados

%sql
 SELECT * FROM delta. `/user/hive/warehouse/compras` VERSION AS OF 2

%md
## Mostrando os dados após o retorno das versões

%scala
spark.sql("select * from compras").show();

%md
## Clonando os dados gerados

%sql
CREATE TABLE delta. `/temporario/hive/clonado` CLONE delta.
`/user/hive/warehouse/compras`

%md
## Restaurando em definitivo os dados na tabela compras

RESTORE TABLE compras TO VERSION AS OF 2

%md
## Mostrando os dados restaurados

%scala
spark.sql("select * from compras").show();
```

OBS.: Caso o seu cluster não esteja mais respondendo é necessário recriá-lo, entretanto os arquivos Delta Lake continuam na base, caso seja necessário, você precisará apagar os arquivos parquet caso crie um novo cluster, pois as tabelas serão perdidas.

Lembrando que estamos utilizando a versão Community que possui apenas 2 horas de cluster ativo.

O comando para eliminar as pastas será:

%fs rm –r < pasta a ser eliminada>
Ex.: %fs rm –r /user/hive/warehouse/compras

Notebook – Otimizando consultas Delta Lake – Script 03

```
%sql
DROP TABLE IF EXISTS hotel;

-- Carregaato de arquivos Parquet sobre dados de hotel
CREATE TABLE hotel
USING parquet
PARTITIONED BY (Categoria)
SELECT _c0 as Ordem, _c1 as Tipo, _c2 as Situacao, _c3 as Tip_local,
_c4 as Categoria, _c5 as Tip_Local2, _c6 as Tip_local3, _c7 as
Acomodacao, _c8 as Cidade, _c9 as Pais, _c10 as Endereco, _c11 as
Latitude, _c12 as Longitude, _c13 as Provincia, _c14 as CEP, _c15 as
UF, _c16 as Data_estadia, _c17 as Revisao_texto, _c17 as
Revisao_titulo, _c19 as Revisao_cidade, _c20 as Endereco_web, _c21 as
Comentario_usuario, _c22 as Resumo, _c23 as Comentario2, _c24 as
Comentario3, _c25 as Comentario4
FROM
csv.`dbfs:/FileStore/tables/hotel/Datafiniti_Hotel_Reviews_Jun19.csv`

%sql
-- Fazendo uma contagem das principais categorias e comentarios
SELECT Categoria,Comentario2,Comentario3,count(*) as Total
FROM hotel
GROUP BY Categoria,Comentario2,Comentario3
ORDER BY Categoria,Total DESC;

%sql
-- Criando a tabela Delta
DROP TABLE IF EXISTS hotel;

CREATE TABLE hotel
USING delta
PARTITIONED BY (categoria)
SELECT _c0 as Ordem, _c1 as Tipo, _c2 as Situacao, _c3 as Tip_local,
_c4 as Categoria, _c5 as Tip_Local2, _c6 as Tip_local3, _c7 as
Acomodacao, _c8 as Cidade, _c9 as Pais, _c10 as Endereco, _c11 as
Latitude, _c12 as Longitude, _c13 as Provincia, _c14 as CEP, _c15 as
UF, _c16 as Data_estadia, _c17 as Revisao_texto, _c17 as
Revisao_titulo, _c19 as Revisao_cidade, _c20 as Endereco_web, _c21 as
Comentario_usuario, _c22 as Resumo, _c23 as Comentario2, _c24 as
Comentario3, _c25 as Comentario4
FROM
csv.`dbfs:/FileStore/tables/hotel/Datafiniti_Hotel_Reviews_Jun19.csv`

%sql
-- Otimizando a consulta da tabela Delta com o campo Pais, é
importante que busque o campo que melhor otimiza
OPTIMIZE hotel ZORDER BY (Pais);

-- Executando a consulta otimizada na tabela Delta
SELECT Categoria,Comentario2,Comentario3,count(*) as Total
FROM hotel
```

```
GROUP BY Categoria,Comentario2,Comentario3
ORDER BY Categoria,Total DESC;

COMPARE OS TEMPOS DA TABELA EM PARQUET DA TABELA DELTA
```

Antes de executar o Script4, vamos fazer um ajuste no Script 01, acrescentando como primeira linha do notebook a eliminação dos arquivos parquet para criação da tabela de compras

O comando para eliminar as pastas será:

%md
Removendo o parquet e arquivos delta, pois o cluster expirou e vamos recriar a tabela de compras

%fs rm -r /user/hive/warehouse/compras

Notebook – Time travel - Comandos– Script 04

```
%md
## Lendo a primeira versão via Python

%python
df = spark.read \
  .format("delta") \
  .option("versionAsOf", "1") \
  .load("/user/hive/warehouse/compras")
df.show()


%md
## Contando a quantidade de registros na terceira versão via SQL

%sql
SELECT count(*) FROM compras VERSION AS OF 3

%md
## Contando a quantidade de registros na terceira versão via SQL -
Outra forma de realizar a tarefa

%sql
SELECT count(*) FROM compras@v3

%md
## Contando a quantidade de registros na terceira versão via SQL -
Outra forma de realizar a tarefa

%sql
SELECT * FROM delta.`/user/hive/warehouse/compras@v3`

%md
```

```
## Descrevendo o histórico dos dados para verificar a quantidade de
versões

%sql
DESCRIBE HISTORY '/user/hive/warehouse/compras'

%md
## Vamos reinserir o registro com ID=1 que eliminamos, uma forma de
realizar o Delta Time Travel

%sql

INSERT INTO compras
SELECT * FROM compras VERSION AS OF 1
WHERE Id = 1

%md
## Exibindo os dados atualizados, após retorno da versão 1, ou seja o
registro deletado foi restaurado

%sql
SELECT * FROM compras

%md
## Mostrando as versões agora, depois do insert

%sql
DESCRIBE HISTORY '/user/hive/warehouse/compras'

%md
## Verificando quantos registro é a diferença da versão atual, para a
versão 3

%sql
SELECT count(distinct ID) - (SELECT count(distinct ID) FROM compras
VERSION AS OF 3) as `Diferença de registros`
FROM compras
```

Notebook – Projeto Livraria - Comandos– Script 05

```
%md
## Criação das tabelas TB_AUTOR,TB_LIVRO e a associativa
TB_LIVRO_AUTOR

%scala
val Tscrisql = "CREATE OR REPLACE TABLE TB_AUTOR (ID_AUTOR DOUBLE,NOME
STRING,SEXO STRING,DATA_NASCIMENTO STRING) USING DELTA ;"
spark.sql(Tscrisql);
val Tscrisql1 =  "CREATE OR REPLACE TABLE TB_LIVRO (ID_LIVRO
DOUBLE,ISBN STRING,TITULO STRING,EDICAO DOUBLE,PRECO
DOUBLE,QTDE_ESTOQUE DOUBLE) USING DELTA;"
spark.sql(Tscrisql1);
val Tscrisql3 = "CREATE OR REPLACE TABLE TB_LIVRO_AUTOR
(ID_LIVRO_AUTOR DOUBLE,ID_LIVRO DOUBLE,ID_AUTOR DOUBLE) USING DELTA;"
spark.sql(Tscrisql3);

%md
## Inserção de registros na tabela TB_AUTOR
```

```
%scala
val scrisql = "Insert into TB_AUTOR
(ID_AUTOR,NOME,SEXO,DATA_NASCIMENTO) values (1,'Joao','M',
'01/01/1970');"
spark.sql(scrisql);
val scrisql1 = "Insert into TB_AUTOR
(ID_AUTOR,NOME,SEXO,DATA_NASCIMENTO) values (2,'Maria','F',
'25/11/1975');"
spark.sql(scrisql1);
val scrisql2 = "Insert into TB_AUTOR
(ID_AUTOR,NOME,SEXO,DATA_NASCIMENTO) values (3,'Sandra','F',
'14/11/1978');"
spark.sql(scrisql2);
val scrisql3 = "Insert into TB_AUTOR
(ID_AUTOR,NOME,SEXO,DATA_NASCIMENTO) values (4,'Tereza','F',
'21/11/1978');"
spark.sql(scrisql3);

%md
## Inserção de registros na tabela TB_LIVRO

%scala
val Lscrisql = "Insert into TB_LIVRO
(ID_LIVRO,ISBN,TITULO,EDICAO,PRECO,QTDE_ESTOQUE) values
(1,'1234567890','Banco de Dados',2,10,407);"
spark.sql(Lscrisql);
val Lscrisql1 = "Insert into TB_LIVRO
(ID_LIVRO,ISBN,TITULO,EDICAO,PRECO,QTDE_ESTOQUE) values
(2,'2345678901','Redes de Computadores',1,10,60);"
spark.sql(Lscrisql1);
val Lscrisql2 = "Insert into TB_LIVRO
(ID_LIVRO,ISBN,TITULO,EDICAO,PRECO,QTDE_ESTOQUE) values
(3,'3456789012','Interface Homem-Maquina',3,10,10);"
spark.sql(Lscrisql2);

%md
## Inserção de registros na tabela TB_LIVRO_AUTOR

%scala
val LAscrisql = "Insert into TB_LIVRO_AUTOR
(ID_LIVRO_AUTOR,ID_LIVRO,ID_AUTOR) values (1,1,1);"
spark.sql(LAscrisql);
val LAscrisql1 = "Insert into TB_LIVRO_AUTOR
(ID_LIVRO_AUTOR,ID_LIVRO,ID_AUTOR) values (2,1,2);"
spark.sql(LAscrisql1);
val LAscrisql2 = "Insert into TB_LIVRO_AUTOR
(ID_LIVRO_AUTOR,ID_LIVRO,ID_AUTOR) values (3,2,3);"
spark.sql(LAscrisql2);
val LAscrisql3 = "Insert into TB_LIVRO_AUTOR
(ID_LIVRO_AUTOR,ID_LIVRO,ID_AUTOR) values (4,3,2);"
spark.sql(LAscrisql3);
val LAscrisql4 = "Insert into TB_LIVRO_AUTOR
(ID_LIVRO_AUTOR,ID_LIVRO,ID_AUTOR) values (5,3,3);"
spark.sql(LAscrisql4);

%md
## Selecionando os registros, ligando todas as tabelas
```

```
%sql
SELECT TB_LIVRO.TITULO,
TB_LIVRO.ISBN,TB_LIVRO.PRECO,TB_AUTOR.NOME,TB_AUTOR.DATA_NASCIMENTO
FROM TB_LIVRO
  INNER JOIN TB_LIVRO_AUTOR ON TB_LIVRO.ID_LIVRO =
TB_LIVRO_AUTOR.ID_LIVRO
  INNER JOIN TB_AUTOR ON TB_AUTOR.ID_AUTOR = TB_LIVRO_AUTOR.ID_AUTOR


%md
## Criando uma check constraints para a tabela TB_AUTOR, somente
permitindo a inserção de 4 registros

%sql
ALTER TABLE TB_AUTOR DROP CONSTRAINT validIds;
ALTER TABLE TB_AUTOR ADD CONSTRAINT validIds CHECK (ID_AUTOR> 0 and
ID_AUTOR < 5);
SHOW TBLPROPERTIES TB_AUTOR;

%md
## Verificando se a check constraints irá funcionar, executando um
insert

%scala
val scrisql = "Insert into TB_AUTOR
(ID_AUTOR,NOME,SEXO,DATA_NASCIMENTO) values (5,'Joao','M',
'01/01/1970');"
spark.sql(scrisql);
```

Notebook – Streaming Delta Lake - Comandos– Script 06

```
%md
## Eliminando arquivos caso existam nas pastas, pois serão destinados
as tabelas Delta

%fs rm -r /tmp/delta/events
%fs rm -r /tmp/delta/checkpoint

%md
## Listando os diversos arquivos Json para carga no Delta Lake

%fs ls /databricks-datasets/structured-streaming/events/

%md
## Exibindo o conteúdo de 1 arquivo Json

%python
#Lendo um dos arquivos JSON
dataf3 = spark.read.json("/databricks-datasets/structured-
streaming/events/file-1.json")
dataf3.show()

%md
## Criando um banco de dados em separado e uma tabela Delta que irá
receber os dados do Json em Streaming
```

```
%sql
CREATE DATABASE IF NOT EXISTS db_stream;
USE db_stream;
DROP TABLE IF EXISTS db_stream.tab_stream;
CREATE TABLE db_stream.tab_stream(
  action STRING,
  time STRING
  )
USING delta
LOCATION "/tmp/delta/events"

%md
## Executando a carga na pasta do Delta Lake, onde serão armazenados
os dados

%python
from pyspark.sql.functions import *
from pyspark.sql.types import *

# Streaming reads and append into delta table (Start !)
read_schema = StructType([
  StructField("action", StringType(), False),
  StructField("time", StringType(), True)
  ])
df2 = (spark.readStream
  .option("maxFilesPerTrigger", "1")
  .schema(read_schema)
  .json("/databricks-datasets/structured-streaming/events/"))
(df2.writeStream
  .format("delta")
  .outputMode("append")
  .option("checkpointLocation", "/tmp/delta/checkpoint")
  .option("path", "/tmp/delta/events").start())

%md
## Exibindo os dados em tempo real oriunda da tabela Delta

%sql select distinct action, count(*) from db_stream.tab_stream
 group by action

%md
## Listando os históricos registrados na tabela Delta

%sql
DESCRIBE HISTORY '/tmp/delta/events'
```

Notebook – Script Schema Evolution - Comandos– Script 07

```
%md
## Criando o primeiro dataframe com dados, com os campos
(Funcionario,Salario)

%python
empresa1 = spark.createDataFrame(
    [
        ("Joao Santos",2000),
        ("Carlos Fernandez",3400)
    ],['Funcionario', 'Salario'] )
display(empresa1)
empresa1.printSchema()

%md
## Setado caminho, armazenando em um atributo

%python
parquetpath = "dbfs:/FileStore/tables/delta/schema_evolution/parquet"

%md
## Criando arquivos parquet com base no primeiro dataframe

%python
(
empresa1
  .write
  .format("parquet")
  .save("/FileStore/tables/delta/schema_evolution/parquet")
)
spark.read.parquet(parquetpath).show()

%md
## Criando o segundo dataframe com dados, acrescentando novos campos
(Setor,Comissao)

%python
empresa2 = spark.createDataFrame(
    [
        ("Financeiro",240),
        ("Marketing",540)
    ],['Setor', 'Comissao'] )
display(empresa2)
empresa2.printSchema()

%md
## Apesar de colocar "append"no parquet, não houve evolução do
esquema, as colunas foram substituídas

%python
empresa2.write.mode("append").parquet(parquetpath)
spark.read.parquet(parquetpath).show()

%md
##Vamos gerar o Schema Evolution com as tabelas Delta
```

```
%python
deltapath = "/FileStore/tables/delta/schema_evolution/delta"
(
empresa1
  .write
  .format("delta")
  .save("/FileStore/tables/delta/schema_evolution/delta")
)
spark.read.format("delta").load(deltapath).show()

%md
##Vamos realizar um "merge" entre os dataframes, note que agora
conseguirá realizar a junção entre os schemas. Os dados inexistentes
foram acrescidos de nulos

%python
(
empresa2
  .write
  .format("delta")
  .mode("append")
  .option("mergeSchema", "true")
  .save(deltapath)
)
spark.read.format("delta").load(deltapath).show()

%md
##Vamos inserir novos dados e aplicar o "append" para verificar como
funcionará a inserção

%python
empresa3 = spark.createDataFrame(
    [
        ("Sandra Lemos",672),
        ("Carla Soares",966),
    ],
    ['Funcionario', 'Comissao']
)

%md
## Vamos acrescentar dados mais dados e verificar a inclusão apenas de
alguns campos

%python
(
empresa3
  .write
  .format("delta")
  .mode("append")
  .option("mergeSchema", "true")
  .save(deltapath)
)
spark.read.format("delta").load(deltapath).show()

%md
##Vamos fazer sobregravação dos formatos delta, variando os modos ,
quando o merge está ativo "option=mergeSchema, mode=overwrite"
```

```
%python
(
empresa3
  .write
  .format("delta")
  .mode("overwrite")
  .option("mergeSchema", "true")
  .save(deltapath)
)
spark.read.format("delta").load(deltapath).show()

%md
## Vamos sobrescrever toda a tabela, perceba as mudancas dos campos
que ficaram e dos registros. "option=overwriteschema , mode=overwrite"

%python
(
empresa3
  .write
  .format("delta")
  .option("overwriteSchema", "true")
  .mode("overwrite")
  .save(deltapath)
)
spark.read.format("delta").load(deltapath).show()

%md
## Vamos criar uma tabela delta com referência aos parquet (delta)
criados

%sql
CREATE TABLE tab_empresa(
  Funcionario STRING,
  Comissao long
  )
USING delta
LOCATION "/FileStore/tables/delta/schema_evolution/delta"

%md
## Vamos listar o histórico gerado de todas as nossas mudanças

%sql
DESCRIBE HISTORY '/FileStore/tables/delta/schema_evolution/delta'

%md
## Listando todas as versões que podemos utilizar

%sql
 SELECT * FROM delta. `/FileStore/tables/delta/schema_evolution/delta`
VERSION AS OF 4
```