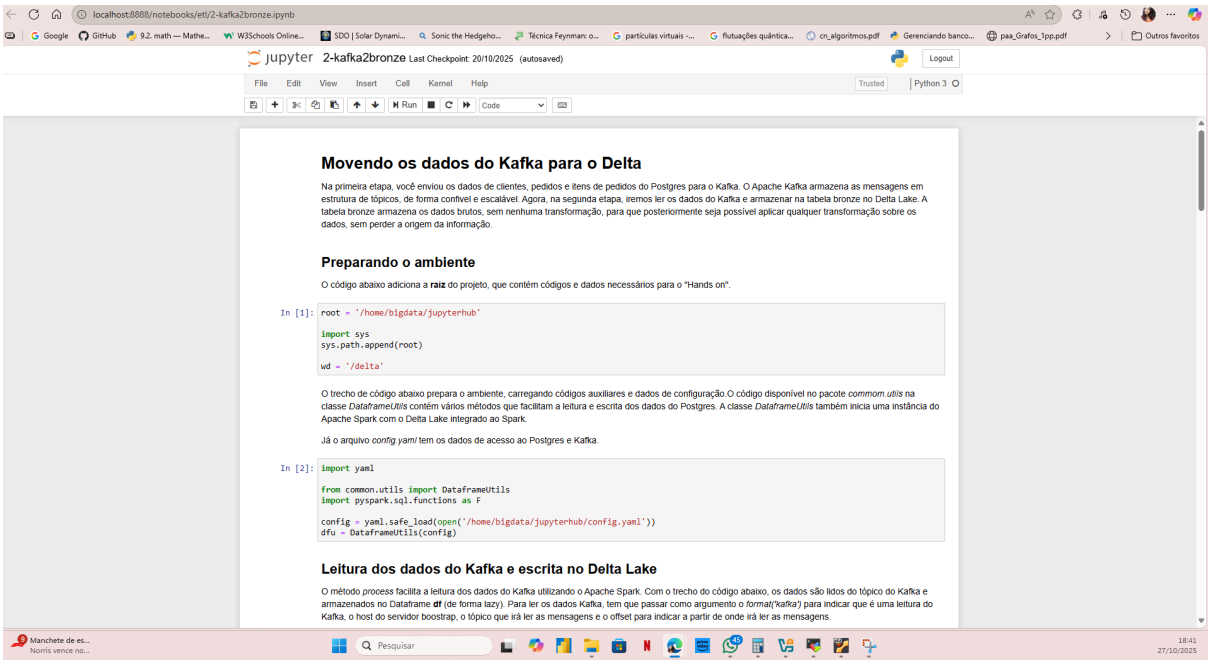
Notebook 2-kafka2bronze



Movendo os dados do Kafka para o Delta

Na primeira etapa, você enviou os dados de clientes, pedidos e itens de pedidos do Postgres para o Kafka. O Apache Kafka armazena as mensagens em estrutura de tópicos, de forma confiwel e escalável. Agora, na segunda etapa, iremos ler os dados do Kafka e armazenar na tabela bronze no Delta Lake. A tabela bronze armazena os dados brutos, sem nenhuma transformação, para que posteriormente seja possível aplicar qualquer transformação sobre os dados, sem perder a origem da informação.

Preparando o ambiente

O código abaixo adiciona a **raiz** do projeto, que contém códigos e dados necessários para o "Hands on".

In [1]:
```
root = '/home/bigdata/jupyterhub'

import sys
sys.path.append(root)

wd = '/delta'
```

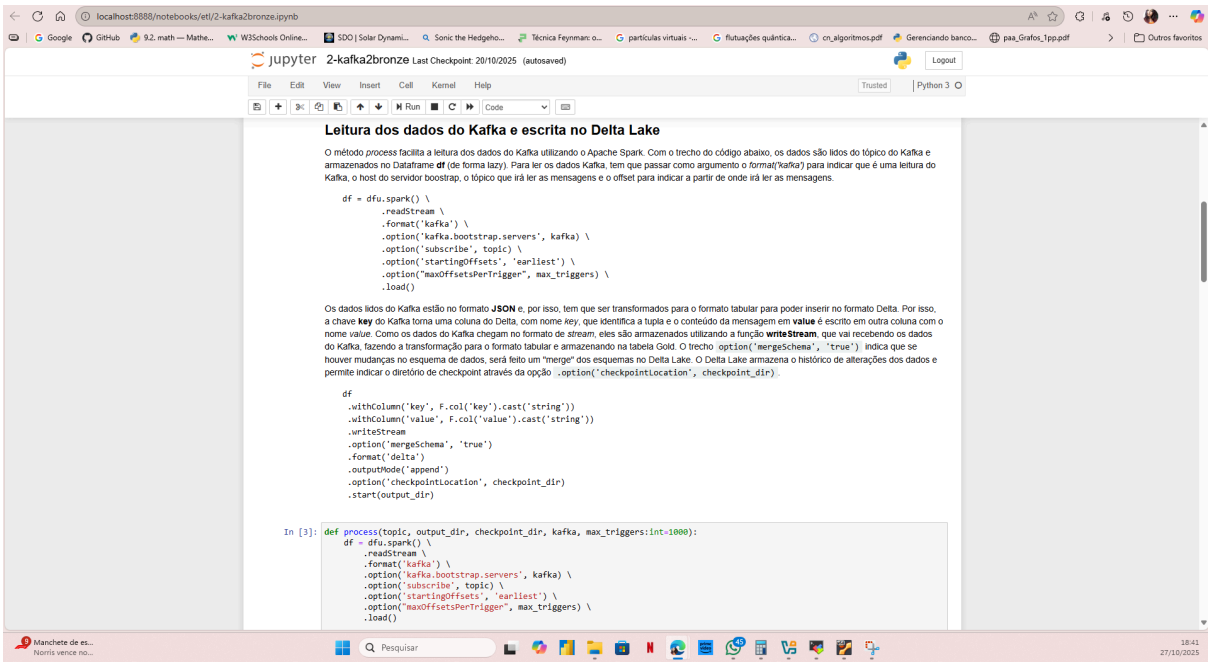
O trecho de código abaixo prepara o ambiente, carregando códigos auxiliares e dados de configuração.O código disponível no pacote *commom.utils* na classe *DataframeUtils* contém vários métodos que facilitam a leitura e escrita dos dados do Postgres. A classe *DataframeUtils* também inicia uma instância do Apache Spark com o Delta Lake integrado ao Spark.

Já o arquivo *config.yaml* tem os dados de acesso ao Postgres e Kafka.

In [2]:
```
import yaml

from common.utils import DataframeUtils
import pyspark.sql.functions as F

config = yaml.safe_load(open('/home/bigdata/jupyterhub/config.yaml'))
dfu = DataframeUtils(config)
```

Leitura dos dados do Kafka e escrita no Delta Lake

O método *process* facilita a leitura dos dados do Kafka utilizando o Apache Spark. Com o trecho do código abaixo, os dados são lidos do tópico do Kafka e armazenados no Dataframe **df** (de forma lazy). Para ler os dados Kafka, tem que passar como argumento o *format('kafka')* para indicar que é uma leitura do Kafka, o host do servidor boostrap, o tópico que irá ler as mensagens e o offset para indicar a partir de onde irá ler as mensagens.



Leitura dos dados do Kafka e escrita no Delta Lake

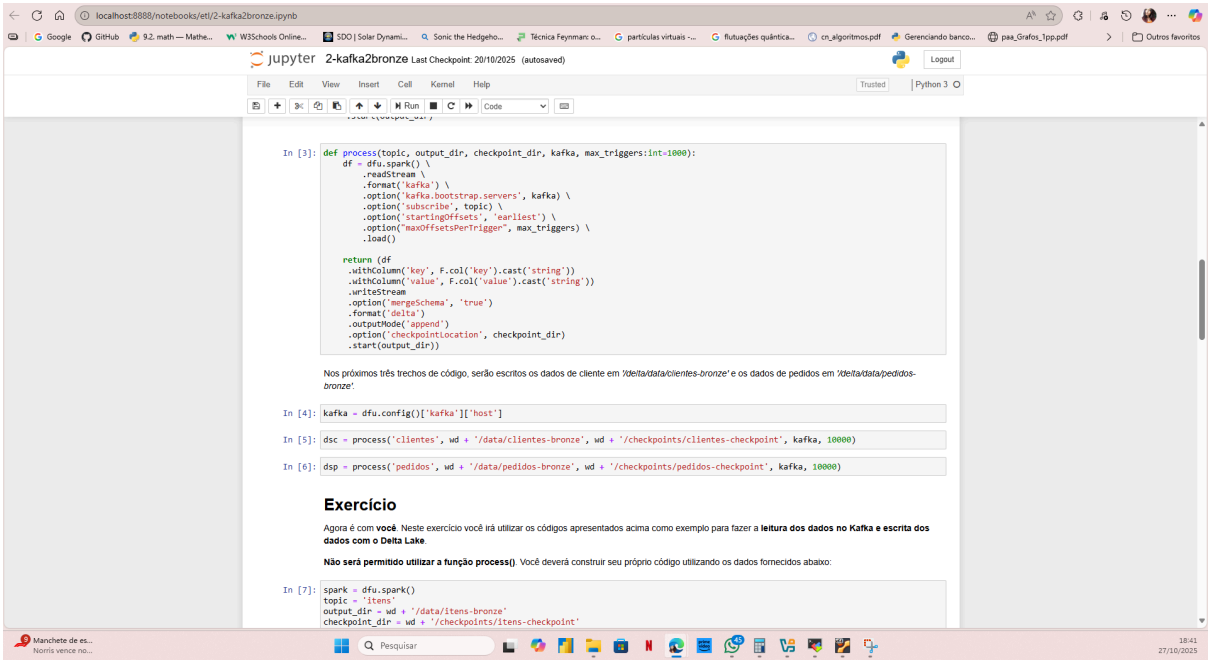
O método *process* facilita a leitura dos dados do Kafka utilizando o Apache Spark. Com o trecho do código abaixo, os dados são lidos do tópico do Kafka e armazenados no Dataframe **df** (de forma lazy). Para ler os dados Kafka, tem que passar como argumento o *format('kafka')* para indicar que é uma leitura do Kafka, o host do servidor boostrap, o tópico que irá ler as mensagens e o offset para indicar a partir de onde irá ler as mensagens.

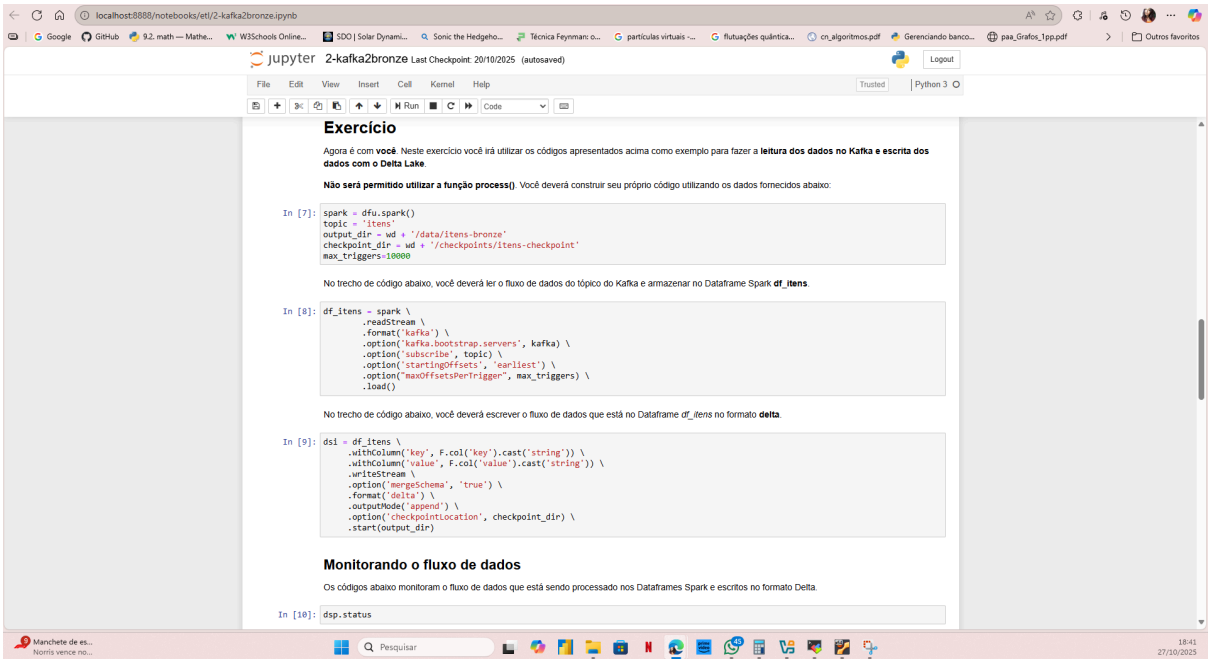
```
df = dfu.spark() \
        .readStream \
        .format('kafka') \
        .option('kafka.bootstrap.servers', kafka) \
        .option('subscribe', topic) \
        .option('startingOffsets', 'earliest') \
        .option("maxOffsetsPerTrigger", max_triggers) \
        .load()
```

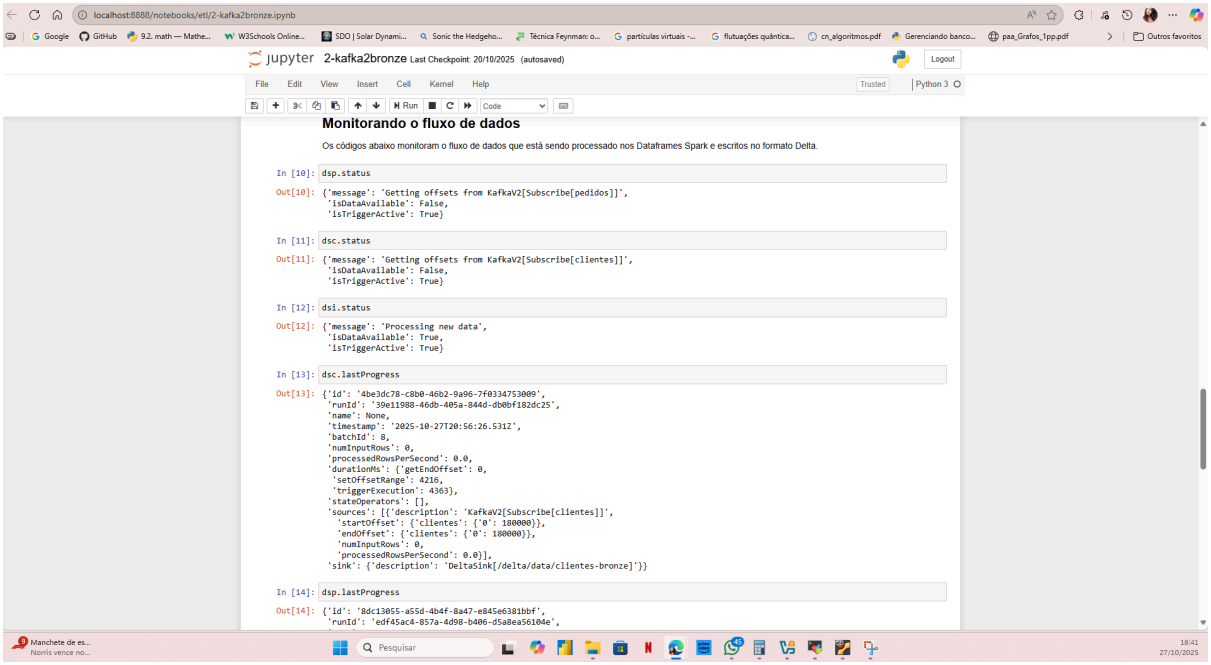
Os dados lidos do Kafka estão no formato **JSON** e, por isso, tem que ser transformados para o formato tabular para poder inserir no formato Delta. Por isso, a chave **key** do Kafka torna uma coluna do Delta, com nome *key*, que identifica a tupla e o conteúdo da mensagem em **value** é escrito em outra coluna com o nome *value*. Como os dados do Kafka chegam no formato de *stream*, eles são armazenados utilizando a função **writeStream**, que vai recebendo os dados do Kafka, fazendo a transformação para o formato tabular e armazenando na tabela Gold. O trecho `option('mergeSchema', 'true')` indica que se houver mudanças no esquema de dados, será feito um "merge" dos esquemas no Delta Lake. O Delta Lake armazena o histórico de alterações dos dados e permite indicar o diretório de checkpoint através da opção `.option('checkpointLocation', checkpoint_dir)`.

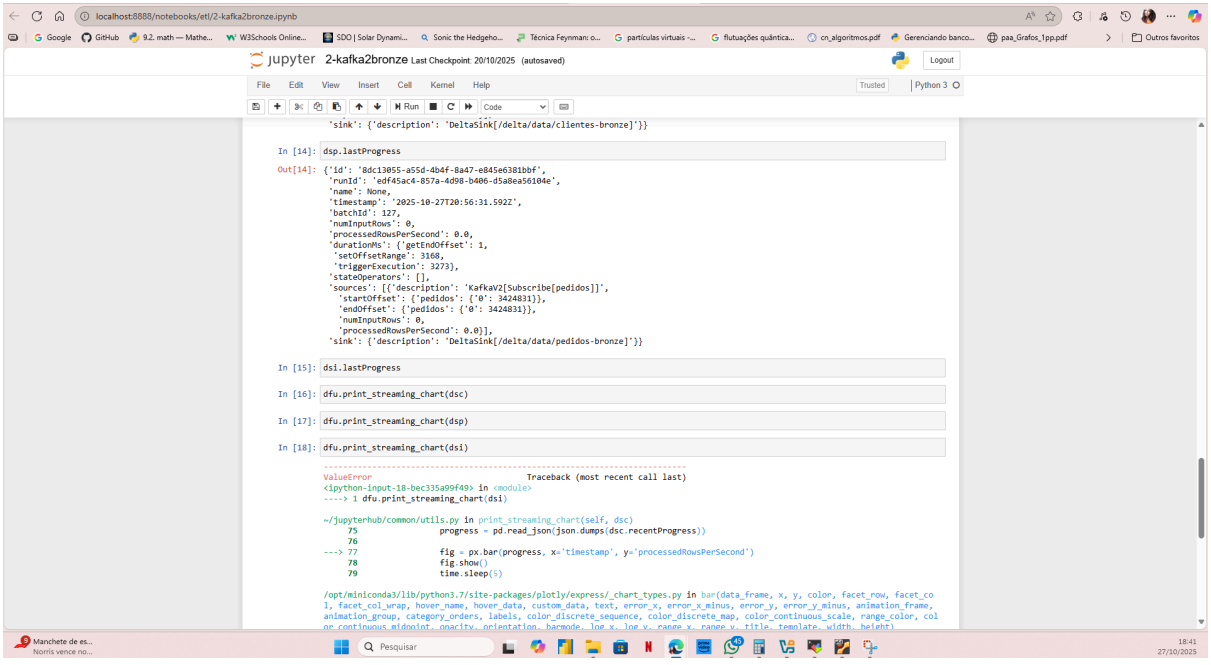
```
df
 .withColumn('key', F.col('key').cast('string'))
 .withColumn('value', F.col('value').cast('string'))
 .writeStream
 .option('mergeSchema', 'true')
 .format('delta')
 .outputMode('append')
 .option('checkpointLocation', checkpoint_dir)
 .start(output_dir)
```

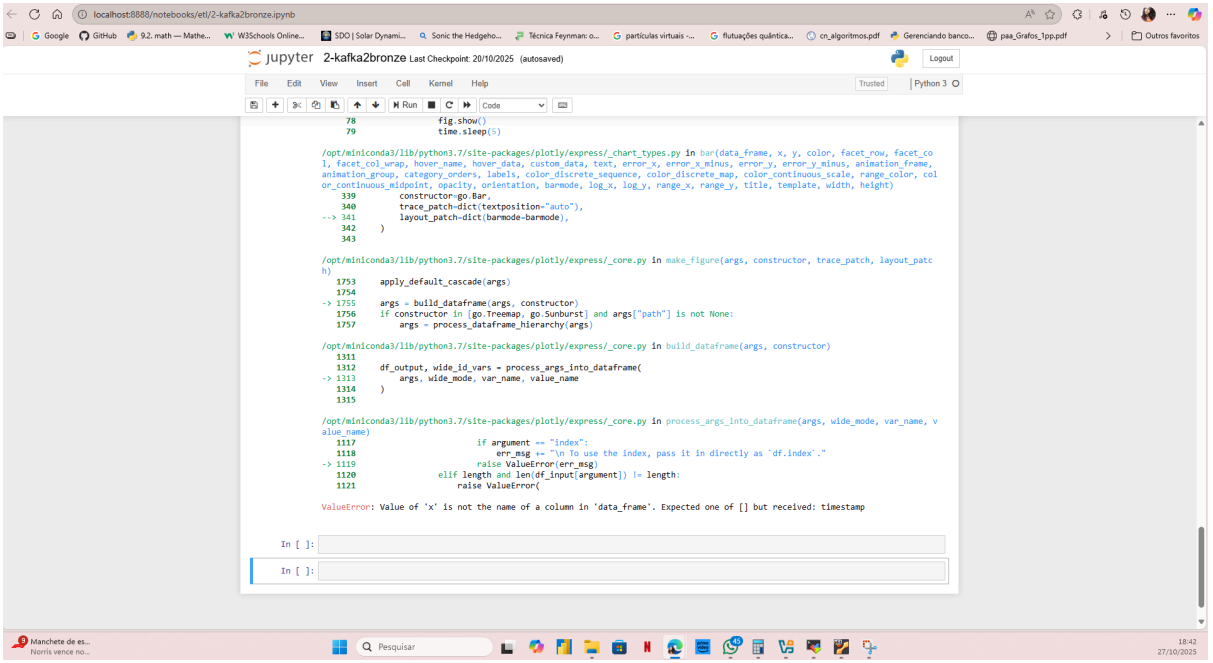
In [3]:
```
def process(topic, output_dir, checkpoint_dir, kafka, max_triggers:int=1000):
    df = dfu.spark() \
        .readStream \
        .format('kafka') \
        .option('kafka.bootstrap.servers', kafka) \
        .option('subscribe', topic) \
        .option('startingOffsets', 'earliest') \
        .option("maxOffsetsPerTrigger", max_triggers) \
        .load()
```


localhost:8888/notebooks/etl/2-kafka2bronze.ipynb
jupyter 2-kafka2bronze Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
In [3]:
def process(topic, output_dir, checkpoint_dir, kafka, max_triggers:int=1000):
df = dfu.spark() \
.readStream \
.format('kafka') \
.option('kafka.bootstrap.servers', kafka) \
.option('subscribe', topic) \
.option('startingOffsets', 'earliest') \
.option("maxOffsetsPerTrigger", max_triggers) \
.load()
return (df
.withColumn('key', F.col('key').cast('string'))
.withColumn('value', F.col('value').cast('string'))
.writeStream
.option('mergeSchema', 'true')
.format('delta')
.outputMode('append')
.option('checkpointLocation', checkpoint_dir)
.start(output_dir))
Nos próximos três trechos de código, serão escritos os dados de cliente em '/delta/data/clientes-bronze' e os dados de pedidos em '/delta/data/pedidos-bronze'.
In [4]:
kafka = dfu.config()['kafka']['host']
In [5]:
dsc = process('clientes', wd + '/data/clientes-bronze', wd + '/checkpoints/clientes-checkpoint', kafka, 10000)
In [6]:
dsp = process('pedidos', wd + '/data/pedidos-bronze', wd + '/checkpoints/pedidos-checkpoint', kafka, 10000)
Exercício
Agora é com você. Neste exercício você irá utilizar os códigos apresentados acima como exemplo para fazer a leitura dos dados no Kafka e escrita dos dados com o Delta Lake.
Não será permitido utilizar a função process(). Você deverá construir seu próprio código utilizando os dados fornecidos abaixo:
In [7]:
spark = dfu.spark()
topic = 'itens'
output_dir = wd + '/data/itens-bronze'
checkpoint_dir = wd + '/checkpoints/itens-checkpoint'
Pesquisar
18:41
27/10/2025

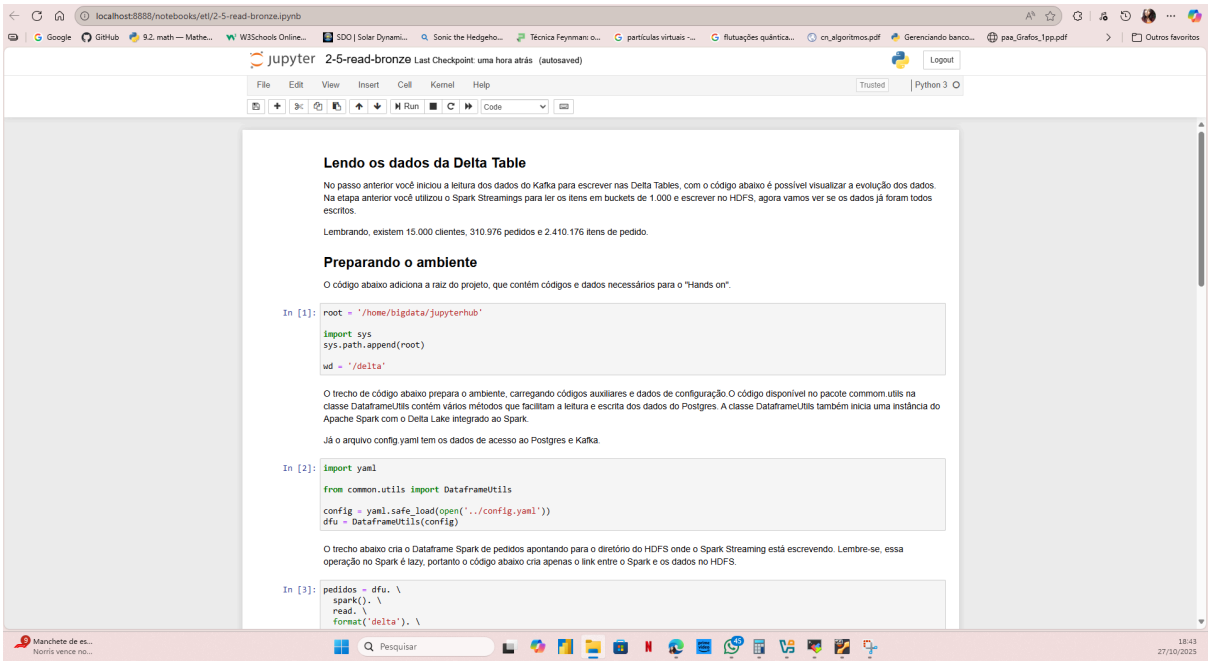

localhost:8888/notebooks/etl/2-kafka2bronze.ipynb
jupyter 2-kafka2bronze Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
Exercício
Agora é com você. Neste exercício você irá utilizar os códigos apresentados acima como exemplo para fazer a leitura dos dados no Kafka e escrita dos dados com o Delta Lake.
Não será permitido utilizar a função process(). Você deverá construir seu próprio código utilizando os dados fornecidos abaixo:
In [7]:
spark = dfu.spark()
topic = 'itens'
output_dir = wd + '/data/itens-bronze'
checkpoint_dir = wd + '/checkpoints/itens-checkpoint'
max_triggers=10000
No trecho de código abaixo, você deverá ler o fluxo de dados do tópico do Kafka e armazenar no Dataframe Spark df_itens.
In [8]:
df_itens = spark \
.readStream \
.format('kafka') \
.option('kafka.bootstrap.servers', kafka) \
.option('subscribe', topic) \
.option('startingOffsets', 'earliest') \
.option("maxOffsetsPerTrigger", max_triggers) \
.load()
No trecho de código abaixo, você deverá escrever o fluxo de dados que está no Dataframe df_itens no formato delta.
In [9]:
dsi = df_itens \
.withColumn('key', F.col('key').cast('string')) \
.withColumn('value', F.col('value').cast('string')) \
.writeStream \
.option('mergeSchema', 'true') \
.format('delta') \
.outputMode('append') \
.option('checkpointLocation', checkpoint_dir) \
.start(output_dir)
Monitorando o fluxo de dados
Os códigos abaixo monitoram o fluxo de dados que está sendo processado nos Dataframes Spark e escritos no formato Delta.
In [10]:
dsp.status
Pesquisar
18:41
27/10/2025

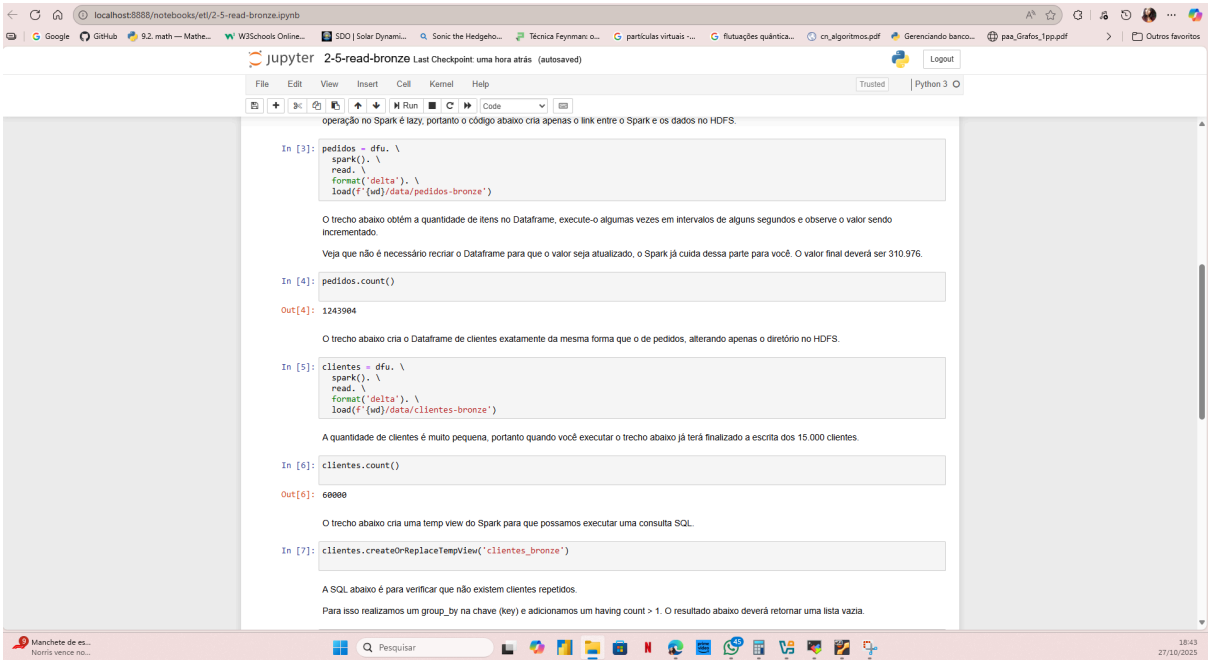

localhost:8888/notebooks/etl/2-kafka2bronze.ipynb
jupyter 2-kafka2bronze Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
Monitorando o fluxo de dados
Os códigos abaixo monitoram o fluxo de dados que está sendo processado nos Dataframes Spark e escritos no formato Delta.
In [10]: dsp.status
Out[10]: {'message': 'Getting offsets from KafkaV2[Subscribe[pedidos]]',
'isDataAvailable': False,
'isTriggerActive': True}
In [11]: dsc.status
Out[11]: {'message': 'Getting offsets from KafkaV2[Subscribe[clientes]]',
'isDataAvailable': False,
'isTriggerActive': True}
In [12]: dsi.status
Out[12]: {'message': 'Processing new data',
'isDataAvailable': True,
'isTriggerActive': True}
In [13]: dsc.lastProgress
Out[13]: {'id': '4be3dc78-c8b0-46b2-9a96-7f0334753009',
'runId': '39e11988-46db-405a-844d-db0bf182dc25',
'name': None,
'timestamp': '2025-10-27T20:56:26.531Z',
'batchId': 8,
'numInputRows': 0,
'processedRowsPerSecond': 0.0,
'durationMs': {'getEndOffset': 0,
'setOffsetRange': 4216,
'triggerExecution': 4363},
'stateOperators': [],
'sources': [{'description': 'KafkaV2[Subscribe[clientes]]',
'startOffset': {'clientes': {'0': 180000}},
'endOffset': {'clientes': {'0': 180000}},
'numInputRows': 0,
'processedRowsPerSecond': 0.0}],
'sink': {'description': 'DeltaSink[/delta/data/clientes-bronze]'}}
In [14]: dsp.lastProgress
Out[14]: {'id': '8dc13055-a55d-4b4f-8a47-e845e6381bbf',
'runId': 'edf45ac4-857a-4d98-b406-d5a8ea56104e',
Manchete de es...
Norris vence no...
Pesquisar
18:41
27/10/2025

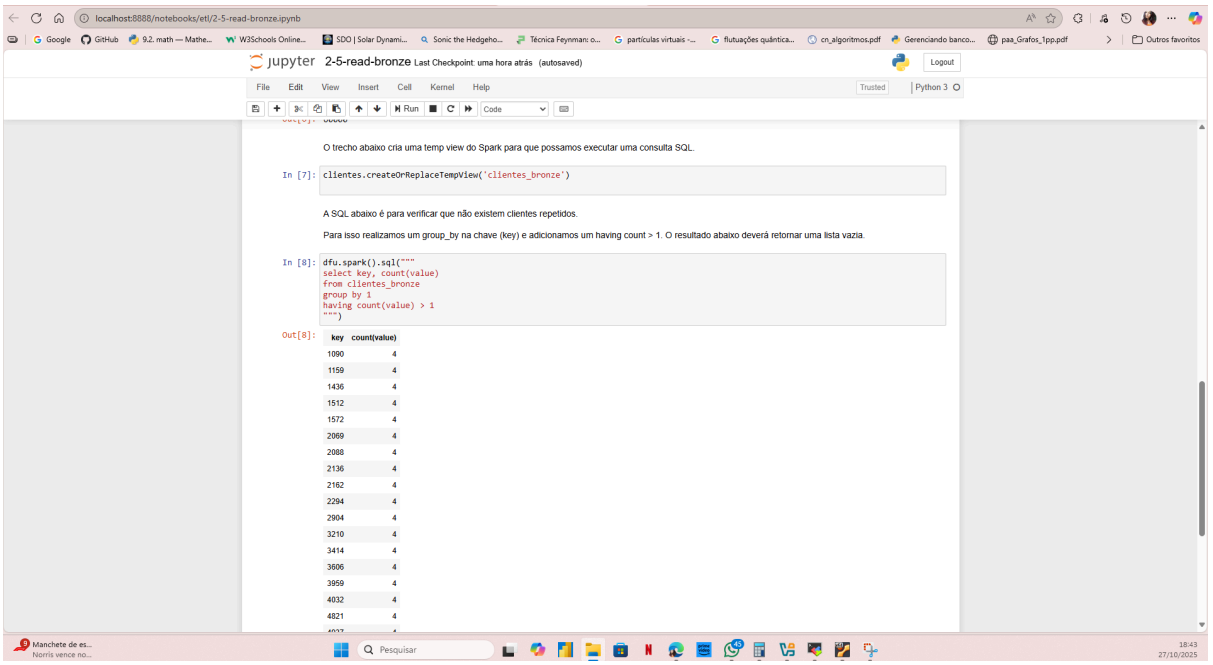

localhost:8888/notebooks/etl/2-kafka2bronze.ipynb
jupyter 2-kafka2bronze Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
'sink': {'description': 'DeltaSink[/delta/data/clientes-bronze]'}}
In [14]: dsp.lastProgress
Out[14]: {'id': '8dc13055-a55d-4b4f-8a47-e845e6381bbf',
'runId': 'edf45ac4-857a-4d98-b406-d5a8ea56104e',
'name': None,
'timestamp': '2025-10-27T20:56:31.592Z',
'batchId': 127,
'numInputRows': 0,
'processedRowsPerSecond': 0.0,
'durationMs': {'getEndOffset': 1,
'setOffsetRange': 3160,
'triggerExecution': 3273},
'stateOperators': [],
'sources': [{'description': 'KafkaV2[Subscribe[pedidos]]',
'startOffset': {'pedidos': {'0': 3424831}},
'endOffset': {'pedidos': {'0': 3424831}},
'numInputRows': 0,
'processedRowsPerSecond': 0.0}],
'sink': {'description': 'DeltaSink[/delta/data/pedidos-bronze]'}}
In [15]: dsi.lastProgress
In [16]: dfu.print_streaming_chart(dsc)
In [17]: dfu.print_streaming_chart(dsp)
In [18]: dfu.print_streaming_chart(dsi)
ValueError Traceback (most recent call last)
<ipython-input-18-bec335a99f49> in <module>
----> 1 dfu.print_streaming_chart(dsi)
~/jupyterhub/common/utils.py in print_streaming_chart(self, dsc)
75 progress = pd.read_json(json.dumps(dsc.recentProgress))
76
---> 77 fig = px.bar(progress, x='timestamp', y='processedRowsPerSecond')
78 fig.show()
79 time.sleep(5)
/opt/miniconda3/lib/python3.7/site-packages/plotly/express/_chart_types.py in bar(data_frame, x, y, color, facet_row, facet_col, facet_col_wrap, hover_name, hover_data, custom_data, text, error_x, error_x_minus, error_y, error_y_minus, animation_frame, animation_group, category_orders, labels, color_discrete_sequence, color_discrete_map, color_continuous_scale, range_color, color_continuous_midpoint, opacity, orientation, barmode, log_x, log_y, range_x, range_y, title, template, width, height)
Manchete de es...
Norris vence no...
Pesquisar
18:41
27/10/2025

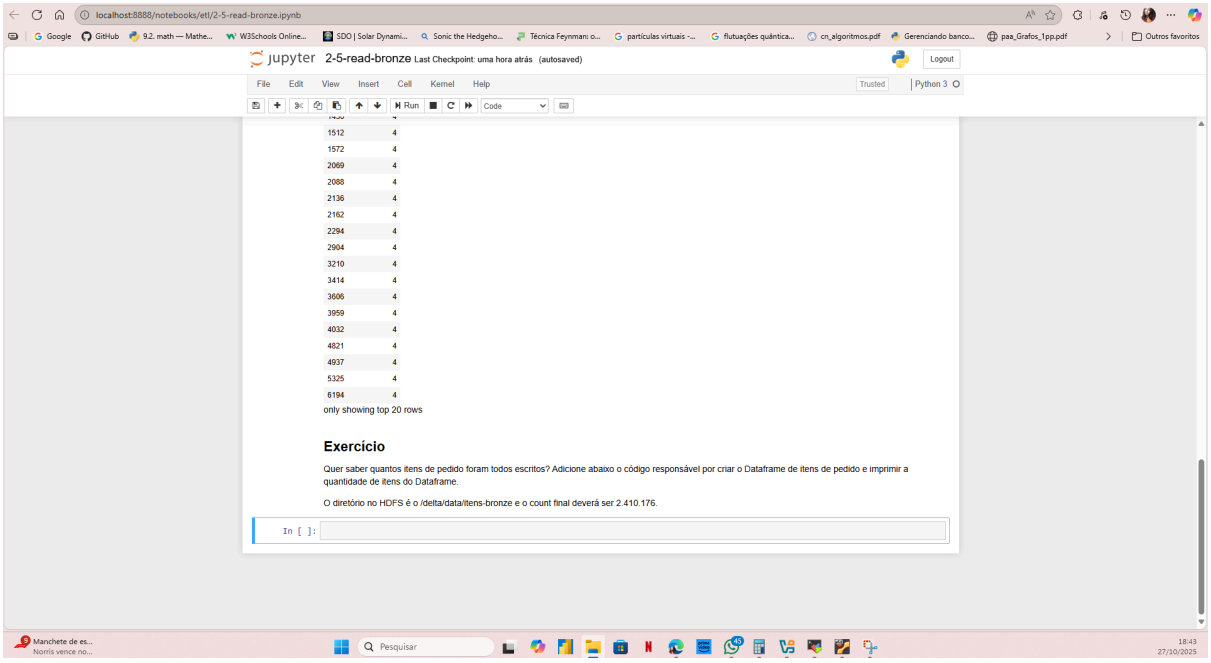


Notebook 2-5-read-bronze




localhost:8888/notebooks/etl/2-5-read-bronze.ipynb
jupyter 2-5-read-bronze Last Checkpoint: uma hora atrás (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
operação no Spark é lazy, portanto o código abaixo cria apenas o link entre o Spark e os dados no HDFS.
In [3]:
pedidos = dfu. \
spark(). \
read. \
format('delta'). \
load(f'{wd}/data/pedidos-bronze')
O trecho abaixo obtém a quantidade de itens no Dataframe, execute-o algumas vezes em intervalos de alguns segundos e observe o valor sendo incrementado.
Veja que não é necessário recriar o Dataframe para que o valor seja atualizado, o Spark já cuida dessa parte para você. O valor final deverá ser 310.976.
In [4]:
pedidos.count()
Out[4]:
1243904
O trecho abaixo cria o Dataframe de clientes exatamente da mesma forma que o de pedidos, alterando apenas o diretório no HDFS.
In [5]:
clientes = dfu. \
spark(). \
read. \
format('delta'). \
load(f'{wd}/data/clientes-bronze')
A quantidade de clientes é muito pequena, portanto quando você executar o trecho abaixo já terá finalizado a escrita dos 15.000 clientes.
In [6]:
clientes.count()
Out[6]:
60000
O trecho abaixo cria uma temp view do Spark para que possamos executar uma consulta SQL.
In [7]:
clientes.createOrReplaceTempView('clientes_bronze')
A SQL abaixo é para verificar que não existem clientes repetidos.
Para isso realizamos um group_by na chave (key) e adicionamos um having count > 1. O resultado abaixo deverá retornar uma lista vazia.
Manchete de es...
Norris vence no...
Pesquisar
18:43
27/10/2025


localhost:8888/notebooks/etl/2-5-read-bronze.ipynb
jupyter 2-5-read-bronze Last Checkpoint: uma hora atrás (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
O trecho abaixo cria uma temp view do Spark para que possamos executar uma consulta SQL.
In [7]:
clientes.createOrReplaceTempView('clientes_bronze')
A SQL abaixo é para verificar que não existem clientes repetidos.
Para isso realizamos um group_by na chave (key) e adicionamos um having count > 1. O resultado abaixo deverá retornar uma lista vazia.
In [8]:
dfu.spark().sql("""
select key, count(value)
from clientes_bronze
group by 1
having count(value) > 1
""")
Out[8]:
key count(value)
1090 4
1159 4
1436 4
1512 4
1572 4
2069 4
2088 4
2136 4
2162 4
2294 4
2904 4
3210 4
3414 4
3606 4
3959 4
4032 4
4821 4
Manchete de es...
Norris vence no...
Pesquisar
18:43
27/10/2025



1512	4
1572	4
2069	4
2088	4
2136	4
2162	4
2294	4
2904	4
3210	4
3414	4
3606	4
3959	4
4032	4
4821	4
4937	4
5325	4
6194	4

only showing top 20 rows

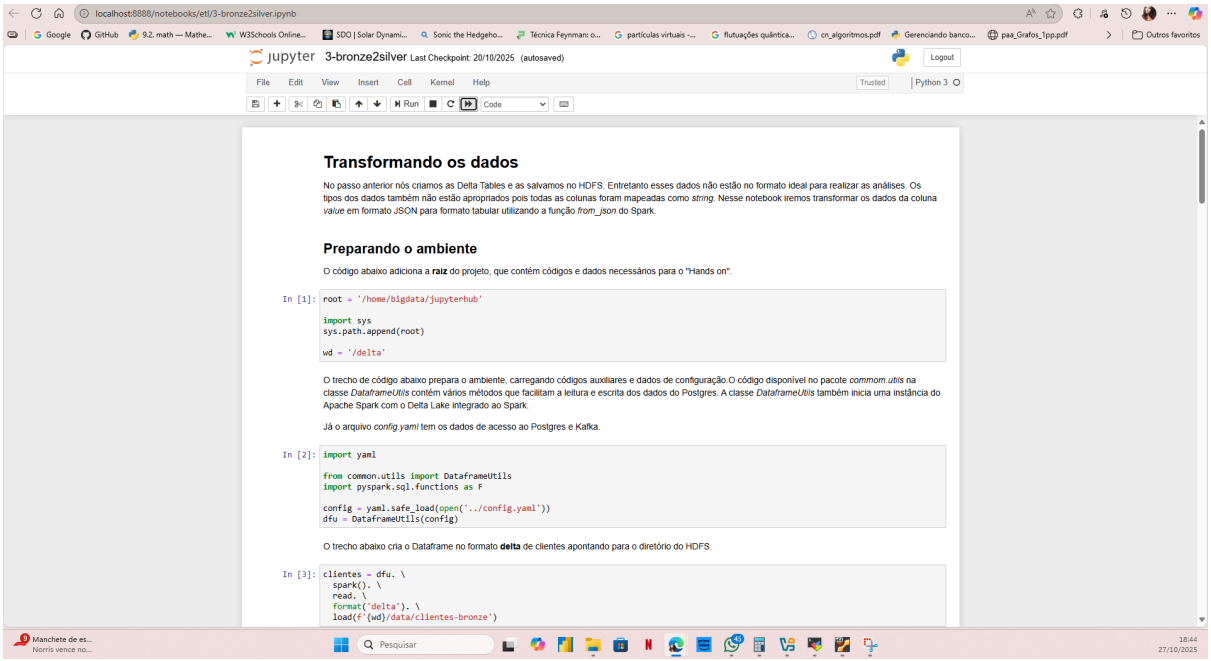
Exercício

Quer saber quantos itens de pedido foram todos escritos? Adicione abaixo o código responsável por criar o Dataframe de itens de pedido e imprimir a quantidade de itens do Dataframe.

O diretório no HDFS é o /delta/data/itens-bronze e o count final deverá ser 2.410.176.

In [ ]:

Notebook 3-bronze2silver



Transformando os dados

No passo anterior nós criamos as Delta Tables e as salvamos no HDFS. Entretanto esses dados não estão no formato ideal para realizar as análises. Os tipos dos dados também não estão apropriados pois todas as colunas foram mapeadas como *string*. Nesse notebook iremos transformar os dados da coluna *value* em formato JSON para formato tabular utilizando a função *from_json* do Spark.

Preparando o ambiente

O código abaixo adiciona a **raiz** do projeto, que contém códigos e dados necessários para o "Hands on".

In [1]:
```
root = '/home/bigdata/jupyterhub'

import sys
sys.path.append(root)

wd = '/delta'
```

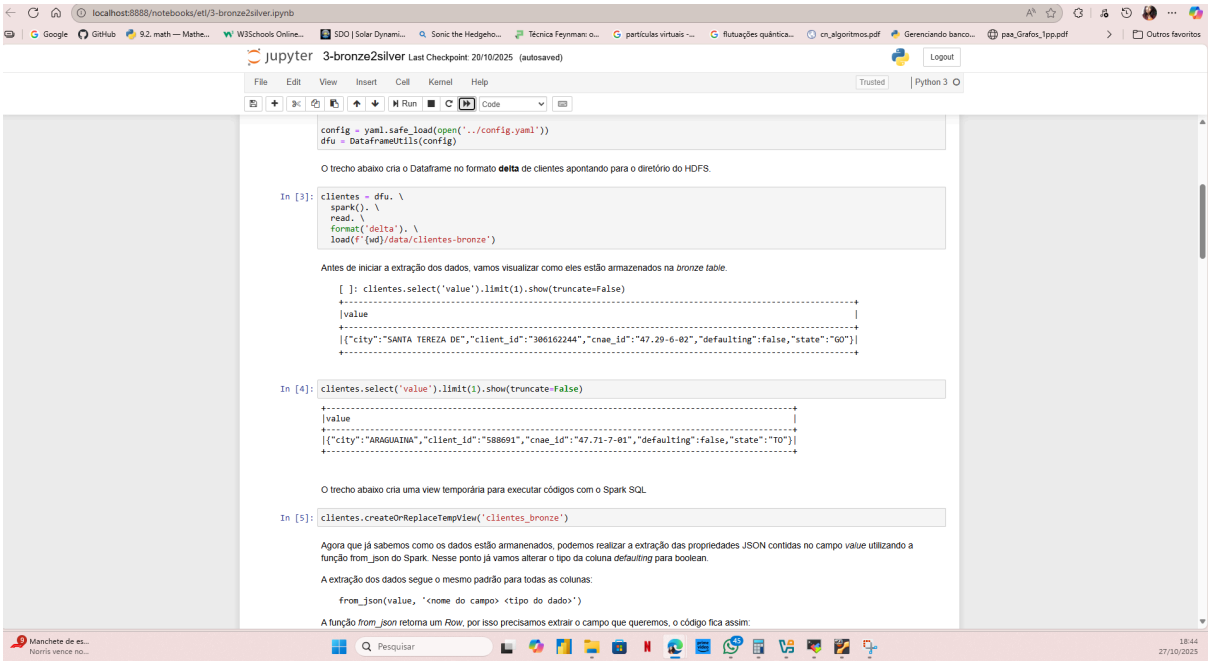
O trecho de código abaixo prepara o ambiente, carregando códigos auxiliares e dados de configuração.O código disponível no pacote *commom.utils* na classe *DataframeUtils* contém vários métodos que facilitam a leitura e escrita dos dados do Postgres. A classe *DataframeUtils* também inicia uma instância do Apache Spark com o Delta Lake integrado ao Spark.

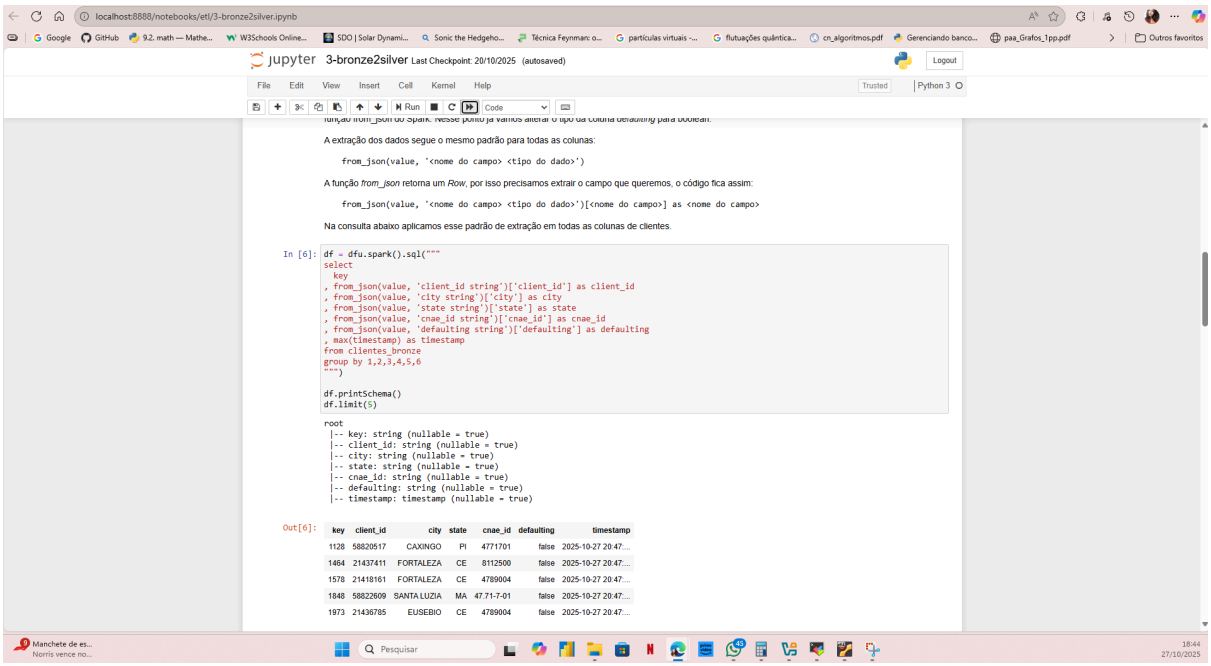
Já o arquivo *config.yaml* tem os dados de acesso ao Postgres e Kafka.

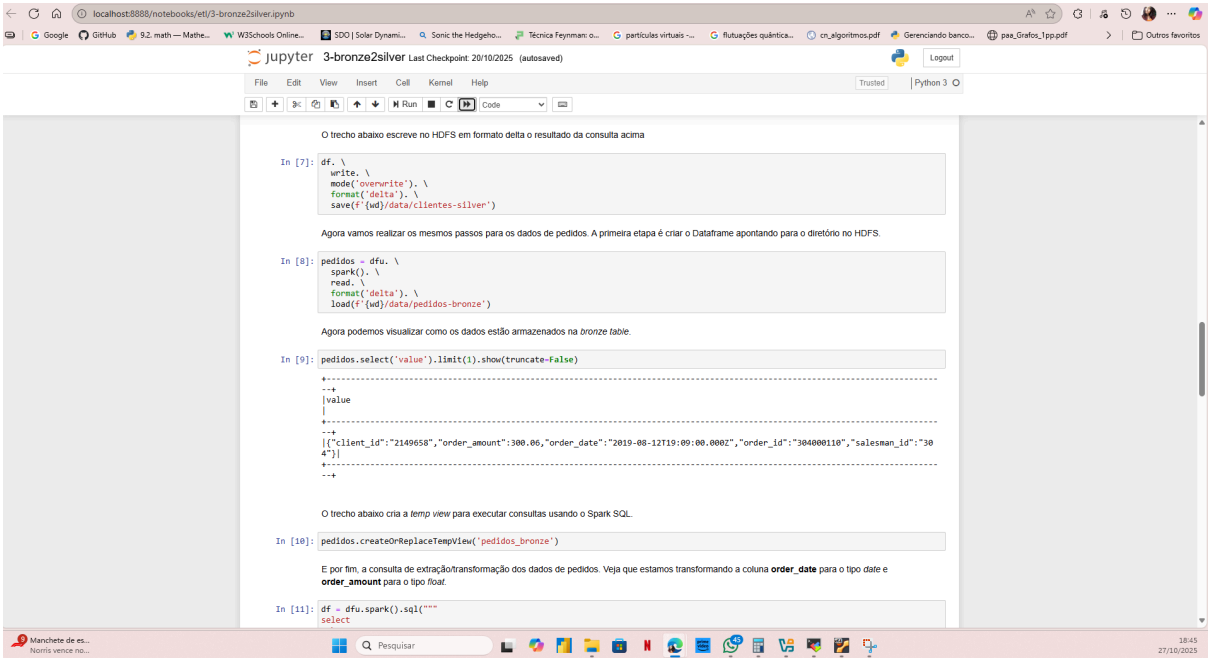
In [2]:
```
import yaml

from common.utils import DataframeUtils
import pyspark.sql.functions as F

config = yaml.safe_load(open('../config.yaml'))
dfu = DataframeUtils(config)
```

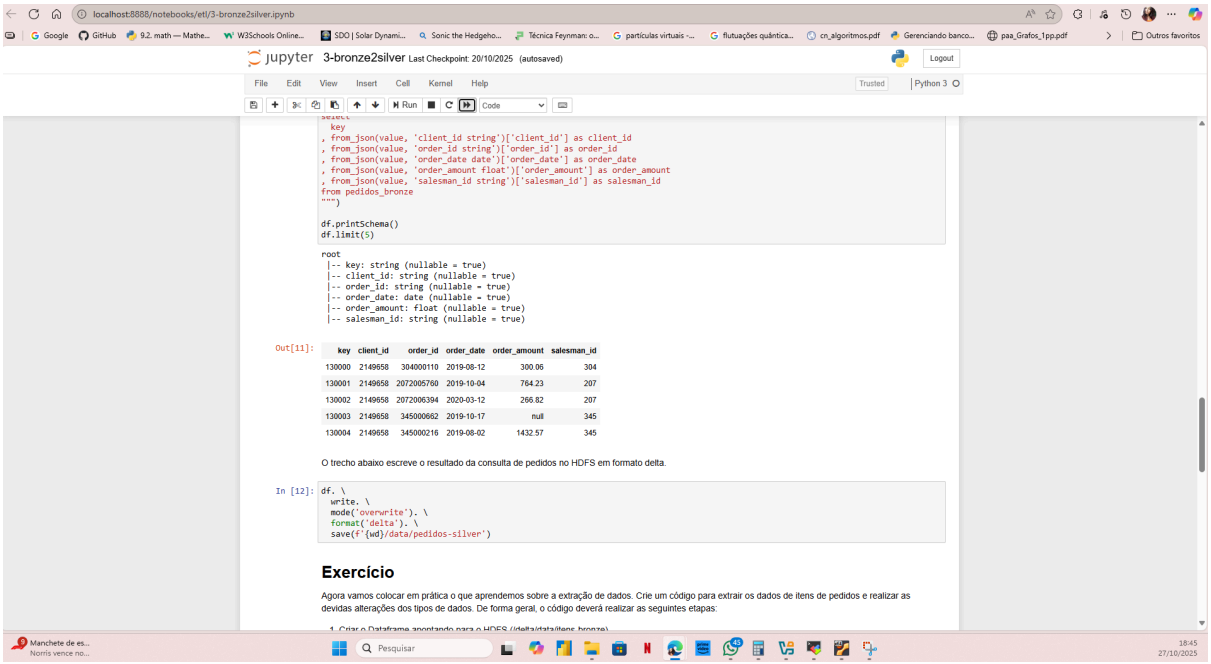
O trecho abaixo cria o Dataframe no formato **delta** de clientes apontando para o diretório do HDFS.

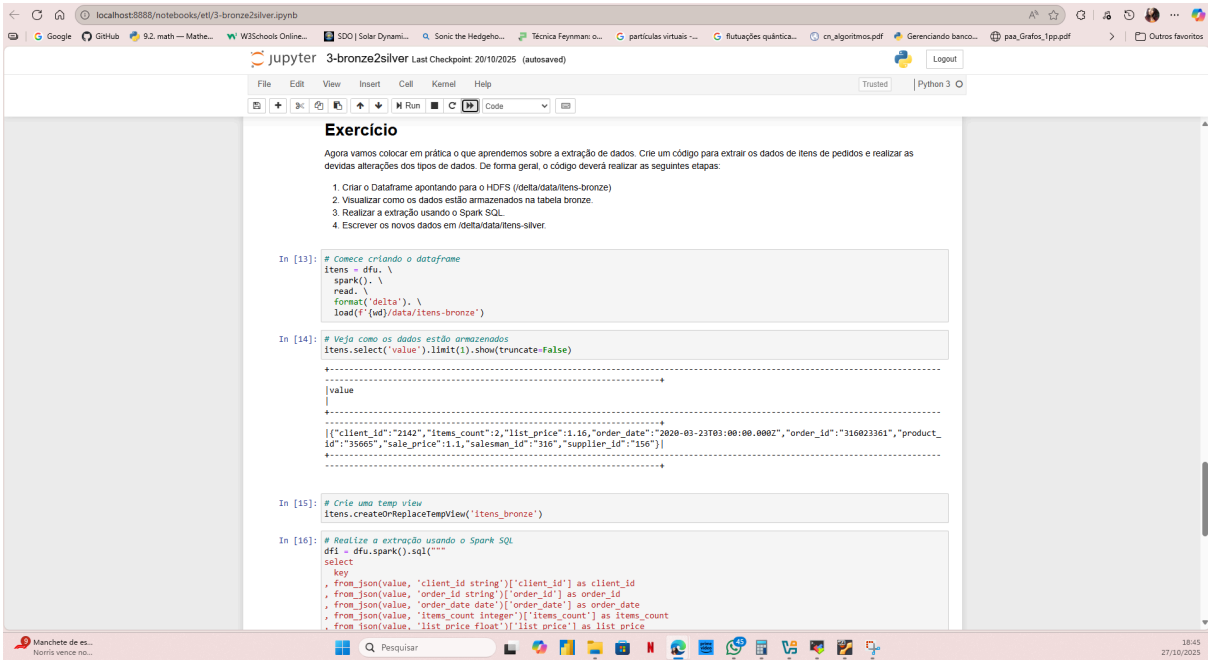
In [3]:
```
clientes = dfu. \
  spark(). \
  read. \
  format('delta'). \
  load(f'{wd}/data/clientes-bronze')
```

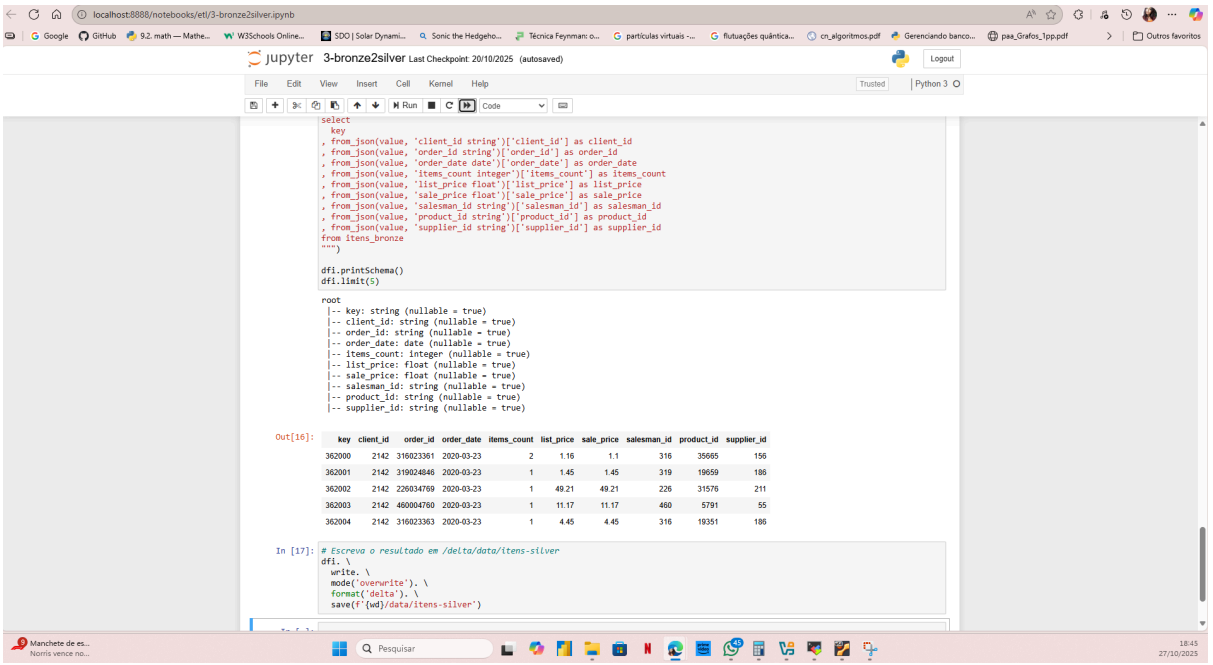

localhost:8888/notebooks/etl/3-bronze2silver.ipynb
jupyter 3-bronze2silver Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
config = yaml.safe_load(open('../config.yaml'))
dfu = DataframeUtils(config)
O trecho abaixo cria o Dataframe no formato delta de clientes apontando para o diretório do HDFS.
In [3]:
clientes = dfu. \
spark(). \
read. \
format('delta'). \
load(f'{wd}/data/clientes-bronze')
Antes de iniciar a extração dos dados, vamos visualizar como eles estão armazenados na bronze table.
[ ]: clientes.select('value').limit(1).show(truncate=False)
|value
|{"city":"SANTA TEREZA DE","client_id":"306162244","cnae_id":"47.29-6-02","defaulting":false,"state":"GO"}|
In [4]:
clientes.select('value').limit(1).show(truncate=False)
|value
|{"city":"ARAGUAINA","client_id":"588691","cnae_id":"47.71-7-01","defaulting":false,"state":"TO"}|
O trecho abaixo cria uma view temporária para executar códigos com o Spark SQL
In [5]:
clientes.createOrReplaceTempView('clientes_bronze')
Agora que já sabemos como os dados estão armanenados, podemos realizar a extração das propriedades JSON contidas no campo value utilizando a função from_json do Spark. Nesse ponto já vamos alterar o tipo da coluna defaulting para boolean.
A extração dos dados segue o mesmo padrão para todas as colunas:
from_json(value, '<nome do campo> <tipo do dado>')
A função from_json retorna um Row, por isso precisamos extrair o campo que queremos, o código fica assim:

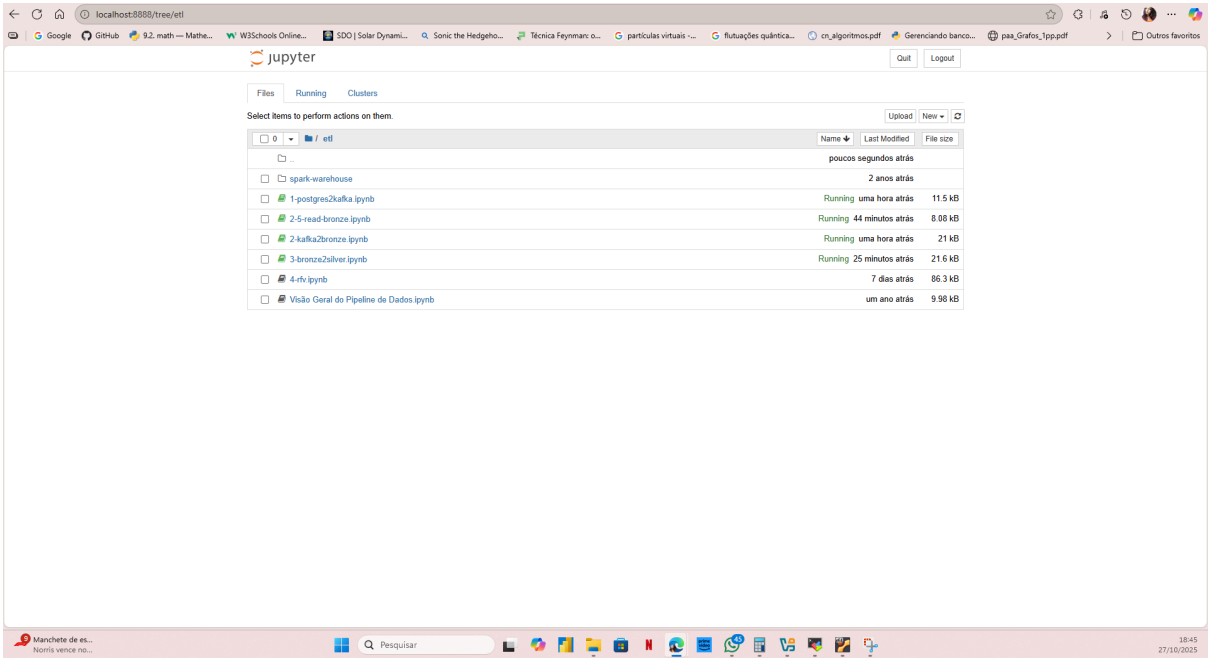

localhost:8888/notebooks/etl/3-bronze2silver.ipynb
jupyter 3-bronze2silver Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
A extração dos dados segue o mesmo padrão para todas as colunas:
from_json(value, '<nome do campo> <tipo do dado>')
A função from_json retorna um Row, por isso precisamos extrair o campo que queremos, o código fica assim:
from_json(value, '<nome do campo> <tipo do dado>')[<nome do campo>] as <nome do campo>
Na consulta abaixo aplicamos esse padrão de extração em todas as colunas de clientes.
In [6]:
df = dfu.spark().sql("""
select
key
, from_json(value, 'client_id string')['client_id'] as client_id
, from_json(value, 'city string')['city'] as city
, from_json(value, 'state string')['state'] as state
, from_json(value, 'cnae_id string')['cnae_id'] as cnae_id
, from_json(value, 'defaulting string')['defaulting'] as defaulting
, max(timestamp) as timestamp
from clientes_bronze
group by 1,2,3,4,5,6
""")
df.printSchema()
df.limit(5)
root
|-- key: string (nullable = true)
|-- client_id: string (nullable = true)
|-- city: string (nullable = true)
|-- state: string (nullable = true)
|-- cnae_id: string (nullable = true)
|-- defaulting: string (nullable = true)
|-- timestamp: timestamp (nullable = true)
Out[6]:
key client_id city state cnae_id defaulting timestamp
1128 58820517 CAXINGO PI 4771701 false 2025-10-27 20:47:...
1464 21437411 FORTALEZA CE 8112500 false 2025-10-27 20:47:...
1578 21418161 FORTALEZA CE 4789004 false 2025-10-27 20:47:...
1848 58822609 SANTA LUZIA MA 47.71-7-01 false 2025-10-27 20:47:...
1973 21436785 EUSEBIO CE 4789004 false 2025-10-27 20:47:...

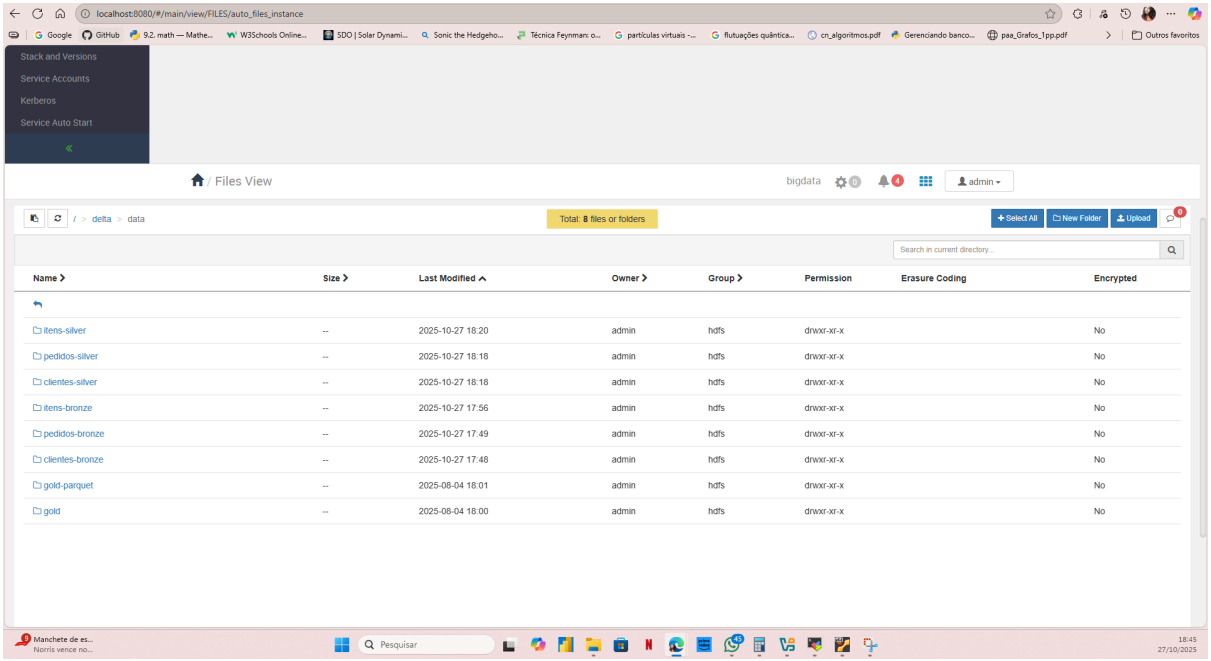

jupyter 3-bronze2silver Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
O trecho abaixo escreve no HDFS em formato delta o resultado da consulta acima
In [7]:
df. \
write. \
mode('overwrite'). \
format('delta'). \
save(f'{wd}/data/clientes-silver')
Agora vamos realizar os mesmos passos para os dados de pedidos. A primeira etapa é criar o Dataframe apontando para o diretório no HDFS.
In [8]:
pedidos = dfu. \
spark(). \
read. \
format('delta'). \
load(f'{wd}/data/pedidos-bronze')
Agora podemos visualizar como os dados estão armazenados na bronze table.
In [9]:
pedidos.select('value').limit(1).show(truncate=False)
+----------
--+
|value
|
+----------
--+
|{"client_id":"2149658","order_amount":300.06,"order_date":"2019-08-12T19:09:00.000Z","order_id":"304000110","salesman_id":"30
4"}|
+----------
--+
O trecho abaixo cria a temp view para executar consultas usando o Spark SQL.
In [10]:
pedidos.createOrReplaceTempView('pedidos_bronze')
E por fim, a consulta de extração/transformação dos dados de pedidos. Veja que estamos transformando a coluna order_date para o tipo date e order_amount para o tipo float.
In [11]:
df = dfu.spark().sql("""
select


jupyter 3-bronze2silver Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
key
, from_json(value, 'client_id string')['client_id'] as client_id
, from_json(value, 'order_id string')['order_id'] as order_id
, from_json(value, 'order_date date')['order_date'] as order_date
, from_json(value, 'order_amount float')['order_amount'] as order_amount
, from_json(value, 'salesman_id string')['salesman_id'] as salesman_id
from pedidos_bronze
""")
df.printSchema()
df.limit(5)
root
|-- key: string (nullable = true)
|-- client_id: string (nullable = true)
|-- order_id: string (nullable = true)
|-- order_date: date (nullable = true)
|-- order_amount: float (nullable = true)
|-- salesman_id: string (nullable = true)
Out[11]:
key client_id order_id order_date order_amount salesman_id
130000 2149658 304000110 2019-08-12 300.06 304
130001 2149658 2072005760 2019-10-04 764.23 207
130002 2149658 2072006394 2020-03-12 266.82 207
130003 2149658 345000662 2019-10-17 null 345
130004 2149658 345000216 2019-08-02 1432.57 345
O trecho abaixo escreve o resultado da consulta de pedidos no HDFS em formato delta.
In [12]:
df. \
write. \
mode('overwrite'). \
format('delta'). \
save(f'{wd}/data/pedidos-silver')
Exercício
Agora vamos colocar em prática o que aprendemos sobre a extração de dados. Crie um código para extrair os dados de itens de pedidos e realizar as devidas alterações dos tipos de dados. De forma geral, o código deverá realizar as seguintes etapas:


localhost:8888/notebooks/etl/3-bronze2silver.ipynb
jupyter 3-bronze2silver Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
Exercício
Agora vamos colocar em prática o que aprendemos sobre a extração de dados. Crie um código para extrair os dados de itens de pedidos e realizar as devidas alterações dos tipos de dados. De forma geral, o código deverá realizar as seguintes etapas:
1. Criar o Dataframe apontando para o HDFS (/delta/data/itens-bronze)
2. Visualizar como os dados estão armazenados na tabela bronze.
3. Realizar a extração usando o Spark SQL.
4. Escrever os novos dados em /delta/data/itens-silver.
In [13]:
# Comece criando o dataframe
itens = dfu. \
spark(). \
read. \
format('delta'). \
load(f'{wd}/data/itens-bronze')
In [14]:
# Veja como os dados estão armazenados
itens.select('value').limit(1).show(truncate=False)
+----------
|value
|
+----------
|{"client_id":"2142","items_count":2,"list_price":1.16,"order_date":"2020-03-23T03:00:00.000Z","order_id":"316023361","product_id":"35665","sale_price":1.1,"salesman_id":"316","supplier_id":"156"}|
+----------
In [15]:
# Crie uma temp view
itens.createOrReplaceTempView('itens_bronze')
In [16]:
# Realize a extração usando o Spark SQL
dfi = dfu.spark().sql("""
select
key
, from_json(value, 'client_id string')['client_id'] as client_id
, from_json(value, 'order_id string')['order_id'] as order_id
, from_json(value, 'order_date date')['order_date'] as order_date
, from_json(value, 'items_count integer')['items_count'] as items_count
, from_json(value, 'list_price float')['list_price'] as list_price


localhost:8888/notebooks/etl/3-bronze2silver.ipynb
jupyter 3-bronze2silver Last Checkpoint: 20/10/2025 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
select
key
, from_json(value, 'client_id string')['client_id'] as client_id
, from_json(value, 'order_id string')['order_id'] as order_id
, from_json(value, 'order_date date')['order_date'] as order_date
, from_json(value, 'items_count integer')['items_count'] as items_count
, from_json(value, 'list_price float')['list_price'] as list_price
, from_json(value, 'sale_price float')['sale_price'] as sale_price
, from_json(value, 'salesman_id string')['salesman_id'] as salesman_id
, from_json(value, 'product_id string')['product_id'] as product_id
, from_json(value, 'supplier_id string')['supplier_id'] as supplier_id
from itens_bronze
""")
dfi.printSchema()
dfi.limit(5)
root
|-- key: string (nullable = true)
|-- client_id: string (nullable = true)
|-- order_id: string (nullable = true)
|-- order_date: date (nullable = true)
|-- items_count: integer (nullable = true)
|-- list_price: float (nullable = true)
|-- sale_price: float (nullable = true)
|-- salesman_id: string (nullable = true)
|-- product_id: string (nullable = true)
|-- supplier_id: string (nullable = true)
Out[16]:
key client_id order_id order_date items_count list_price sale_price salesman_id product_id supplier_id
362000 2142 316023361 2020-03-23 2 1.16 1.1 316 35665 156
362001 2142 319024846 2020-03-23 1 1.45 1.45 319 19659 186
362002 2142 226034769 2020-03-23 1 49.21 49.21 226 31576 211
362003 2142 460004760 2020-03-23 1 11.17 11.17 460 5791 55
362004 2142 316023363 2020-03-23 1 4.45 4.45 316 19351 186
In [17]:
Escreva o resultado em /delta/data/itens-silver
dfi. \
write. \
mode('overwrite'). \
format('delta'). \
save(f'{wd}/data/itens-silver')


localhost:8888/tree/etl
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
/ etl
Name
Last Modified
File size
poucos segundos atrás
spark-warehouse
2 anos atrás
1-postgres2kafka.ipynb
Running uma hora atrás
11.5 kB
2-5-read-bronze.ipynb
Running 44 minutos atrás
8.08 kB
2-kafka2bronze.ipynb
Running uma hora atrás
21 kB
3-bronze2silver.ipynb
Running 25 minutos atrás
21.6 kB
4-rfv.ipynb
7 dias atrás
86.3 kB
Visão Geral do Pipeline de Dados.ipynb
um ano atrás
9.98 kB


localhost:8080/#/main/view/FILES/auto_files_instance
Stack and Versions
Service Accounts
Kerberos
Service Auto Start
/ Files View
bigdata
admin
/ > delta > data
Total: 8 files or folders
Select All
New Folder
Upload
Search in current directory...
Name
Size
Last Modified
Owner
Group
Permission
Erasure Coding
Encrypted
itens-silver -- 2025-10-27 18:20 admin hdfs drwxr-xr-x No
pedidos-silver -- 2025-10-27 18:18 admin hdfs drwxr-xr-x No
clientes-silver -- 2025-10-27 18:18 admin hdfs drwxr-xr-x No
itens-bronze -- 2025-10-27 17:56 admin hdfs drwxr-xr-x No
pedidos-bronze -- 2025-10-27 17:49 admin hdfs drwxr-xr-x No
clientes-bronze -- 2025-10-27 17:48 admin hdfs drwxr-xr-x No
gold-parquet -- 2025-08-04 18:01 admin hdfs drwxr-xr-x No
gold -- 2025-08-04 18:00 admin hdfs drwxr-xr-x No