

Teste Spark

Os estudos e experiências que realizei foram feitos no Databricks Free Edition, contendo algumas limitações de desenvolvimento

Ferramentas:

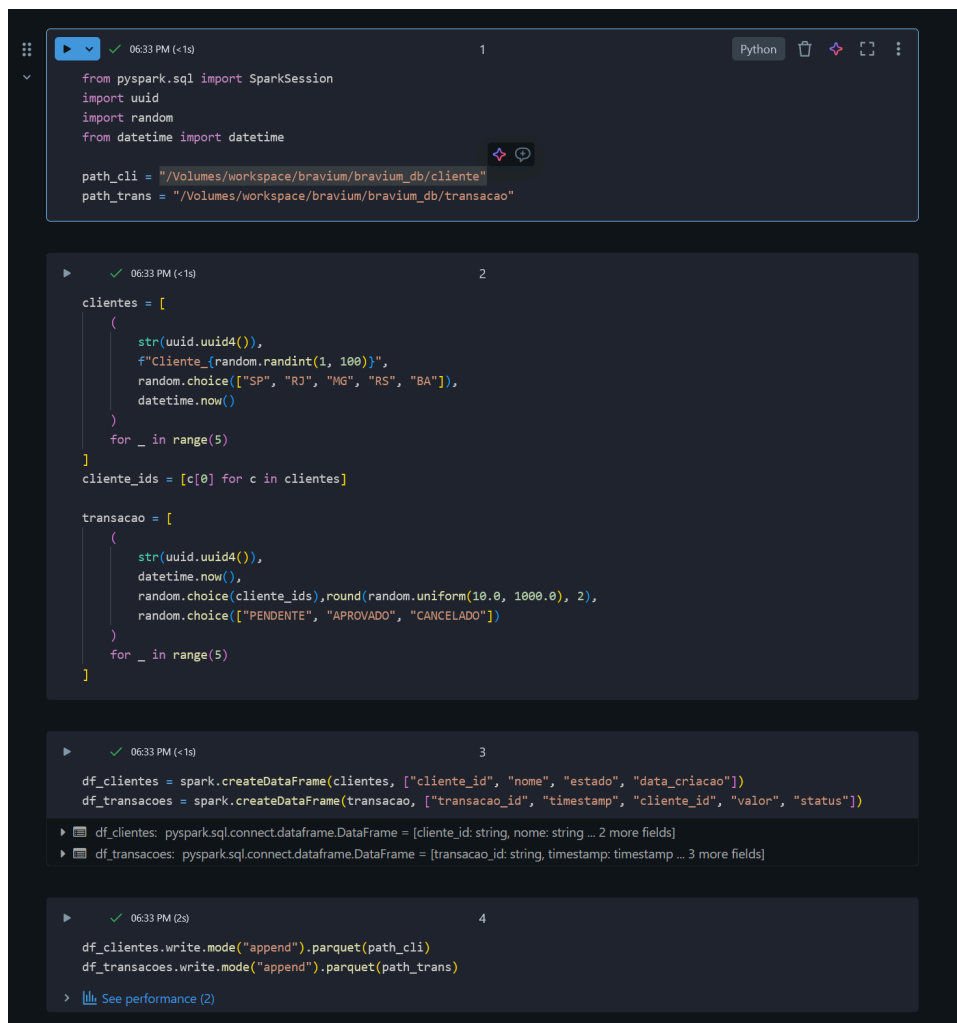
- Databricks
- Spark

Notebooks construídos estão disponíveis em:

<https://github.com/joaopmts/Databricks/tree/main/Bravium>

Gerador de inputs

Fiz um gerador de inputs, simulando duas tabelas: Clientes e Transação. Com valores fictícios e simples para manipulação e visualização pela arquitetura medalhão.



```
from pyspark.sql import SparkSession
import uuid
import random
from datetime import datetime

path_cli = "/Volumes/workspace/bravium/bravium_db/cliente"
path_trans = "/Volumes/workspace/bravium/bravium_db/transacao"

clientes = [
    (
        str(uuid.uuid4()),
        f"Cliente_{random.randint(1, 100)}",
        random.choice(["SP", "RJ", "MG", "RS", "BA"]),
        datetime.now()
    )
    for _ in range(5)
]
cliente_ids = [c[0] for c in clientes]

transacao = [
    (
        str(uuid.uuid4()),
        datetime.now(),
        random.choice(cliente_ids), round(random.uniform(10.0, 1000.0), 2),
        random.choice(["PENDENTE", "APROVADO", "CANCELADO"])
    )
    for _ in range(5)
]

df_clientes = spark.createDataFrame(clientes, ["cliente_id", "nome", "estado", "data_criacao"])
df_transacoes = spark.createDataFrame(transacao, ["transacao_id", "timestamp", "cliente_id", "valor", "status"])

df_clientes.write.mode("append").parquet(path_cli)
df_transacoes.write.mode("append").parquet(path_trans)
```

Tabela Cliente:

- ID
- Nome
- Estado
- Data

Tabela Transação:

- ID
- Data
- ID Cliente
- Valor
- Status

Os arquivos têm output parquet, sendo despejados em um volume no Databricks:

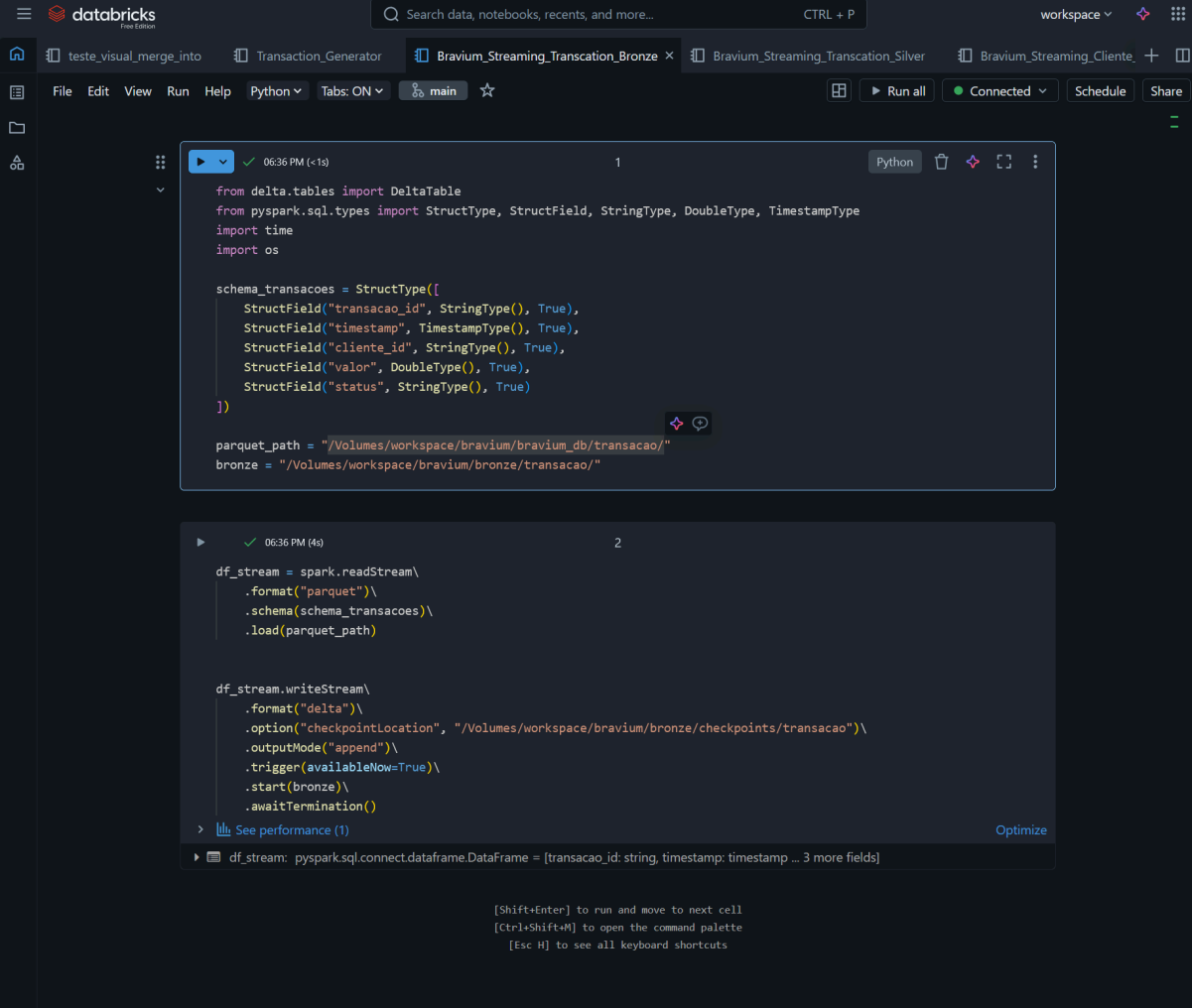
- .../bravium_db/cliente
- .../bravium_db/transacao

Eles simulam o contexto de “Esses dados chegam no Data Lake em arquivos **parquet** com streaming ou estão disponíveis para coleta a partir de conexão com o banco de dados do sistema.”

Arquitetura Medalhão

Transação:

Ingestão Camada Bronze:



The screenshot shows a Databricks workspace with a notebook titled 'Bravium_Streaming_Transacao_Bronze'. The notebook contains two code cells. The first cell defines a schema for transactions and sets the parquet path. The second cell reads the stream from the parquet path and writes it to a Delta table with checkpoints.

```
from delta.tables import DeltaTable
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType
import time
import os

schema_transacoes = StructType([
    StructField("transacao_id", StringType(), True),
    StructField("timestamp", TimestampType(), True),
    StructField("cliente_id", StringType(), True),
    StructField("valor", DoubleType(), True),
    StructField("status", StringType(), True)
])

parquet_path = "/Volumes/workspace/bravium/bravium_db/transacao/"
bronze = "/Volumes/workspace/bravium/bronze/transacao/"

df_stream = spark.readStream\
    .format("parquet")\
    .schema(schema_transacoes)\
    .load(parquet_path)

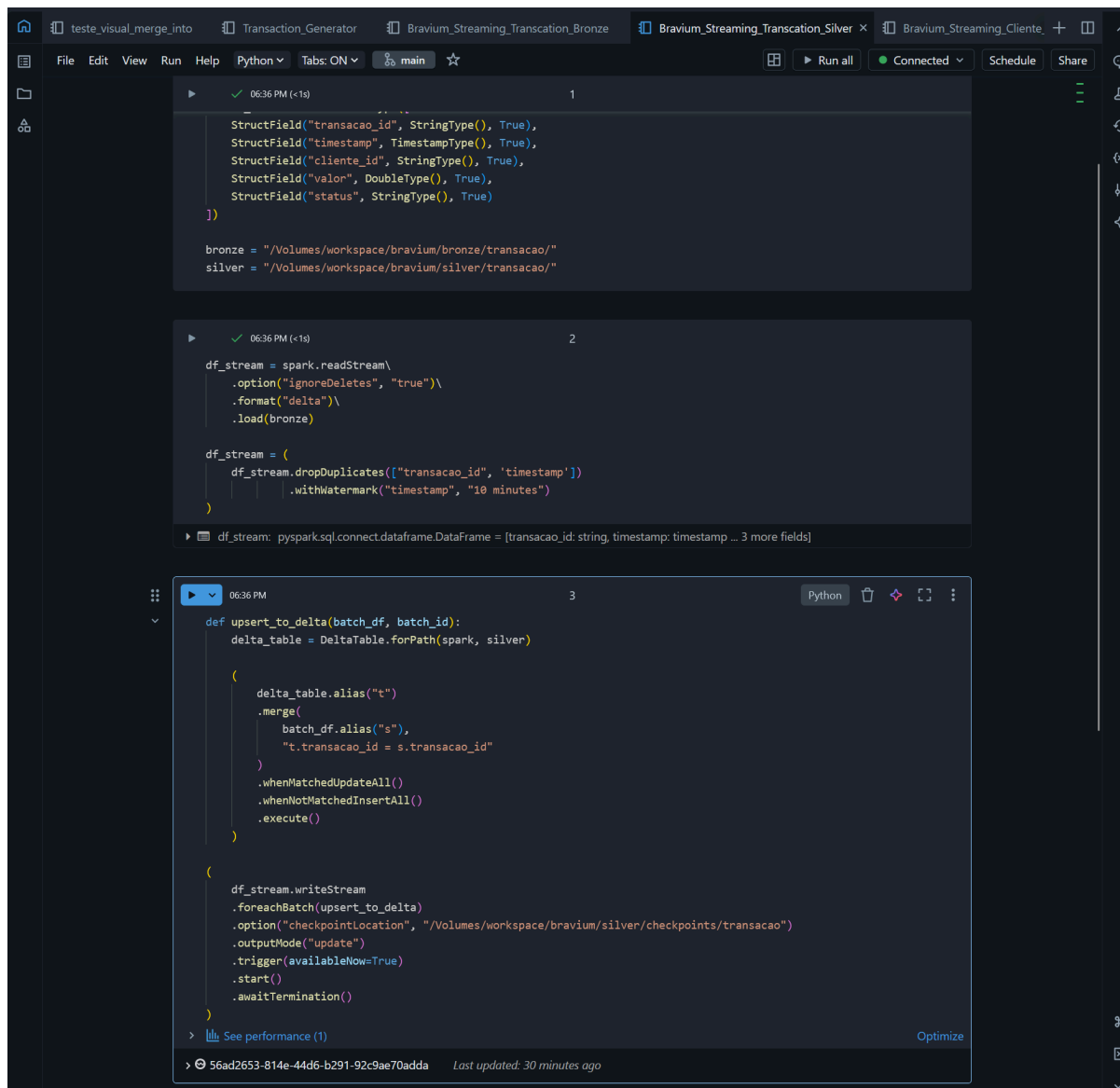
df_stream.writeStream\
    .format("delta")\
    .option("checkpointLocation", "/Volumes/workspace/bravium/bronze/checkpoints/transacao")\
    .outputMode("append")\
    .trigger(availableNow=True)\
    .start(bronze)\
    .awaitTermination()
```

Utilizando Spark Streaming foi feita uma ingestão via Streaming simples, apenas para centralização dos dados dos bancos parceiros. Colocando os arquivos de transação em .../bravium/bronze/transação, porém, agora em formato Delta.

Aqui, queria fazer um streaming contínuo (sem trigger), porém a plataforma me limitou a usar streamings em batch para rodar os jobs no databricks.

Coloquei checkpoint no streaming para ter rastreabilidade da operação, consistência e tolerância a falhas.

Ingestão Camada Silver:



```
06:36 PM (< 1s) 1
StructField("transacao_id", StringType(), True),
StructField("timestamp", TimestampType(), True),
StructField("cliente_id", StringType(), True),
StructField("valor", DoubleType(), True),
StructField("status", StringType(), True)
])

bronze = "/Volumes/workspace/bravium/bronze/transacao/"
silver = "/Volumes/workspace/bravium/silver/transacao/"

06:36 PM (< 1s) 2
df_stream = spark.readStream\
    .option("ignoreDeletes", "true")\
    .format("delta")\
    .load(bronze)

df_stream = (
    df_stream.dropDuplicates(["transacao_id", "timestamp"])
    .withWatermark("timestamp", "10 minutes")
)
df_stream: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]

06:36 PM 3 Python
def upsert_to_delta(batch_df, batch_id):
    delta_table = DeltaTable.forPath(spark, silver)

    (
        delta_table.alias("t")
        .merge(
            batch_df.alias("s"),
            "t.transacao_id = s.transacao_id"
        )
        .whenMatchedUpdateAll()
        .whenNotMatchedInsertAll()
        .execute()
    )

    df_stream.writeStream
        .foreachBatch(upsert_to_delta)
        .option("checkpointLocation", "/Volumes/workspace/bravium/silver/checkpoints/transacao")
        .outputMode("update")
        .trigger(availableNow=True)
        .start()
        .awaitTermination()

> See performance \(1\) Optimize
> 56ad2653-814e-44d6-b291-92c9ae70adda Last updated: 30 minutes ago
```

Para a camada silver, é feita a leitura em Streaming da camada bronze, porém aqui coloquei alguns verificadores como de deduplicação e watermark para evitar processamento de dados duplicados, reproprocessamento de linhas, limitação de memória pelo watermark e verificação de casos “too late” pelo timestamp.

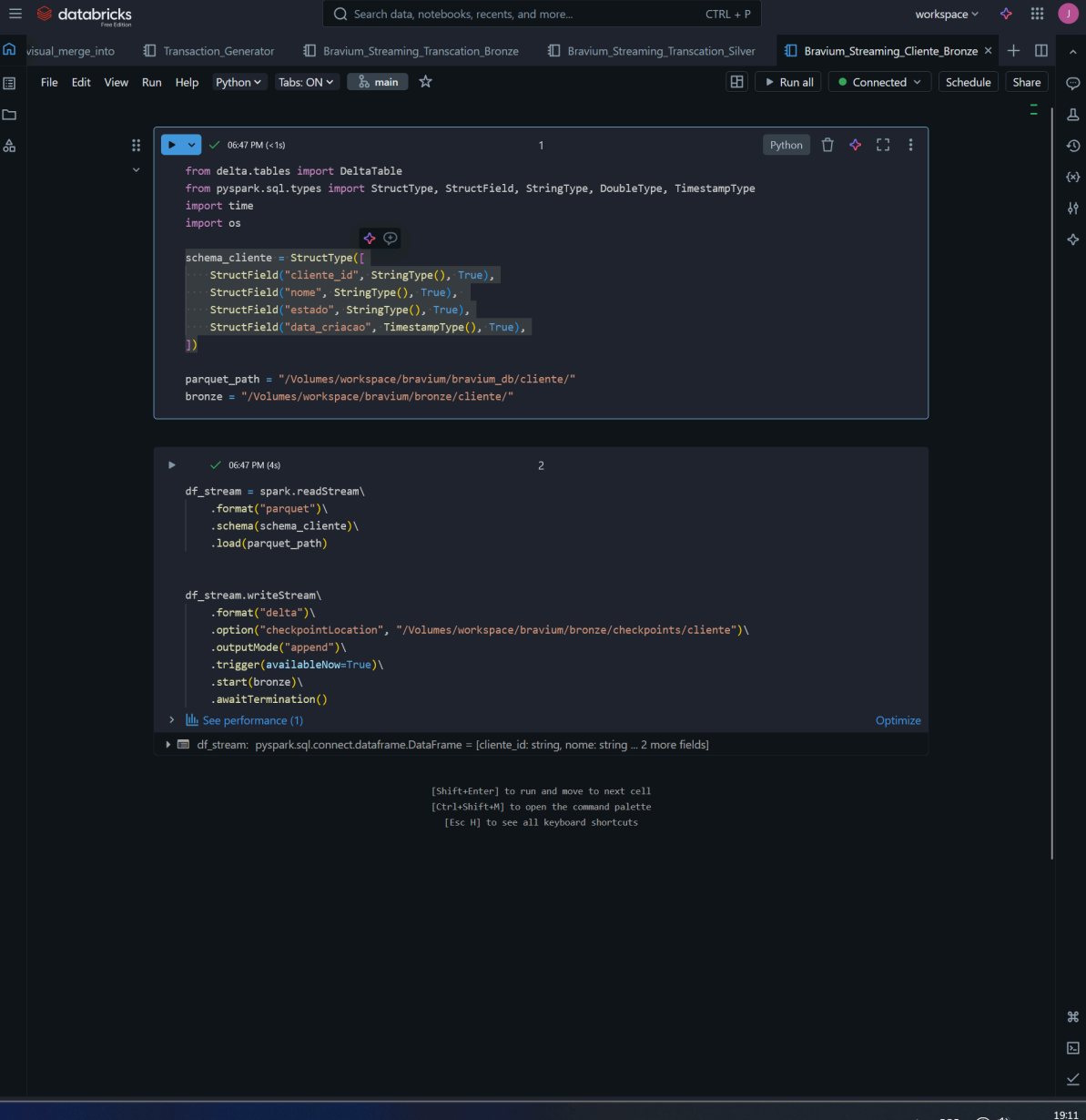
Para a persistência dos dados, fiz um upsert pelo Spark também em formato Delta, utilizando como chave o id da transação.

Também foi colocado checkpoints no processo.

Dados persistidos em .../bravium/silver/transação

Cliente:

Ingestão Camada Bronze:



The screenshot shows the Databricks workspace interface with two notebooks open. The first notebook, titled 'Bravium_Streaming_Cliente_Bronze', contains Python code for defining a schema and paths. The second notebook, titled 'Transaction_Generator', contains code for reading a stream and writing it to a Delta table. The interface includes a top navigation bar, a left sidebar with file explorer and workspace tabs, and a right sidebar with a command palette and shortcuts.

```
from delta.tables import DeltaTable
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType
import time
import os

schema_cliente = StructType([
    StructField("cliente_id", StringType(), True),
    StructField("nome", StringType(), True),
    StructField("estado", StringType(), True),
    StructField("data_criacao", TimestampType(), True),
])

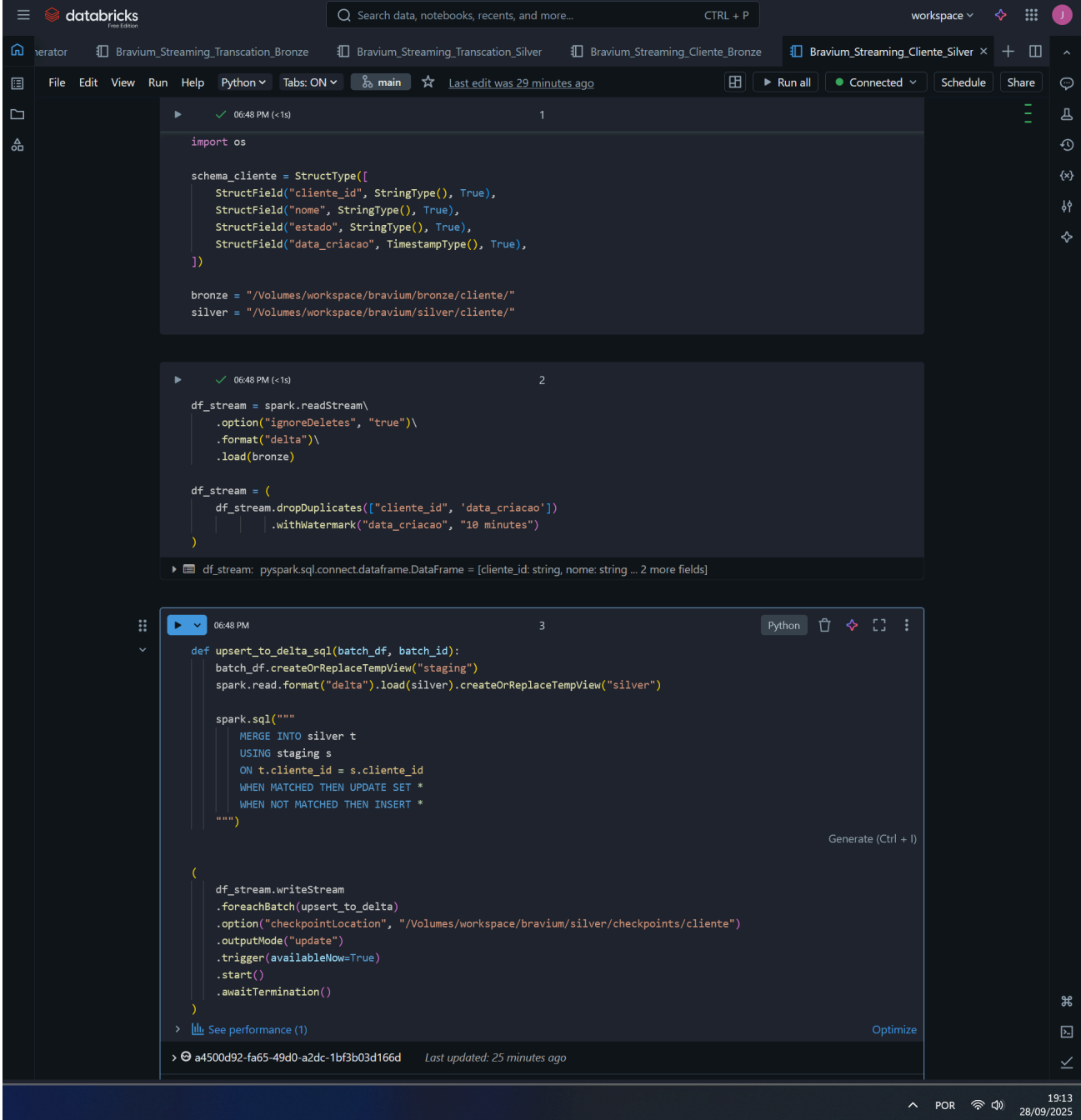
parquet_path = "/Volumes/workspace/bravium/bravium_db/cliente/"
bronze = "/Volumes/workspace/bravium/bronze/cliente/"

df_stream = spark.readStream\
    .format("parquet")\
    .schema(schema_cliente)\
    .load(parquet_path)

df_stream.writeStream\
    .format("delta")\
    .option("checkpointLocation", "/Volumes/workspace/bravium/bronze/checkpoints/cliente")\
    .outputMode("append")\
    .trigger(availableNow=True)\
    .start(bronze)\
    .awaitTermination()
```

Sigo a mesma estrutura de ingestão da tabela de transação

Ingestão Camada Silver:



```
import os

schema_cliente = StructType([
    StructField("cliente_id", StringType(), True),
    StructField("nome", StringType(), True),
    StructField("estado", StringType(), True),
    StructField("data_criacao", TimestampType(), True),
])

bronze = "/Volumes/workspace/bravium/bronze/cliente/"
silver = "/Volumes/workspace/bravium/silver/cliente/"

df_stream = spark.readStream\
    .option("ignoreDeletes", "true")\
    .format("delta")\
    .load(bronze)

df_stream = (
    df_stream.dropDuplicates(["cliente_id", "data_criacao"])
    .withWatermark("data_criacao", "10 minutes")
)

df_stream: pyspark.sql.connect.dataframe.DataFrame = [cliente_id: string, nome: string ... 2 more fields]

def upsert_to_delta_sql(batch_df, batch_id):
    batch_df.createOrReplaceTempView("staging")
    spark.read.format("delta").load(silver).createOrReplaceTempView("silver")

    spark.sql("""
        MERGE INTO silver t
        USING staging s
        ON t.cliente_id = s.cliente_id
        WHEN MATCHED THEN UPDATE SET *
        WHEN NOT MATCHED THEN INSERT *
    """)

    (
        df_stream.writeStream
        .foreachBatch(upsert_to_delta)
        .option("checkpointLocation", "/Volumes/workspace/bravium/silver/checkpoints/cliente")
        .outputMode("update")
        .trigger(availableNow=True)
        .start()
        .awaitTermination()
    )

> See performance \(1\) Optimize

> a4500d92-fa65-49d0-a2dc-1bf3b03d166d Last updated: 25 minutes ago
```

Também sigo a mesma estrutura da tabela de transação, porém com uma diferença. No upsert agora fiz pelo Spark SQL, utilizando a função MERGE INTO.

Quando criei a estrutura, criei todos em formato Delta, mas não salvei uma das tabelas como DeltaTable. Por isso a adaptação para o spark na execução da query SQL.

Testes

Eu orquestrei os notebooks utilizando o Jobs do Databricks, porém a visualização do resultado não é tão clara devido ao volume de dados criados.

Por isso, optei por fazer um teste unitário simples que representa o que acontece no passo a passo do fluxo.

O fluxo segue desta maneira:

Em paralelo:

- Gerador de Transações/Clientes
- Ingestão Bronze de Transação e Clientes
- Ingestão Silver de Transação e Clientes

Para uma forma mais nítida de visualização, os passos foram feitos em série, porém com os testes pelo Jobs, rodam em paralelo normalmente.

Tabela de transação:

teste_visual_merge_intoTransaction_GeneratorBravium_Streaming_Transaction_BronzeBravium_Streaming_Transaction_SilverBravium_Streaming_Cliente

FileEditViewRunHelpPython ▾Tabs: ON ▾main ☆ Last edit was 34 minutes ago

06:34 PM (3)

1

```
display(df_tran)
display(bronze_tran)
display(silver_tran)
#display(bronze_tran.filter(bronze_tran.transacao_id == '9b4Fca8c-f970-4731-ac5d-46fe2868784a'))
#display(silver_tran.filter(silver_tran.transacao_id == '9b4Fca8c-f970-4731-ac5d-46fe2868784a'))
```

See performance (3)

bronze_tran: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]

df_tran: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]

silver_tran: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]

Table ▾ +

transacao_idtimestampcliente_idvalorstatus

1

fd11398-9a59-4855-9838-f9dcb6fb14c9

2025-09-28T21:33:43.232+00:00

ba0995cd-44b7-49f3-bee1-56dddee8bf1

376.02

CANCELADO

2

5962d130-2c45-4715-b17a-768d46f65802

2025-09-28T21:33:43.232+00:00

296cca9b-afb8-41d3-b153-1cd122ac30...

32.16

PENDENTE

3

0a9d773a-031f-4e24-a97a-6fb2ce1b8c24

2025-09-28T21:33:43.232+00:00

da1004fa-b45f-4ec3-ba7f-adfe890164d7

314.9

APROVADO

4

7a165210-35fd-42e0-b392-004b2b59f243

2025-09-28T21:33:43.232+00:00

da1004fa-b45f-4ec3-ba7f-adfe890164d7

274.58

APROVADO

5

e2750046-ba23-4aaa-8142-bad7e0a11364

2025-09-28T21:33:43.232+00:00

296cca9b-afb8-41d3-b153-1cd122ac30...

431.29

APROVADO

5 rows | 2.93s runtime

Refreshed 46 minutes ago

Table ▾ +

transacao_idtimestampcliente_idvalorstatus

1

fd11398-9a59-4855-9838-f9dcb6fb14c9

2025-09-28T21:33:43.232+00:00

ba0995cd-44b7-49f3-bee1-56dddee8bf1

376.02

CANCELADO

2

5962d130-2c45-4715-b17a-768d46f65802

2025-09-28T21:33:43.232+00:00

296cca9b-afb8-41d3-b153-1cd122ac30...

32.16

PENDENTE

3

0a9d773a-031f-4e24-a97a-6fb2ce1b8c24

2025-09-28T21:33:43.232+00:00

da1004fa-b45f-4ec3-ba7f-adfe890164d7

314.9

APROVADO

4

7a165210-35fd-42e0-b392-004b2b59f243

2025-09-28T21:33:43.232+00:00

da1004fa-b45f-4ec3-ba7f-adfe890164d7

274.58

APROVADO

5

e2750046-ba23-4aaa-8142-bad7e0a11364

2025-09-28T21:33:43.232+00:00

296cca9b-afb8-41d3-b153-1cd122ac30...

431.29

APROVADO

5 rows | 2.93s runtime

Refreshed 46 minutes ago

Table ▾ +

transacao_idtimestampcliente_idvalorstatus

1

fd11398-9a59-4855-9838-f9dcb6fb14c9

2025-09-28T21:33:43.232+00:00

ba0995cd-44b7-49f3-bee1-56dddee8bf1

376.02

CANCELADO

2

7a165210-35fd-42e0-b392-004b2b59f243

2025-09-28T21:33:43.232+00:00

da1004fa-b45f-4ec3-ba7f-adfe890164d7

274.58

APROVADO

3

0a9d773a-031f-4e24-a97a-6fb2ce1b8c24

2025-09-28T21:33:43.232+00:00

da1004fa-b45f-4ec3-ba7f-adfe890164d7

314.9

APROVADO

4

e2750046-ba23-4aaa-8142-bad7e0a11364

2025-09-28T21:33:43.232+00:00

296cca9b-afb8-41d3-b153-1cd122ac30...

431.29

APROVADO

5

5962d130-2c45-4715-b17a-768d46f65802

2025-09-28T21:33:43.232+00:00

296cca9b-afb8-41d3-b153-1cd122ac30...

32.16

PENDENTE

5 rows | 2.93s runtime

Refreshed 46 minutes ago

06:36 PM (1)

2

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType
from datetime import datetime
```

Como primeiro passo, executei o notebook para gerar valores para Transação, obtendo os resultados da primeira visualização. Após executei o notebook fazendo a ingestão para a camada bronze e em seguida para a camada Silver.


```
06:36 PM (1s) 2

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType
from datetime import datetime





schema_transacoes = StructType([
    StructField("transacao_id", StringType(), True),
    StructField("timestamp", TimestampType(), True),
    StructField("cliente_id", StringType(), True),
    StructField("valor", DoubleType(), True),
    StructField("status", StringType(), True)
])

data = [
    ("fdd11398-9a59-4855-9838-f9dcb6fb14c9",
     datetime.fromisoformat("2025-09-28T18:41:15.681+00:00"),
     "7597e9cd-9c37-42a3-a977-1d62806436bf",
     756.15,
     "APROVADO"
    )
]

df = spark.createDataFrame(data, schema=schema_transacoes)

df.write.mode("append").parquet("/Volumes/workspace/bravium/bravium_db/transacao/")
> See performance \(1\) Optimize
df: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]
```

Em seguida inserir novos dados, no volume que simula os bancos parceiros. Com novas informações para a transação “fdd11398-9a59-4855-9838-f9dcb6fb14c9”. Mudando o status de CANCELADO para APROVADO

 transacao_id	 timestamp	 cliente_id	1.2 valor	 status
fdd11398-9a59-4855-9838-f9dcb6fb14c9	2025-09-28T21:33:43.232+00:00	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	376.02	CANCELADO

A seguir, fiz a execução dos scripts de Streaming para Bronze e Silver. Obtendo os seguintes resultados:

06:37 PM (3s)

3

```
df_tran = spark.read.format("parquet").load("/Volumes/workspace/bravium/bravium_db/transacao/")
bronze_tran = spark.read.format("delta").load("/Volumes/workspace/bravium/bronze/transacao/")
silver_tran = spark.read.format("delta").load("/Volumes/workspace/bravium/silver/transacao/")

display(df_tran.filter(df_tran.transacao_id == 'fdd11398-9a59-4855-9838-f9dcb6fb14c9'))
display(bronze_tran.filter(bronze_tran.transacao_id == 'fdd11398-9a59-4855-9838-f9dcb6fb14c9'))
display(silver_tran.filter(silver_tran.transacao_id == 'fdd11398-9a59-4855-9838-f9dcb6fb14c9'))
```

> [See performance \(3\)](#)

▶ bronze_tran: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]

▶ df_tran: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]

▶ silver_tran: pyspark.sql.connect.dataframe.DataFrame = [transacao_id: string, timestamp: timestamp ... 3 more fields]

Table +

🔍

🔼

📄

🗒

	<div>transacao_id</div>	<div>timestamp</div>	<div>cliente_id</div>	1.2 valor	<div>status</div>
1	fdd11398-9a59-4855-9838-f9dcb6fb14...	2025-09-28T21:33:43.232+00:00	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	376.02	CANCELADO
2	fdd11398-9a59-4855-9838-f9dcb6fb14...	2025-09-28T18:41:15.681+00:00	7597e9cd-9c37-42a3-a977-1d62806436...	756.15	APROVADO

📄

🔼

2 rows | 2.62s runtime

Refreshed 48 minutes ago

Table +

🔍

🔼

📄

🗒

	<div>transacao_id</div>	<div>timestamp</div>	<div>cliente_id</div>	1.2 valor	<div>status</div>
1	fdd11398-9a59-4855-9838-f9dcb6fb14...	2025-09-28T21:33:43.232+00:00	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	376.02	CANCELADO
2	fdd11398-9a59-4855-9838-f9dcb6fb14...	2025-09-28T18:41:15.681+00:00	7597e9cd-9c37-42a3-a977-1d62806436...	756.15	APROVADO

📄

🔼

2 rows | 2.62s runtime

Refreshed 48 minutes ago

Table +

🔍

🔼

📄

🗒

	<div>transacao_id</div>	<div>timestamp</div>	<div>cliente_id</div>	1.2 valor	<div>status</div>
1	fdd11398-9a59-4855-9838-f9dcb6fb14...	2025-09-28T18:41:15.681+00:00	7597e9cd-9c37-42a3-a977-1d62806436...	756.15	APROVADO

Tabela de clientes:

Interface do Databricks mostrando o código Python executado no notebook `teste_visual_merge_into`. O código carrega dados de clientes de diferentes fontes (parquet, delta) e os exibe. Abaixo, três tabelas resultantes da execução, todas com 5 linhas e 2.92s de runtime.

```
df_cliente = spark.read.format("parquet").load("/Volumes/workspace/bravium/bravium_db/cliente/")
bronze_cliente = spark.read.format("delta").load("/Volumes/workspace/bravium/bronze/cliente/")
silver_cliente = spark.read.format("delta").load("/Volumes/workspace/bravium/silver/cliente/")

display(df_cliente)
display(bronze_cliente)
display(silver_cliente)
#display(df_cliente.filter(df_cliente.transacao_id == '9b4fca8c-f970-4731-ac5d-46fe2868784a'))
#display(bronze_cliente.filter(bronze_cliente.transacao_id == '9b4fca8c-f970-4731-ac5d-46fe2868784a'))
#display(silver_cliente.filter(silver_cliente.transacao_id == '9b4fca8c-f970-4731-ac5d-46fe2868784a'))
```

Tabela 1: df_cliente

	cliente_id	nome	estado	data_criacao
1	296cca9b-afb8-41d3-b153-1cd122ac304e	Cliente_38	SP	2025-09-28T21:33:43.232+00:00
2	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Cliente_25	RJ	2025-09-28T21:33:43.232+00:00
3	bd217e8d-2a34-4450-bbf1-b1fed3e00098	Cliente_22	RJ	2025-09-28T21:33:43.232+00:00
4	da1004fa-b45f-4ec3-ba7f-adfe890164d7	Cliente_46	BA	2025-09-28T21:33:43.232+00:00
5	28541225-1425-4649-b9d7-cea6abb5d1c3	Cliente_19	MG	2025-09-28T21:33:43.232+00:00

Tabela 2: bronze_cliente

	cliente_id	nome	estado	data_criacao
1	296cca9b-afb8-41d3-b153-1cd122ac304e	Cliente_38	SP	2025-09-28T21:33:43.232+00:00
2	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Cliente_25	RJ	2025-09-28T21:33:43.232+00:00
3	bd217e8d-2a34-4450-bbf1-b1fed3e00098	Cliente_22	RJ	2025-09-28T21:33:43.232+00:00
4	da1004fa-b45f-4ec3-ba7f-adfe890164d7	Cliente_46	BA	2025-09-28T21:33:43.232+00:00
5	28541225-1425-4649-b9d7-cea6abb5d1c3	Cliente_19	MG	2025-09-28T21:33:43.232+00:00

Tabela 3: silver_cliente

	cliente_id	nome	estado	data_criacao
1	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Cliente_25	RJ	2025-09-28T21:33:43.232+00:00
2	bd217e8d-2a34-4450-bbf1-b1fed3e00098	Cliente_22	RJ	2025-09-28T21:33:43.232+00:00
3	da1004fa-b45f-4ec3-ba7f-adfe890164d7	Cliente_46	BA	2025-09-28T21:33:43.232+00:00
4	296cca9b-afb8-41d3-b153-1cd122ac304e	Cliente_38	SP	2025-09-28T21:33:43.232+00:00
5	28541225-1425-4649-b9d7-cea6abb5d1c3	Cliente_19	MG	2025-09-28T21:33:43.232+00:00

Os mesmos processos foram feitos para a tabela de clientes, obtendo os resultados acima seguindo a arquitetura medalhão.

Para o teste individual, fiz o input de uma modificação do cliente "ba0995cd-44b7-49f3-bee1-56dcfdee8bf1", mudando o nome para João e Estado para SP.

```
▶ 06:46 PM (1s) 5

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType
from datetime import datetime

schema_cliente = StructType([
    StructField("cliente_id", StringType(), True),
    StructField("nome", StringType(), True),
    StructField("estado", StringType(), True),
    StructField("data_criacao", TimestampType(), True),
])

data = [(
    "ba0995cd-44b7-49f3-bee1-56dcfdee8bf1",
    "Joao",
    "SP",
    datetime.fromisoformat("2025-09-28T18:41:15.681+00:00"),
)]

df = spark.createDataFrame(data, schema=schema_cliente)

df.write.mode("append").parquet("/Volumes/workspace/bravium/bravium_db/cliente")
> See performance \(1\) Optimize

▶ df: pyspark.sql.connect.dataframe.DataFrame = [cliente_id: string, nome: string ... 2 more fields]
```

E assim obtive os seguintes resultados:

06:48 PM (3s)6

```
df_cliente = spark.read.format("parquet").load("/Volumes/workspace/bravium/bravium_db/cliente/")
bronze_cliente = spark.read.format("delta").load("/Volumes/workspace/bravium/bronze/cliente/")
silver_cliente = spark.read.format("delta").load("/Volumes/workspace/bravium/silver/cliente/")

display(df_cliente.filter(df_cliente.cliente_id == 'ba0995cd-44b7-49f3-bee1-56dcfdee8bf1'))
display(bronze_cliente.filter(bronze_cliente.cliente_id == 'ba0995cd-44b7-49f3-bee1-56dcfdee8bf1'))
display(silver_cliente.filter(silver_cliente.cliente_id == 'ba0995cd-44b7-49f3-bee1-56dcfdee8bf1'))
```

> [See performance \(3\)](#)

▶ bronze_cliente: pyspark.sql.connect.dataframe.DataFrame = [cliente_id: string, nome: string ... 2 more fields]

▶ df_cliente: pyspark.sql.connect.dataframe.DataFrame = [cliente_id: string, nome: string ... 2 more fields]

▶ silver_cliente: pyspark.sql.connect.dataframe.DataFrame = [cliente_id: string, nome: string ... 2 more fields]

Table +

🔍 🔍 📄 🗑

	cliente_id	nome	estado	data_criacao
1	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Cliente_25	RJ	2025-09-28T21:33:43.232+00:00
2	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Joao	SP	2025-09-28T18:41:15.681+00:00

📄

2 rows | 2.55s runtime

Refreshed 40 minutes ago

Table +

🔍 🔍 📄 🗑

	cliente_id	nome	estado	data_criacao
1	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Cliente_25	RJ	2025-09-28T21:33:43.232+00:00
2	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Joao	SP	2025-09-28T18:41:15.681+00:00

📄

2 rows | 2.55s runtime

Refreshed 40 minutes ago

Table +

🔍 🔍 📄 🗑

	cliente_id	nome	estado	data_criacao
1	ba0995cd-44b7-49f3-bee1-56dcfdee8bf1	Joao	SP	2025-09-28T18:41:15.681+00:00

📄

1 row | 2.55s runtime

Refreshed 40 minutes ago

Conclusão

Considero um sucesso a aplicação dos conceitos de arquitetura medalhão, de Spark Streaming e persistência dos dados no Data Lake utilizando os conceitos básicos de banco de Dados como ACID e Idempotência, também como de otimização dos processos de ETL.