

Relatório: Implementação de ETL no Databricks

1. Introdução

O objetivo deste relatório é apresentar a criação de um processo de ETL (Extract, Transform, Load) utilizando a plataforma Databricks. Esse ambiente foi escolhido devido à sua capacidade de lidar com grandes volumes de dados de maneira escalável, eficiente e colaborativa, além de integrar ferramentas de análise de dados e machine learning em um único espaço.

2. Por que utilizar o Databricks para ETL?

a. Escalabilidade e Performance

O Databricks permite processar dados em larga escala utilizando clusters distribuídos, otimizando o tempo de execução de tarefas complexas.

b. Integração com Diferentes Fontes de Dados

Oferece conectores para bancos de dados relacionais (SQL Server, PostgreSQL, etc.), data lakes (Azure Data Lake, AWS S3), APIs e arquivos locais (CSV, JSON, Parquet).

c. Interface Colaborativa e Reprodutibilidade

Com notebooks interativos, a equipe pode colaborar em tempo real, versionar código e compartilhar análises.

d. Compatibilidade com Linguagens Populares

Suporte a PySpark, SQL, Scala e R, o que oferece flexibilidade na escolha da linguagem para transformar os dados.

e. Economia de Custos

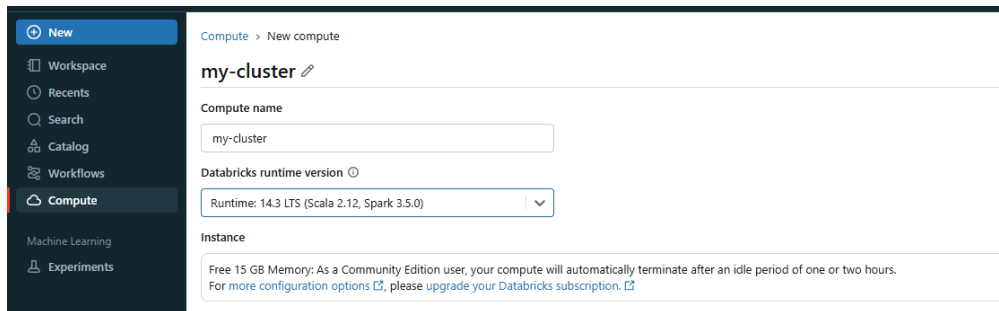
Clusters podem ser configurados para desligar automaticamente quando o processamento é concluído, evitando custos desnecessários.

A criação do **cluster** no Databricks é o primeiro passo para começar qualquer processamento de dados ou análise. Um cluster é, essencialmente, um conjunto de máquinas virtuais que trabalham juntas para executar suas tarefas de processamento. Ele é fundamental porque fornece os recursos de computação necessários para lidar com grandes volumes de dados e executar operações complexas.

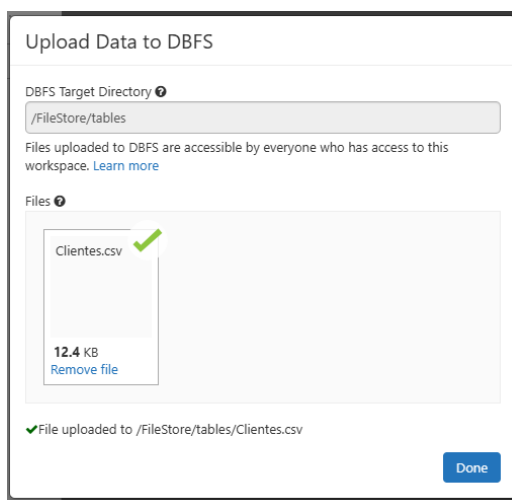
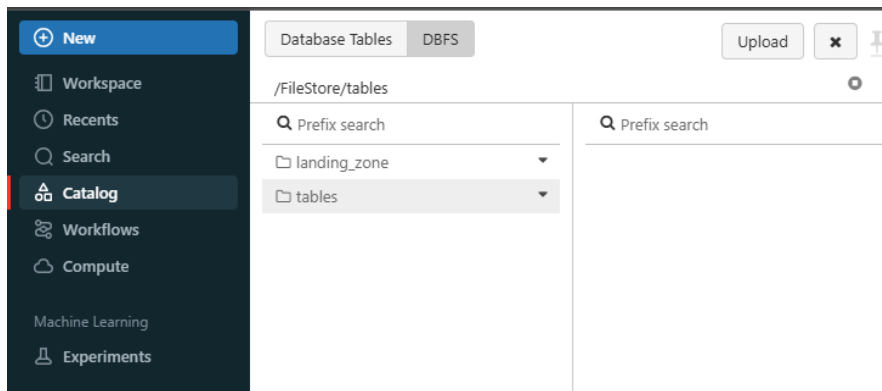
O que o cluster faz?

- **Processamento de Dados em Larga Escala:** Utiliza a arquitetura distribuída do Spark para processar grandes volumes de dados de forma rápida e eficiente.
- **Execução de Códigos e Tarefas:** Todos os scripts Python (PySpark), SQL, R ou Scala que você executa no Databricks são processados dentro do cluster.
- **Armazenamento Temporário de Dados:** Durante o processamento, os dados intermediários são armazenados na memória ou no disco do cluster.

- **Escalabilidade:** Pode ser ajustado para lidar com diferentes cargas de trabalho, adicionando mais nós ou aumentando a capacidade.



Exportação de uma Tabela como CSV no Databricks

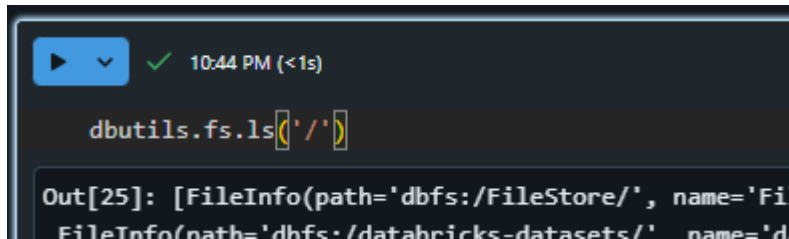


Biblioteca dbutils

O *dbutils* é uma biblioteca nativa do Databricks que oferece utilitários para facilitar operações comuns em um ambiente de notebooks. Ele é amplamente utilizado para gerenciamento de arquivos no sistema de arquivos do Databricks (DBFS), execução de comandos, manipulação de dados e configuração de parâmetros.

Comandos e Justificativas

1. Listando Diretórios e Arquivos



```
dbutils.fs.ls('/')

Out[25]: [FileInfo(path='dbfs:/FileStore/', name='FileStore'),
          FileInfo(path='dbfs:/databricks-datasets/', name='databricks-datasets/'),
          ...]
```

- **Objetivo:**
 - a. O comando `dbutils.fs.ls(path)` lista o conteúdo de um diretório no DBFS (Databricks File System).
 - b. O método `display()` é usado para visualizar o resultado em um formato de tabela no notebook.
- **Por que executar?**
 - a. Serve para verificar quais diretórios e arquivos estão disponíveis no caminho especificado, neste caso, na raiz `/` e na pasta `/FileStore/tables`.

2. Visualizando o Conteúdo de um Arquivo



```
display(dbutils.fs.ls('/FileStore/tables'))

▶ (3) Spark Jobs
```

- **Objetivo:**
 - O comando `dbutils.fs.head(file_path)` exibe as primeiras linhas de um arquivo no DBFS.
- **Por que executar?**
 - Ajuda a inspecionar o conteúdo do arquivo antes de processá-lo, garantindo que os dados estão no formato esperado (como CSV, JSON, etc.).

3. Criando uma Lista de Arquivos para Exclusão

```

excluir = [
    "/FileStore/DRE.xlsx",
    "/FileStore/order_items.csv",
    "/FileStore/orders.csv",
    "/FileStore/products.csv",
]

for apagar_arquivos in excluir:
    dbutils.fs.rm(apagar_arquivos)

```

- **Objetivo:**
 - a. A lista excluir contém os caminhos dos arquivos que serão excluídos no DBFS.
- **Por que criar?**
 - b. Centraliza os arquivos a serem apagados, facilitando a manipulação em massa.

4. Iterando e Removendo Arquivos

- **Objetivo:**
 - a. O método `dbutils.fs.rm(path)` remove um arquivo ou diretório no DBFS.
 - b. O loop `for` percorre cada item da lista `excluir` e executa a remoção.
- **Por que executar?**
 - c. Este procedimento limpa arquivos desnecessários do sistema, liberando espaço e mantendo a organização do diretório.

5. Verificando Diretórios Após Alterações

```
display(dbutils.fs.ls('/FileStore/tables'))
```

► (3) Spark Jobs

Table ▾ +				
	^A _C path	^A _C name	¹ ₃ size	¹ ₃ modificationTime
2	dbfs/FileStore/tables/Clientes.csv	Clientes.csv	12356	1733015498000
3	dbfs/FileStore/tables/DRE.csv	DRE.csv	699	1733194109000
4	dbfs/FileStore/tables/DRE.xlsx	DRE.xlsx	631425	1733194117000
5	dbfs/FileStore/tables/DiferentesSaidas/	DiferentesSaidas/	0	0

- **Objetivo:**
 - a. Reexecuta a listagem da pasta `/FileStore/tables` para confirmar que os arquivos foram removidos.

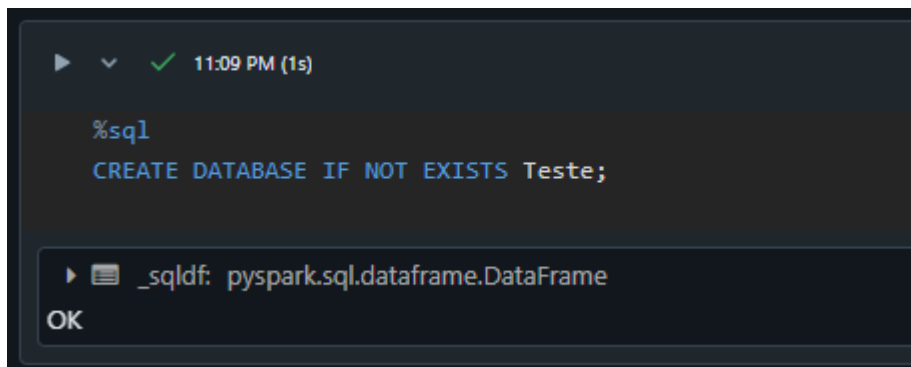
- **Por que executar?**
 - b. Garante que as alterações esperadas foram aplicadas corretamente e que os arquivos deletados não aparecem mais no diretório.

Criando Tabelas via Comando SQL no Databricks

Neste procedimento, utilizamos comandos SQL para criar um banco de dados e tabelas no Databricks, carregando os dados diretamente de um arquivo CSV armazenado no Databricks File System (DBFS). Esse método é útil para estruturar os dados de forma tabular, permitindo consultas posteriores.

Etapas Realizadas

1. Criação do Banco de Dados



The screenshot shows a Databricks SQL execution window. At the top, there is a play button, a dropdown arrow, a green checkmark, and the timestamp '11:09 PM (1s)'. Below this, the SQL command is displayed: `%sql` followed by `CREATE DATABASE IF NOT EXISTS Teste;`. At the bottom, a message indicates the command was executed successfully: `_sqlidf: pyspark.sql.dataframe.DataFrame`, followed by an 'OK' button.

- **Comando: CREATE DATABASE**
 - **Objetivo:** Criar um banco de dados chamado Teste caso ele ainda não exista.
 - **Por que executar?**
Garantir que o banco de dados Teste esteja disponível para armazenar as tabelas. O uso de IF NOT EXISTS evita erros caso o banco já exista.

2. Definindo o Escopo do Banco de Dados

- **Comando: USE**
 - **Objetivo:** Definir o banco de dados Teste como o escopo ativo para que as próximas operações (como criação de tabelas ou consultas) sejam realizadas neste banco.
 - **Por que executar?**
Direcionar todas as operações para o banco de dados correto, evitando confusões com outros bancos existentes.

3. Criação da Tabela “DRE”

```
use Teste;  
CREATE TABLE IF NOT EXISTS DRE  
USING csv  
OPTIONS (  
  'path' '/FileStore/tables/DRE.csv',  
  'header' 'true',  
  'inferSchema' 'true'  
);
```

- **Comando: CREATE TABLE**
 - **Objetivo:** Criar uma tabela chamada DRE utilizando o arquivo CSV localizado no DBFS.
 - **Opções Especificadas:**
 - USING csv: Especifica o formato do arquivo de origem como CSV.
 - OPTIONS:
 - 'path' '/FileStore/tables/DRE.csv': Define o caminho do arquivo CSV no DBFS.
 - 'header' 'true': Indica que a primeira linha do arquivo contém os nomes das colunas.
 - 'inferSchema' 'true': Permite que o sistema deduza automaticamente os tipos de dados das colunas no arquivo CSV.
 - **Por que executar?**
 - Transformar os dados do arquivo CSV em uma tabela SQL dentro do banco de dados Teste, facilitando consultas e análises estruturadas.

4. Consultando os Dados da Tabela

▶

3 minutes ago (1s)

5

%sql

SELECT * FROM dre|

▶ (1) Spark Jobs

▶

_sqldf: pyspark.sql.dataframe.DataFrame = [Cod.DRE: integer, Descrição: string ... 2 more fields]

Table

▼

+

	¹ ₃ Cod.DRE	^A _C Descrição	^A _C Operação	^A _C Tipo	
1	1	RECEITA BRUTA	(+)	A	
2	2	Deduções da Receita	(-)	A	
3	3	RECEITA LÍQUIDA	null	ST	
4	4	Salários e Encargos	(-)	A	
5	5	Gastos com Pessoal	(-)	A	

Organização da Arquitetura dos Dados

Camada Bronze - Processamento de Dados Brutos

A camada Bronze é responsável por armazenar os dados em seu formato bruto, antes de qualquer processamento ou transformação. O objetivo dessa camada é preservar a integridade dos dados de origem, garantindo que qualquer análise posterior possa ser auditada e reprocessada, se necessário. A camada Bronze serve como o primeiro ponto de entrada para os dados no pipeline de processamento, garantindo um histórico fiel e inalterado das informações.

Estrutura de Diretórios

Os dados são armazenados em diretórios separados para cada tabela de dados, seguindo a estrutura de diretórios:

```
/mnt/bronze
├── dre/
├── brands/
├── categories/
├── order_items/
├── orders/
├── staffs/
├── stocks/
└── stores /
```

Cada diretório contém os dados em formato Delta, que é um formato otimizado para grandes volumes de dados e facilita operações de leitura, escrita e atualização.

Fontes de Dados

Os dados brutos são extraídos de diferentes fontes, como arquivos CSV e JSON, localizados em diferentes caminhos no sistema. Abaixo estão os detalhes das tabelas e seus respectivos caminhos:

Tabelas CSV:

1. **brands** - Localização: /FileStore/tables/brands.csv
2. **categories** - Localização: /FileStore/tables/categories.csv
3. **order_items** - Localização: /FileStore/tables/order_items.csv
4. **orders** - Localização: /FileStore/tables/orders.csv
5. **staffs** - Localização: /FileStore/tables/staffs.csv
6. **stocks** - Localização: /FileStore/tables/stocks.csv
7. **stores** - Localização: /FileStore/tables/stores.csv
8. **dre** - Localização: /FileStore/tables/dre.csv

Código Implementado

O código responsável por esse processamento é o seguinte:


```

# Definindo a localização da camada Bronze
bronze_path = "/mnt/bronze"

# Lista de tabelas e seus respectivos caminhos CSV
tabelas_bronze = {
    "dre": "/FileStore/tables/DRE.csv",
    "brands": "/FileStore/tables/brands.csv",
    "categories": "/FileStore/tables/categories.csv",
    "order_items": "/FileStore/tables/order_items.csv",
    "orders": "/FileStore/tables/orders.csv",
    "staffs": "/FileStore/tables/staffs.csv",
    "stocks": "/FileStore/tables/stocks.csv",
    "stores": "/FileStore/tables/stores.csv"
}

# Loop para processar cada tabela e salvar na camada Bronze
for tabela, caminho_csv in tabelas_bronze.items():
    print(f"Processando tabela: {tabela}")

    # Leitura do CSV bruto
    df = spark.read.format("csv") \
        .option("header", "true") \
        .option("inferSchema", "true") \
        .load(caminho_csv)

    # Caminho para salvar a tabela na camada Bronze
    bronze_tabela_path = f"{bronze_path}/{tabela}"

    # Salvando os dados no formato Delta
    df.write.format("delta") \
        .mode("overwrite") \
        .save(bronze_tabela_path)

    print(f"Tabela {tabela} salva na camada Bronze em {bronze_tabela_path}")

# Exibindo a conclusão do processo
print("Camada Bronze criada com sucesso!")

```

Benefícios da Camada Bronze

- **Integridade dos Dados:** Os dados são armazenados de forma bruta e imutável, o que permite reproprocessamentos futuros com os dados originais.
- **Auditoria:** A camada Bronze oferece um ponto de auditoria para garantir a rastreabilidade dos dados ao longo do pipeline.
- **Escalabilidade:** Utilizando o formato Delta, é possível realizar operações de leitura e escrita em grande escala de forma eficiente.
- **Facilidade de Processamento Posterior:** Os dados na camada Bronze estão prontos para serem processados em camadas posteriores (Silver e Gold), com a possibilidade de aplicar transformações ou enriquecimentos adicionais.

Camada Silver- Refinando os Dados

A camada **Silver** foi criada para fornecer dados refinados e mais estruturados a partir da camada **Bronze**. As tabelas nesta camada são mais limpas e consistem em informações

prontas para análise, com transformações básicas aplicadas, como a remoção de duplicatas e a conversão de tipos de dados.

Estrutura da Camada Silver

1. Tabelas Processadas

A camada **Silver** contém as seguintes tabelas:

Tabela	Descrição	Localização no Sistema de Arquivos
brands	Informações sobre as marcas disponíveis	/mnt/silver/brands
categories	Categorias de produtos ou serviços oferecidos	/mnt/silver/categories
order_items	Itens de cada pedido realizado na plataforma	/mnt/silver/order_items
orders	Detalhes sobre os pedidos feitos pelos clientes	/mnt/silver/orders
staffs	Informações sobre os funcionários envolvidos nos pedidos	/mnt/silver/staffs
stocks	Quantidade de estoque disponível para os produtos	/mnt/silver/stocks
stores	Detalhes das lojas	/mnt/silver/stores
dre	Dados receitas e informações de contabilidade	/mnt/silver/dre

Processamento das Tabelas

As tabelas são processadas da camada **Bronze** para a camada **Silver** seguindo os seguintes passos:

1. **Leitura de Dados:** As tabelas são lidas a partir da camada **Bronze**, que contém dados brutos sem processamento ou limpeza.
2. **Remoção de Duplicatas:** Para garantir a integridade dos dados, as duplicatas são removidas.
3. **Transformações de Tipos de Dados:** Algumas transformações de tipos de dados são feitas para garantir que as colunas tenham o tipo correto (por exemplo, conversão de strings para data, onde aplicável).
 - a. Exemplo: Se a coluna date estiver presente, ela é convertida para o tipo date.
4. **Armazenamento na Camada Silver:** Após as transformações, os dados são salvos em sua versão refinada na camada **Silver**.

Armazenamento na Camada Silver

As tabelas processadas são salvas na camada **Silver** em formato Delta, permitindo que elas sejam acessadas facilmente e prontas para análises avançadas ou agregações.

- **Comando de Salvamento:**

```
silver_tabela_path = f"{silver_path}/{tabela}"

df_silver.write.format("delta") \
    .mode("overwrite") \
    .save(silver_tabela_path)
```

- **Criação de Tabelas no Catálogo:** As tabelas também são registradas no catálogo Delta para fácil acesso através de consultas SQL.

```
# Lista de tabelas e seus respectivos caminhos
tabelas = [
    {"nome": "brands", "caminho": "/mnt/silver/brands"},
    {"nome": "categories", "caminho": "/mnt/silver/categories"},
    {"nome": "order_items", "caminho": "/mnt/silver/order_items"},
    {"nome": "orders", "caminho": "/mnt/silver/orders"},
    {"nome": "staffs", "caminho": "/mnt/silver/staffs"},
    {"nome": "stocks", "caminho": "/mnt/silver/stocks"},
    {"nome": "stores", "caminho": "/mnt/silver/stores"},
    {"nome": "ocorrencias_aeronauticas", "caminho": "/mnt/silver/ocorrencias_aeronauticas"},
]

# Processar cada tabela e salvá-la no schema silver
for tabela in tabelas:
    nome_tabela = tabela["nome"]
    caminho = tabela["caminho"]

    # Carregar a tabela do armazenamento Delta
    df = spark.read.format("delta").load(caminho)

    # Salvar a tabela no catálogo no schema silver
    df.write.format("delta") \
        .mode("overwrite") \
        .saveAsTable(f"silver.{nome_tabela}")

    print(f"Tabela {nome_tabela} salva no schema silver com sucesso!")
```

Benefícios da Camada Silver

- **Limpeza e Consistência:** A camada **Silver** garante que os dados sejam consistentes e limpos, com duplicatas removidas e tipos de dados corrigidos.
- **Prontidão para Análises:** As tabelas na camada **Silver** estão prontas para serem utilizadas em análises mais avançadas, agregações e dashboards.
- **Armazenamento em Delta:** O uso do formato Delta permite transações ACID, o que garante a integridade dos dados e possibilita versionamento e controle de mudanças.

Como Usar as Tabelas na Camada Silver

Para acessar as tabelas da camada **Silver** em consultas, você pode utilizar o SQL ou o PySpark:

- **SQL:**

```
%sql
select * from silver.stores
```

- **PySpark:**

```
df_brands = spark.table("silver.brands")
df_brands.display()
```

Camada Gold- Estruturação para análise

A camada **Gold** representa a camada final no pipeline de dados, onde os dados estão totalmente preparados para análise e geração de relatórios. Nesta camada, os dados são agregados, consolidados e estruturados para atender diretamente às necessidades de negócios e insights estratégicos.

Estrutura da Camada Gold

A camada **Gold** contém tabelas que fornecem informações analíticas refinadas. Atualmente, estão disponíveis as seguintes tabelas:

Tabela	Descrição	Localização no Sistema de Arquivos
gold_clients	Informações consolidadas sobre os clientes, incluindo número de pedidos e valor total gasto	/mnt/gold/gold_clients
gold_store_performance	Métricas de desempenho das lojas, como receita total, número de pedidos e estoque restante	/mnt/gold/gold_store_performance

Processamento das Tabelas

1. Tabela gold_clients

Descrição:

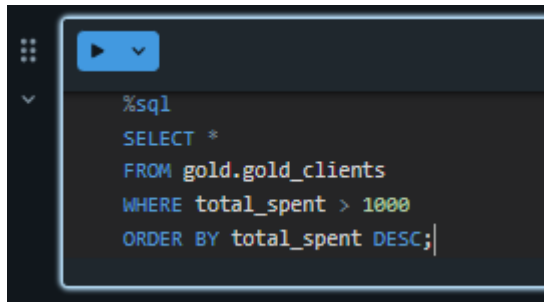
Essa tabela consolida informações relacionadas aos clientes, com base nos pedidos realizados e itens pedidos. Ela fornece métricas como:

- **Total de Pedidos (total_orders):** Número total de pedidos realizados por cliente.
- **Quantidade Total (total_quantity):** Quantidade total de itens adquiridos.
- **Gasto Total (total_spent):** Soma total do valor pago pelos itens, considerando descontos.
- **Total de Descontos (total_discount):** Total economizado por cliente em descontos.

Processamento:

1. **Fonte de Dados:** Junção das tabelas **orders** e **order_items** da camada Silver.
2. **Transformações Aplicadas:**
 - a. Agrupamento por **customer_id**.
 - b. Cálculo de métricas agregadas.
3. **Armazenamento:** Dados salvos como tabela Delta e registrados no catálogo Delta sob o esquema **gold**.

Exemplo de Query SQL:



```
%sql
SELECT *
FROM gold.gold_clients
WHERE total_spent > 1000
ORDER BY total_spent DESC;
```

2. Tabela gold_store_performance

Descrição:

Essa tabela apresenta métricas de desempenho por loja, considerando vendas, receita, itens vendidos e estoque restante.

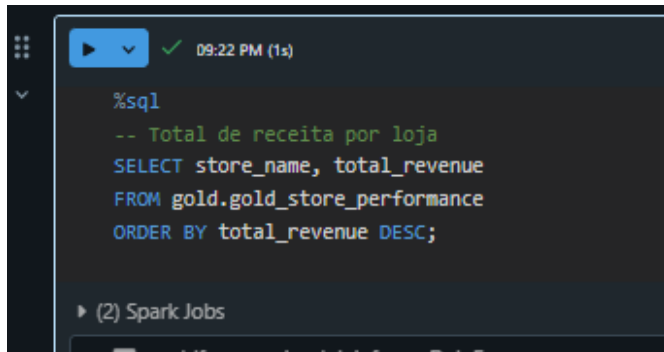
Métricas Calculadas:

- **Total de Pedidos (total_orders):** Quantidade total de pedidos realizados em cada loja.
- **Receita Total (total_revenue):** Receita bruta gerada por cada loja, calculada como $(list_price * quantity) - discount$.
- **Itens Vendidos (total_items_sold):** Quantidade total de itens vendidos por loja.
- **Estoque Restante (remaining_stock):** Soma dos itens ainda disponíveis no estoque para cada loja.

Processamento:

1. **Fonte de Dados:** Junção das tabelas **orders**, **order_items**, **stores** e **stocks**.
2. **Transformações Aplicadas:**
 - a. Renomeação da coluna **quantity** da tabela **stocks** para **stock_quantity**, evitando conflitos de nomes.
 - b. Agrupamento por **store_id** e **store_name**.
 - c. Cálculo das métricas agregadas listadas acima.
3. **Armazenamento:** Dados salvos como tabela Delta e registrados no catálogo Delta sob o esquema **gold**.

Exemplo de Query SQL:



```
%sql
-- Total de receita por loja
SELECT store_name, total_revenue
FROM gold.gold_store_performance
ORDER BY total_revenue DESC;
```

▶ (2) Spark Jobs

Transformações Gerais da Camada Gold

As tabelas da camada **Gold** são criadas a partir de dados já refinados da camada **Silver**. O objetivo é agregar os dados e calcular métricas relevantes para análises estratégicas.

1. Junções:

- As tabelas são integradas utilizando chaves primárias e estrangeiras para consolidar os dados.
- Por exemplo, a tabela `gold_clients` une `orders` e `order_items` pela coluna `order_id`.

2. Métricas Agregadas:

- São calculadas utilizando funções como `count`, `sum` e operações matemáticas complexas.

3. Renomeação de Colunas:

- Colunas duplicadas são renomeadas para evitar ambiguidades durante o processamento, como a coluna `quantity` da tabela `stocks`.

4. Armazenamento em Formato Delta:

- As tabelas são salvas em formato Delta para garantir:
 - Transações ACID.
 - Controle de versão.
 - Suporte a atualizações incrementais.

Conexão do Databricks com PowerBI

Com tabelas estruturadas, é possível expandir as análises e fazê-las de maneira mais assertiva no PowerBI.

Para isso, abri o PowerBI desktop e em Get Data, fiz a importação das tabelas.

Get Data

×

All

Azure

Online Services

All

Azure Databricks

Databricks

Databricks

Server Hostname ⓘ

HTTP Path ⓘ

Example: `sql/protocolv1/o/1814582234607533/7508-187377-agent704`

- **Ações:**
 - Configuração do conector no **PowerBI**:
 - **Server Hostname**: `community.cloud.databricks.com`
 - **HTTP Path**: Fornecido pelo cluster no Databricks.
 - Criação de tabelas intermediárias e conexão com o banco de dados via JDBC/ODBC.

Spark

JDBC/ODBC

Server Hostname

Port

Protocol

HTTP Path

- Exibição de tabelas no PowerBI para geração de relatórios.