

Primeiramente iniciamos com a criação de um notebook para configurarmos a criação do Database. Aqui já definimos os 3 bancos de dados necessários na arquitetura medalhão.

A arquitetura medalhão tem um papel fundamental para um engenheiro de dados. Com elas, todo o projeto fica organizado e sabe-se como os dados estão em cada camada para futuras manipulações e consultas conforme a estratégia de negócio.

Na camada BRONZE temos os dados “sujos”, aqueles dados que vem do jeito que estão, sem tratamento



The screenshot shows a Databricks SQL notebook cell. At the top, there is a status bar with a play button, a green checkmark, the date '14/11/2024 (3s)', and the cell number '1'. To the right of the status bar are icons for 'SQL', a full-screen icon, and a menu icon. The main area of the cell contains the following SQL code:

```
%sql

CREATE DATABASE IF NOT EXISTS spark_catalog.bronze
LOCATION 'dbfs/FilesStore/SoC/bronze/';
```

At the bottom of the cell is an input field with the text 'OK'.



The screenshot shows a Databricks SQL notebook cell. At the top, there is a status bar with a play button, a green checkmark, the date '14/11/2024 (<1s)', and the cell number '2'. To the right of the status bar are icons for 'SQL', a full-screen icon, and a menu icon. The main area of the cell contains the following SQL code:

```
%sql

CREATE DATABASE IF NOT EXISTS spark_catalog.silver
LOCATION 'dbfs/FilesStore/SoC/silver';
```

At the bottom of the cell is an input field with the text 'OK'.



The screenshot shows a Databricks SQL notebook cell. At the top, there is a status bar with a play button, a green checkmark, the date '14/11/2024 (<1s)', and the cell number '3'. To the right of the status bar are icons for 'SQL', a full-screen icon, and a menu icon. The main area of the cell contains the following SQL code:

```
%sql

CREATE DATABASE IF NOT EXISTS spark_catalog.gold
LOCATION 'dbfs/FilesStore/SoC/gold';
```

At the bottom of the cell is an input field with the text 'OK'.

Na camada BRONZE temos os dados que chamamos “crus”, “originais”, aqueles dados que vem do jeito que estão, sem tratamento algum, necessitando de assim fazê-los.

✓ 14/11/2024 (<1s)

1

```
#Aqui começamos a importar as bibliotecas que serão usadas na camada bronze do projeto.
```

```
from pyspark.sql.functions \
    import \
        current_date, \
        current_timestamp, \
        expr, \
        col, \
        split, \
        hour, \
        to_date, \
        substring, \
        trim
```

✓ 14/11/2024 (<1s)

2

Python



```
#nome do DataBase e tabela que serão criados.
```

```
database = 'bronze'
```

```
tabela = 'vendas'
```

```
#caminho do arquivo
```

```
path = 'dbfs:/FileStore/SoC/Vendas2020_2024.csv'
```

▶ ✓ 14/11/2024 (1s) 3

```
#Aqui criamos o Dataframe com base no arquivo que vamos processar no DataBricks. São colocados parâmetros de como daremos vida ao dataframe.
df = spark.read.format("csv") \
    .option("header", True) \
    .option("delimiter", ';') \
    .option("inferSchema", True) \
    .load(path)

# Capturando todas as linhas, exceto a primeira. Isso pode acontecer se vc tem uma linha que por causa do formato como vem no arquivo ela não é necessária.
df = df.exceptAll(df.limit(1))
```

▶ (2) jobs Spark

▶ df: pyspark.sql.dataframe.DataFrame = [Código: string, Descrição: string ... mais 6 campos]

Aqui já podemos iniciar a padronização dos nomes das colunas da tabela, e já temos previamente o s tipos de dados que o parâmetro “.option(“inferSchema”, True)” reconhece no DataFrame.

▶ ✓ 14/11/2024 (<1s) 4 Python

```
df = df.withColumnRenamed("Código", "ProdutoID")
df = df.withColumnRenamed("Descrição", "Descricao")
df = df.withColumnRenamed("Valor Unitário", "Valor_Unitario")
df = df.withColumnRenamed("Valor Total", "Valor_Total")
df = df.withColumnRenamed("Nota Fiscal", "Nota_Fiscal")
df = df.withColumnRenamed("Emissão", "Emissao")
df = df.withColumnRenamed("Unnamed: 8", "Unnamed")
```

▼ df: pyspark.sql.dataframe.DataFrame

ProdutoID: string  
Descricao: string  
Qtd: integer  
Valor\_Unitario: double  
Valor\_Total: double  
Nota\_Fiscal: integer  
Emissao: string  
Vendedor: integer

Aqui separamos alguns dados que vieram unificados numa mesma coluna, como no caso 'Data' e 'Hora'. Foi utilizado linhas de código Python para realizar essa ação. Abaixo pode ser observada cada parte do código.



```
# Separando a data e a hora da coluna "Emissao"
df = df.withColumn("Data_Emissao", trim(split(col("Emissao"), " ")[0])) \
    .withColumn("Hora_Emissao", trim(split(col("Emissao"), " ")[1]))

#-----

# Convertendo a coluna "Data_Emissao" para o tipo Date (com formato dd-MM-yyyy)
df = df.withColumn("Data_Emissao", to_date(col("Data_Emissao"), 'dd/MM/yyyy'))

#-----

# Removendo a coluna antiga "Emissao"
df = df.drop("Emissao")
```


df: pyspark.sql.dataframe.DataFrame = [ProdutoID: string, Descricao: string ... mais 7 campos]

▶ ✓ 14/11/2024 (<1s)

6

#Aqui é atribuído as funções que capturam a data da carga e data e hora da carga para um melhor controle de versões e quando foi feito tal processamento.

```
df = df.withColumn("data_carga", current_date())
df = df.withColumn("data_hora_carga", expr("current_timestamp() - INTERVAL 3 HOURS"))
```

▶  df: pyspark.sql.dataframe.DataFrame = [ProdutoID: string, Descricao: string ... mais 9 campos]

▶ ✓ 14/11/2024 (14s)

7

```
df.write\
    .format('delta') \
    .mode('overwrite') \
    .option('mergeSchema', 'true') \
    .option('overwriteSchema', 'true') \
    .saveAsTable(f'{database}.{tabela}')
print("Dados gravados com sucesso!")
```

▶ (11) jobs Spark

Dados gravados com sucesso!