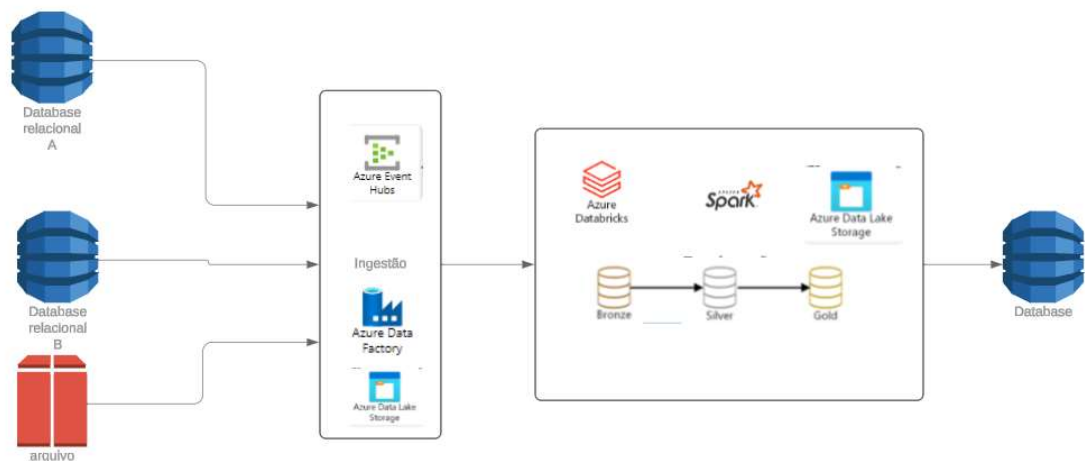


1. Arquitetura e Configuração: Descreva a arquitetura geral de uma implementação típica do Databricks. Como você configuraria um ambiente Databricks para otimizar o desempenho e a escalabilidade?

R: Para uma implementação do Databricks é necessário ter um Servidor ou um Cluster com a plataforma instalada, com as conexões de origem e destino configurada e habilitada. A partir daí é possível criar notebooks para extração, transformação e carga de dados a partir das origens que pode ser diversas e para o destino final que pode ser diverso também. As origens pode ser um Blob Storage, databases ou até mesmo arquivos. O destino também pode ser um Blob Storage entre as camadas (bronze e silver), que é mais performático e, depois na camada ouro, um banco de dados relacional ou não relacional.

Uma possível solução para que se tenha desempenho e escalabilidade é configurar o Spark e trabalhar com Blobs entre as camadas. Além disso é possível configurar o Scala para poder realizar cargas nos Databases de destino com maior performance. Por exemplo ao fazer um merge em um banco de dados relacional.

Abaixo um exemplo de desenho de uma solução.



2. Apache Spark e Databricks: Como o Databricks se integra ao Apache Spark? Quais são as principais vantagens do uso do Databricks em comparação com uma instalação padrão do Apache Spark?

R: Integração do Databricks com Spark: O Databricks foi construído em cima do Apache Spark e utiliza clusters para o processamento de grandes volumes de dados de forma distribuída. A plataforma também possui recursos para construção de pipelines, permitindo que os usuários processem dados em etapas e visualizem os resultados em tempo real.

Vantagens do Databricks. Na plataforma do Databricks dentro do notebook podemos trabalhar com várias linguagens de programação como por exemplo o Scala, R, SQL, etc., em apenas um notebook você pode misturar as linguagens. Além dos armazenamentos internos como os DBFS.

3. Notebooks e Linguagens de Programação: Explique como você usaria Notebooks no Databricks para criar e executar código. Além disso, como o Databricks suporta várias linguagens de programação, e como você decidiria qual linguagem usar em um projeto específico?

R: Para o uso de notebooks, o usuário precisa entrar na plataforma do databricks, pode ser inclusive a Community, dentro da área de Data Science & Engineering no menu lateral esquerdo na opção "Create -> Notebook". Os notebooks podem ser usados para criar um pipeline de ETL (Extract Transform e Load), usando as linguagens que o databricks suporte. Entre elas o python. Por prática comum, é comum, para criar um pipeline de ETL deve-se criar 3 notebooks um para extração dos dados como é na origem, chamado de camada Bronze, outro para a tratamento dos dados, chamado de camada Silver e outro para fazer agregações entre outras transformações, chamado de camada Gold. A escolha da linguagem depende do volume de dados que o usuário está trabalhando. Para ganhar poder de processamento é comum utilizar os comandos em Spark onde o processamento ocorre de forma distribuída. Além disso a escolha também pode ser com base na afinidade do usuário.

4. Integração de Fontes de Dados: Como o Databricks facilita a integração com diferentes fontes de dados, como Data Lakes, bancos de dados relacionais e fontes externas? Você pode fornecer um exemplo prático de como lidar com essas integrações?

R: O Databricks possui diversos padrões que possibilitam integração com diversas fontes de dados, entre elas conexão com fontes externas de dados, fazendo uso de API Rest fornecendo um endpoint unificado para gerenciar a plataforma. Além disso conectores otimizados que possibilitam a ingestão de dados no Data Lake, por exemplo JDBC entre outros. Muitos dos bancos de dados já tem seu run-time incluído no Databricks Runtime. Como exemplo posso citar a integração com um Storage da Azure por exemplo, onde é possível ler arquivos de diversas tipos, JSON, CSV, etc. Para isso basta montar o drive com a chave de acesso ao storage, no notebook e fazer a leitura e gravação dos arquivos.

5. Machine Learning no Databricks: Descreva a abordagem que você seguiria para desenvolver e treinar modelos de machine learning no Databricks. Quais são as principais ferramentas e bibliotecas que você usaria para esse fim?

R: A plataforma disponibiliza um área específica para Machine Learning, que inclusive o nome é Machine Learning. Nessa área é possível criar notebooks também, para ler as bases do Databricks DBFS ou arquivos no storage. Dentro dos notebooks é possível criar modelos de machine learning, por exemplo rede neural etc. Além disso é possível criar painéis e visualização de dados. O Databricks Runtime for Machine Learning é o responsável por tudo isso, com clusters que integram versões compatíveis das bibliotecas mais comuns, como TensorFlow, PyTorch e Keras, também incluem suporte de GPU pré-configurado com drivers e biblioteca de suporte.

6. Segurança e Controle de Acesso: Como o Databricks aborda questões de segurança e controle de acesso? Quais são as práticas recomendadas para garantir a proteção dos dados e ambientes de desenvolvimento?

R: A plataforma possui controles de acessos que garante que cada usuário só tenha acesso ao que ele precisa dentro da plataforma. Inclusive é possível o trabalho colaborativo, por exemplo ao mesmo tempo que um usuário está criando pipeline de dados outros usuários com perfil de cientista de dados por exemplo, podem acessar os dados já carregados. O acesso pode ser dados por grupo de usuários ou usuários individuais. Além disso é compatível com as clouds mais comuns (Azure, AWS e GCP).

7. Desafios e Soluções: Pergunta: Conte-nos sobre um desafio específico que você enfrentou ao trabalhar com Databricks e como o resolveu. Qual foi a solução implementada e quais foram os resultados alcançados?

R: Em um projeto que trabalhei, tive que criar um ETL para carregar dados de uma fonte de dados que era muito grande, com bilhões de registros BigQuery da GCP). O pipeline estava se perdendo com a volumetria e demorando muito. Então fiz uma implementação usando o Airflow e o python, lendo os registros de forma quebrada (particionada). Com isso resolveu meu problema que tive e a carga ficou mais rápida.

8. Conjunto de Dados do Kaggle: Escolha um conjunto de dados do Kaggle relacionado a vendas. Certifique-se de que o conjunto de dados inclui informações como datas, produtos, quantidades vendidas, etc.

Projeto de Engenharia de Dados:

Ingestão e Carregamento de Dados: Carregue o conjunto de dados no Databricks.

Explore o esquema dos dados e faça ajustes conforme necessário.

Transformações de Dados: Realize transformações necessárias, como tratamento de valores nulos, conversões de tipos, etc. Adicione uma coluna calculada, por exemplo, o valor total de cada transação. Agregue os dados para obter estatísticas de vendas, por exemplo, o total de vendas por produto ou por categoria. Introduza uma regra mais complexa, como identificar padrões de comportamento de compra ao longo do tempo ou criar categorias personalizadas de produtos com base em determinados critérios.

Saída em Parquet e Delta: Grave os dados transformados e agregados em um formato Parquet para persistência eficiente. Grave os mesmos dados em formato Delta Lake para aproveitar as funcionalidades de versionamento e transações ACID.

Exploração Adicional (Opcional): Execute consultas exploratórias para entender melhor os dados e validar as transformações. Crie visualizações ou relatórios para comunicar insights. Agende o notebook para execução automática em intervalos regulares para garantir a atualização contínua dos dados

SOLUÇÃO

Base de dados utilizada: <https://www.kaggle.com/datasets/kyanyoga/sample-sales-data>

Notebook 1 (Camada Bronze)

Comandos:

```
from datetime import datetime
source_origin_param = "dbfs:/FileStore/FileStore/"
dest_origin_param =
"dbfs:/FileStore/FileStore/Vendas_Bronze/extracted_at=%s" %
datetime.now().strftime('%Y-%m-%d')
extracted_at_param = datetime.now().strftime('%Y-%m-%d')

RAW_PATH = "{}{}".format(source_origin_param, 'sales_data_sample.csv')
BRONZE_PATH = "{}/{ {}".format(dest_origin_param, 'sales_data_sample_bronze')

print(RAW_PATH)
```

```

print(BRONZE_PATH)
dfVendas = spark.read.format("csv").option("header", "true").load(RAW_PATH)
dfVendas.display()
PARQUET_PATH = BRONZE_PATH + '.parquet'
dfVendas.write.format("parquet").mode("overwrite").save(PARQUET_PATH)
DELTA_PATH = BRONZE_PATH + '.delta'
dfVendas.write.format("delta").mode("overwrite").save(DELTA_PATH)

```

Telas da execução:

001_Bronze Python ☆

File Edit View Run Help Last edit was 17 minutes ago Provide feedback

Run all Cluster_Aprendizado Share Publish

Cmd 1

```
1 from datetime import datetime
```

Command took 0.86 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 11:39:55 on Cluster_Aprendizado

Cmd 2

```

1 now = datetime.now()
2 source_origin_param = "dbfs:/FileStore/FileStore/"
3 dest_origin_param = "dbfs:/FileStore/FileStore/Vendas_Bronze/extracted_at=%s" % datetime.now().strftime('%Y-%m-%d')
4 extracted_at_param = datetime.now().strftime('%Y-%m-%d')
5
6 RAW_PATH = "{}{}".format(source_origin_param, 'sales_data_sample.csv')
7 BRONZE_PATH = "{}{}".format(dest_origin_param, 'sales_data_sample_bronze')
8
9 print(RAW_PATH)
10 print(BRONZE_PATH)

```

dbfs:/FileStore/FileStore/sales_data_sample.csv
dbfs:/FileStore/FileStore/Vendas_Bronze/extracted_at=2024-01-29/sales_data_sample_bronze

Command took 0.88 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 11:39:16 on Cluster_Aprendizado

Cmd 3

```

1 dfVendas = spark.read.format("csv").option("header", "true").load(RAW_PATH)
2 dfVendas.display()

```

(2) Spark Jobs

dfVendas: pyspark.sql.dataframe.DataFrame = [ORDERNUMBER: string, QUANTITYORDERED: string ... 23 more fields]

Table

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE
1	10107	30	95.7	2	2871	2/24/2003 0:00	Shipped	1	2	2003	Motorcycles
2	10121	34	81.35	5	2765.9	5/7/2003 0:00	Shipped	2	5	2003	Motorcycles

Table

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE
1	10107	30	95.7	2	2871	2/24/2003 0:00	Shipped	1	2	2003	Motorcycles
2	10121	34	81.35	5	2765.9	5/7/2003 0:00	Shipped	2	5	2003	Motorcycles
3	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	Motorcycles
4	10145	45	83.26	6	3746.7	8/25/2003 0:00	Shipped	3	8	2003	Motorcycles
5	10159	49	100	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	Motorcycles
6	10168	36	96.66	1	3479.76	10/28/2003 0:00	Shipped	4	10	2003	Motorcycles
7	10180	29	86.13	9	2497.77	11/11/2003 0:00	Shipped	4	11	2003	Motorcycles

2,823 rows | 2.58 seconds runtime Refreshed 4 minutes ago

Command took 2.58 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 11:39:21 on Cluster_Aprendizado

Cmd 4

```

1 PARQUET_PATH = BRONZE_PATH + '.parquet'
2 dfVendas.write.format("parquet").mode("overwrite").save(PARQUET_PATH)

```

(1) Spark Jobs

Command took 1.87 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 11:41:03 on Cluster_Aprendizado

Cmd 5

```

1 DELTA_PATH = BRONZE_PATH + '.delta'
2 dfVendas.write.format("delta").mode("overwrite").save(DELTA_PATH)

```

Notebook 2 (Camada Silver)

Comandos:

```

from datetime import datetime
from pyspark.sql.functions import lit

```

```

source_origin_param =
"dbfs:/FileStore/FileStore/Vendas_Bronze/extracted_at=%s" %
datetime.now().strftime('%Y-%m-%d')

```

```

dest_origin_param    =
"dbfs:/FileStore/FileStore/Vendas_Silver/extracted_at=%s" %
datetime.now().strftime('%Y-%m-%d')
extracted_at_param   =  datetime.now().strftime('%Y-%m-%d')

BRONZE_PATH = "{}/{ {}".format(source_origin_param, 'sales_data_sample_bronze')
SILVER_PATH = "{}/{ {}".format(dest_origin_param, 'sales_data_sample_silver')

print(BRONZE_PATH)
print(SILVER_PATH)
BRONZE_PATH_DELTA = BRONZE_PATH + ".delta"
print(BRONZE_PATH_DELTA)
dfVendas =
spark.read.option("fetchsize","100000").format("delta").load(BRONZE_PATH_DELTA)
dfVendas.display()

now = datetime.now().strftime('%Y-%m-%d')
dfVendas_Silver = dfVendas
dfVendas_Silver = dfVendas_Silver \
    .withColumn('ORDERDATE',dfVendas_Silver['ORDERDATE'].cast('date')) \
    .withColumn('ORDERNUMBER',dfVendas_Silver['ORDERNUMBER'].cast('int')) \
    .withColumn('QTR_ID',dfVendas_Silver['QTR_ID'].cast('int')) \
    .withColumn('MONTH_ID',dfVendas_Silver['MONTH_ID'].cast('int')) \
    .withColumn('YEAR_ID',dfVendas_Silver['YEAR_ID'].cast('int')) \
    .withColumn('MSRP',dfVendas_Silver['MSRP'].cast('int')) \
    .withColumn('ORDERLINENUMBER',dfVendas_Silver['ORDERLINENUMBER'].cast('int')) \
    .withColumn('QUANTITYORDERED',dfVendas_Silver['QUANTITYORDERED'].cast('int')) \
    .withColumn('SALES',dfVendas_Silver['SALES'].cast('float')) \
    .withColumn('PRICEEACH',dfVendas_Silver['PRICEEACH'].cast('float')) \
    .withColumn("DATA_CARGA",lit(now).cast('date'))
dfVendas_Silver.printSchema()

SILVER_PATH = SILVER_PATH + '.delta'
dfVendas_Silver.write.format("delta").mode("overwrite").save(SILVER_PATH)

```

TELAS:

002_Silver Python

File Edit View Run Help Last edit was 21 minutes ago Provide feedback

```
Cmd 1

1 from datetime import datetime
2 source_origin_param = "dbfs:/FileStore/FileStore/Vendas_Bronze/extracted_at=%s" % datetime.now().strftime('%Y-%m-%d')
3 dest_origin_param = "dbfs:/FileStore/FileStore/Vendas_Silver/extracted_at=%s" % datetime.now().strftime('%Y-%m-%d')
4 extracted_at_param = datetime.now().strftime('%Y-%m-%d')
5
6 BRONZE_PATH = "{}/{}/{}".format(source_origin_param, 'sales_data_sample_bronze')
7 SILVER_PATH = "{}/{}/{}".format(dest_origin_param, 'sales_data_sample_silver')
8
9 print(BRONZE_PATH)
10 print(SILVER_PATH)

dbfs:/FileStore/FileStore/Vendas_Bronze/extracted_at=2024-01-29/sales_data_sample_bronze
dbfs:/FileStore/FileStore/Vendas_Silver/extracted_at=2024-01-29/sales_data_sample_silver

Command took 0.13 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 14:35:21 on Cluster_Aprendizado
```

```
Cmd 2

1 BRONZE_PATH_DELTA = BRONZE_PATH + ".delta"
2 print(BRONZE_PATH_DELTA)
3 dfVendas = spark.read.option("fetchsize", "100000").format("delta").load(BRONZE_PATH_DELTA)
4 dfVendas.display()

▶ (1) Spark Jobs

▶ dfVendas: pyspark.sql.dataframe.DataFrame = [ORDERNUMBER: string, QUANTITYORDERED: string ... 23 more fields]

dbfs:/FileStore/FileStore/Vendas_Bronze/extracted_at=2024-01-29/sales_data_sample_bronze.delta
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE
1	10107	30	95.7	2	2871	2/24/2003 0:00	Shipped	1	2	2003	Motorcycles
2	10121	34	81.35	5	2765.9	5/7/2003 0:00	Shipped	2	5	2003	Motorcycles
3	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	Motorcycles
4	10145	45	83.26	6	3746.7	8/25/2003 0:00	Shipped	3	8	2003	Motorcycles
5	10159	49	100	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	Motorcycles
6	10168	36	96.66	1	3479.76	10/28/2003 0:00	Shipped	4	10	2003	Motorcycles
7	10180	29	86.13	9	2497.77	11/11/2003 0:00	Shipped	4	11	2003	Motorcycles

2,823 rows | 1.58 seconds runtime Refreshed 17 minutes ago

Command took 1.58 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 14:40:20 on Cluster_Aprendizado

```
Cmd 3

1 now = datetime.now().strftime('%Y-%m-%d')
2 dfVendas_Silver = dfVendas
3 dfVendas_Silver = dfVendas_Silver \
4     .withColumn('ORDERDATE', dfVendas_Silver['ORDERDATE'].cast('date')) \
5     .withColumn('ORDERNUMBER', dfVendas_Silver['ORDERNUMBER'].cast('int')) \
6     .withColumn('QTR_ID', dfVendas_Silver['QTR_ID'].cast('int')) \
7     .withColumn('MONTH_ID', dfVendas_Silver['MONTH_ID'].cast('int')) \
8     .withColumn('YEAR_ID', dfVendas_Silver['YEAR_ID'].cast('int')) \
9     .withColumn('MSRP', dfVendas_Silver['MSRP'].cast('int')) \
10    .withColumn('ORDERLINENUMBER', dfVendas_Silver['ORDERLINENUMBER'].cast('int')) \
11    .withColumn('QUANTITYORDERED', dfVendas_Silver['QUANTITYORDERED'].cast('int')) \
12    .withColumn('SALES', dfVendas_Silver['SALES'].cast('float')) \
13    .withColumn('PRICEEACH', dfVendas_Silver['PRICEEACH'].cast('float')) \
14    .withColumn("DATA_CARGA", lit(now).cast('date'))
15 dfVendas_Silver.printSchema()

▶ dfVendas_Silver: pyspark.sql.dataframe.DataFrame = [ORDERNUMBER: integer, QUANTITYORDERED: integer ... 24 more fields]

|-- STATUS: string (nullable = true)
|-- QTR_ID: integer (nullable = true)
|-- MONTH_ID: integer (nullable = true)
|-- YEAR_ID: integer (nullable = true)
|-- PRODUCTLINE: string (nullable = true)
|-- MSRP: integer (nullable = true)
|-- PRODUCTCODE: string (nullable = true)
|-- CUSTOMERNAME: string (nullable = true)
|-- PHONE: string (nullable = true)
|-- ADDRESSLINE1: string (nullable = true)
|-- ADDRESSLINE2: string (nullable = true)
|-- CITY: string (nullable = true)
|-- STATE: string (nullable = true)
```

The screenshot shows a Databricks notebook interface. At the top, the notebook is titled '002_Silver' with a 'Python' language selector and a star icon. Below the title is a menu bar with 'File', 'Edit', 'View', 'Run', and 'Help'. A status bar indicates 'Last edit was 22 minutes ago' and a 'Provide feedback' link. The main editor area contains a schema definition for a table, listing various fields with their data types and nullability. Below the schema, a command execution bar shows the command took 0.44 seconds. The command log area, labeled 'Cmd 4', shows two lines of Python code: setting a path and saving a DataFrame. Below the command log, it indicates '(6) Spark Jobs' and shows the command took 8.69 seconds.

```
-- STATUS: string (nullable = true)
-- QTR_ID: integer (nullable = true)
-- MONTH_ID: integer (nullable = true)
-- YEAR_ID: integer (nullable = true)
-- PRODUCTLINE: string (nullable = true)
-- MSRP: integer (nullable = true)
-- PRODUCTCODE: string (nullable = true)
-- CUSTOMERNAME: string (nullable = true)
-- PHONE: string (nullable = true)
-- ADDRESSLINE1: string (nullable = true)
-- ADDRESSLINE2: string (nullable = true)
-- CITY: string (nullable = true)
-- STATE: string (nullable = true)
-- POSTALCODE: string (nullable = true)
-- COUNTRY: string (nullable = true)
-- TERRITORY: string (nullable = true)
-- CONTACTLASTNAME: string (nullable = true)
-- CONTACTFIRSTNAME: string (nullable = true)
-- DEALSIZE: string (nullable = true)
-- DATA_CARGA: date (nullable = true)

Command took 0.44 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 14:53:46 on Cluster_Aprendizado

Cmd 4

1  SILVER_PATH = SILVER_PATH + '.delta'
2  dfVendas_Silver.write.format("delta").mode("overwrite").save(SILVER_PATH)

▶ (6) Spark Jobs

Command took 8.69 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 14:56:34 on Cluster_Aprendizado
```

Notebook 3 (Camada Gold)

Comandos:

```
from datetime import datetime
from pyspark.sql.functions import lit

source_origin_param =
"dbfs:/FileStore/FileStore/Vendas_Silver/extracted_at=%s" %
datetime.now().strftime('%Y-%m-%d')
dest_origin_param   = "dbfs:/FileStore/FileStore/Vendas_Gold/extracted_at=%s"
% datetime.now().strftime('%Y-%m-%d')
extracted_at_param  = datetime.now().strftime('%Y-%m-%d')

SILVER_PATH = "{}/{ {}".format(source_origin_param, 'sales_data_sample_silver')
GOLD_PATH   = "{}/{ {}".format(dest_origin_param, 'sales_data_sample_gold')

print(SILVER_PATH)
print(GOLD_PATH)
SILVER_PATH_DELTA = SILVER_PATH + ".delta"
print(SILVER_PATH_DELTA)
```



```

dfVendas =
spark.read.option("fetchsize","100000").format("delta").load(SILVER_PATH_DELTA)
dfVendas.display()
dfVendas_Gold = dfVendas
dfVendas_Gold = dfVendas_Gold \
    .withColumn("TOTAL_PRODUTOS", dfVendas_Gold.QUANTITYORDERED *
dfVendas_Gold.PRICEEACH)

```

#gravando a saída GOLD

```

PARQUET_PATH = GOLD_PATH + '.parquet'
dfVendas_Gold.write.format("parquet").mode("overwrite").save(PARQUET_PATH)

```

```

GOLD_PATH = GOLD_PATH + '.delta'
dfVendas_Gold.write.format("delta").mode("overwrite").option("mergeSchema",
"true").partitionBy("YEAR_ID").save(GOLD_PATH)

```

```

from pyspark.sql.functions import sum, avg

```

Soma

```

dfVendas_Gold.select(sum("TOTAL_PRODUTOS")).show()

```

#Agrupamento

```

dfVendas_Gold.groupBy("PRODUCTLINE").agg(avg("SALES")).show()

```

TELAS:

```

databricks

003_Gold Python ☆
File Edit View Run Help Last edit was 11 minutes ago Provide feedback

Cmd 1

1 from datetime import datetime
2 from pyspark.sql.functions import lit
3
4 source_origin_param = "dbfs://FileStore/FileStore/Vendas_Silver/extracted_at=%s" % datetime.now().strftime('%Y-%m-%d')
5 dest_origin_param = "dbfs://FileStore/FileStore/Vendas_Gold/extracted_at=%s" % datetime.now().strftime('%Y-%m-%d')
6 extracted_at_param = datetime.now().strftime('%Y-%m-%d')
7
8 SILVER_PATH = "{}/{}".format(source_origin_param, 'sales_data_sample_silver')
9 GOLD_PATH = "{}/{}".format(dest_origin_param, 'sales_data_sample_gold')
10
11 print(SILVER_PATH)
12 print(GOLD_PATH)

dbfs://FileStore/FileStore/Vendas_Silver/extracted_at=2024-01-29/sales_data_sample_silver
dbfs://FileStore/FileStore/Vendas_Gold/extracted_at=2024-01-29/sales_data_sample_gold
Command took 0.10 seconds -- by marceloacarlos@gmail.com at 29/01/2024, 15:01:14 on Cluster_Aprendizado

Cmd 2

1 SILVER_PATH_DELTA = SILVER_PATH + ".delta"
2 print(SILVER_PATH_DELTA)
3 dfVendas = spark.read.option("fetchsize","100000").format("delta").load(SILVER_PATH_DELTA)
4 dfVendas.display()

(2) Spark Jobs
dfVendas: pyspark.sql.dataframe.DataFrame = [ORDERNUMBER: integer, QUANTITYORDERED: integer ... 24 more fields]
dbfs://FileStore/FileStore/Vendas_Silver/extracted_at=2024-01-29/sales_data_sample_silver.delta

```


003_Gold

Python

☆

File

Edit

View

Run

Help

Last edit was 12 minutes ago

Provide feedback

▶ Run all

● Cluster_J

Table

+

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID
1	10103	26	100	11	5404.62	null	Shipped	1	1	2003
2	10112	29	100	1	7209.11	null	Shipped	1	3	2003
3	10126	38	100	11	7329.06	null	Shipped	2	5	2003
4	10140	37	100	11	7374.1	null	Shipped	3	7	2003
5	10150	45	100	8	10993.5	null	Shipped	3	9	2003
6	10163	21	100	1	4860.24	null	Shipped	4	10	2003
7	10174	34	100	4	8014.82	null	Shipped	4	11	2003

2,823 rows | 2.50 seconds runtime

Command took 2.50 seconds -- by marceloscarlos@gmail.com at 29/01/2024, 15:03:32 on Cluster_Aprendizado

Cmd 3

```
1 dfVendas_Gold = dfVendas
2 dfVendas_Gold = dfVendas_Gold \
3   .withColumn("TOTAL_PRODUTOS", dfVendas_Gold.QUANTITYORDERED * dfVendas_Gold.PRICEEACH)
```

dfVendas_Gold: pyspark.sql.dataframe.DataFrame = [ORDERNUMBER: integer, QUANTITYORDERED: integer ... 25 more fields]

Command took 0.13 seconds -- by marceloscarlos@gmail.com at 29/01/2024, 15:09:00 on Cluster_Aprendizado

Cmd 4

```
1 #gravando a saída GOLD
2 PARQUET_PATH = GOLD_PATH + '.parquet'
```

003_Gold

Python

☆

File

Edit

View

Run

Help

Last edit was 13 minutes ago

Provide feedback

▶ Run all

● Cluster_J

```
1 #gravando a saída GOLD
2 PARQUET_PATH = GOLD_PATH + '.parquet'
3 dfVendas_Gold.write.format("parquet").mode("overwrite").save(PARQUET_PATH)
4
5 GOLD_PATH = GOLD_PATH + '.delta'
6 dfVendas_Gold.write.format("delta").mode("overwrite").option("mergeSchema", "true").partitionBy("YEAR_ID").save(GOLD_PATH)
```

(7) Spark Jobs

Command took 10.39 seconds -- by marceloscarlos@gmail.com at 29/01/2024, 15:15:34 on Cluster_Aprendizado

Cmd 5

```
1 from pyspark.sql.functions import sum, avg
2
3 # Soma
4 dfVendas_Gold.select(sum("TOTAL_PRODUTOS")).show()
5
6 #Agrupamento
7 dfVendas_Gold.groupBy("PRODUCTLINE").agg(avg("SALES")).show()
```

(4) Spark Jobs

+-----+
|sum(TOTAL_PRODUTOS)|
+-----+
| 8290886.785461426|
+-----+

```

1  from pyspark.sql.functions import sum, avg
2
3  # Soma
4  dfVendas_Gold.select(sum("TOTAL_PRODUTOS")).show()
5
6  #Agrupamento
7  dfVendas_Gold.groupBy("PRODUCTLINE").agg(avg("SALES")).show()

```

► (4) Spark Jobs

```

+-----+
| sum(TOTAL_PRODUTOS) |
+-----+
| 8290886.785461426 |
+-----+

```

```

+-----+-----+
| PRODUCTLINE | avg(SALES) |
+-----+-----+
| Motorcycles | 3523.831840584285 |
| Vintage Cars | 3135.3391030828484 |
| Ships | 3053.150128747663 |
| Trucks and Buses | 3746.810110504049 |
| Classic Cars | 4053.3771052395055 |
| Trains | 2938.2268700537743 |
| Planes | 3186.286180882672 |
+-----+-----+

```