DLT Exercises:

Used Google Colab:

Exercise 1: Creating ETL pipeline using DLT

```
#  Exercise 1: Creating a Complete ETL Pipeline using Delta Live Tables

# DLT in python

import dlt
from pyspark.sql.functions import col

@dlt.table
def raw_transactions():
    return (
        spark.read.format("csv")
        .option("header", "true")
        .load("/dbfs/FileStore/transactions.csv")
    )

@dlt.table
def transformed_transactions():
    return (
        dlt.read("raw_transactions")
        .withColumn("TotalAmount", col("Quantity") * col("Price"))
    )

@dlt.table
def final_transactions():
    return dlt.read("transformed_transactions")
```

```
# DLT in SQL

CREATE OR REPLACE LIVE TABLE raw_transactions AS
SELECT *
FROM read_csv('/dbfs/FileStore/transactions.csv');

CREATE OR REPLACE LIVE TABLE transformed_transactions AS
SELECT *, Quantity * Price AS TotalAmount
FROM raw_transactions;

CREATE OR REPLACE LIVE TABLE final_transactions AS
SELECT *
FROM transformed_transactions;
```

Exercise 2: Delta Lake Operations:

```
#  Exercise 2: Delta Lake Operations - Read, Write, Update, Delete, Merge
# Task 1 Read Data from Delta lake
# python
delta_df = spark.read.format("delta").load("/delta/final_transactions")

delta_df.show(5)

# sql
spark.sql("""SELECT *
FROM delta.`/delta/final_transactions`
""")
```

```
# Task 2 Write data to delta

new_transactions = [
    (6, '2024-09-06', 'C005', 'Keyboard', 4, 100),
    (7, '2024-09-07', 'C006', 'Mouse', 10, 20)
]

new_transactions_df = spark.createDataFrame(new_transactions,
    schema=["TransactionID", "TransactionDate", "CustomerID", "Product",
"Quantity", "Price"])

new_transactions_df.write.format("delta").mode("append").save("/delta/final_transac
tions")
```

```
# Task 3 Update data in delta
# python
from pyspark.sql.functions import col

updated_df = delta_df.withColumn(
    "Price",
    when(col("Product") == "Laptop", 1300).otherwise(col("Price"))
)

updated_df.write.format("delta").mode("overwrite").save("/delta/final_transactions"
)

# sql
spark.sql("""UPDATE delta.`/delta/final_transactions`
SET Price = 1300
WHERE Product = 'Laptop';""")
```

```
# Task 4 Delete from delta
# python

delta_df = delta_df.filter(col("Quantity") >= 3)

delta_df.write.format("delta").mode("overwrite").save("/delta/final_transactions")

#sql
spark.sql("""DELETE FROM delta.`/delta/final_transactions`
WHERE Quantity < 3;""")
```

```
# Task 5 Merge data into delta
spark.sql("""
MERGE INTO delta.`/delta/final_transactions` AS existing
USING (
    SELECT * FROM VALUES
    (1, '2024-09-01', 'C001', 'Laptop', 1, 1250),
    (8, '2024-09-08', 'C007', 'Charger', 2, 30)
) AS updates (TransactionID, TransactionDate, CustomerID, Product, Quantity, Price)
ON existing.TransactionID = updates.TransactionID
WHEN MATCHED THEN
    UPDATE SET
        existing.TransactionDate = updates.TransactionDate,
        existing.CustomerID = updates.CustomerID,
        existing.Product = updates.Product,
        existing.Quantity = updates.Quantity,
        existing.Price = updates.Price
WHEN NOT MATCHED THEN
    INSERT (TransactionID, TransactionDate, CustomerID, Product, Quantity, Price)
    VALUES (updates.TransactionID, updates.TransactionDate, updates.CustomerID,
updates.Product, updates.Quantity, updates.Price);
""")
```

Exercise 3: History, Time Travel, Vacuum

```
# Exercise 3 History, time travel, vacuum

# Task 1 history of the delta table
history_df = spark.sql("DESCRIBE HISTORY delta.`/delta/final_transactions`")
history_df.show()

spark.sql("DESCRIBE HISTORY delta.`/delta/final_transactions`;")
```

```
# Task 2 Time travel
```

```
time_travel_df = spark.read.format("delta").option("versionAsOf",
5).load("/delta/final_transactions")
time_travel_df.show()

spark.sql("SELECT * FROM delta.`/delta/final_transactions` VERSION AS OF 5;")

# using timestamp

spark.sql("SELECT * FROM delta.`/delta/final_transactions` TIMESTAMP AS OF '2024-
09-06 00:00:00';")
```

```
# Task 3 Vacuum

spark.sql("VACUUM delta.`/delta/final_transactions` RETAIN 168 HOURS")
```

```
# Task 4 Parquet to delta

csv_data = spark.read.format("csv").option("header",
"true").load("/dbfs/FileStore/transactions.csv")
csv_data.write.format("parquet").mode("overwrite").save("/dbfs/FileStore/transactio
ns_parquet")

parquet_data =
spark.read.format("parquet").load("/dbfs/FileStore/transactions_parquet")
parquet_data.write.format("delta").mode("overwrite").save("/delta/transactions_delt
a")
```

Exercise 4: Implementing Incremental Load Pattern

```
#  Exercise 4: Implementing Incremental Load Pattern using Delta Lake

# Task 1 Setup inital data

initial_data = [
    (1, "2024-09-01", "C001", "Laptop", 1, 1200),
    (2, "2024-09-02", "C002", "Tablet", 2, 300),
    (3, "2024-09-03", "C001", "Headphones", 5, 50)
]

columns = ["TransactionID", "TransactionDate", "CustomerID", "Product", "Quantity",
"Price"]

initial_df = spark.createDataFrame(initial_data, columns)

initial_df.write.format("delta").mode("overwrite").save("/delta/transactions")
```

```
# Task 2 Setup incremental data
incremental_data = [
    (4, "2024-09-04", "C003", "Smartphone", 1, 800),
    (5, "2024-09-05", "C004", "Smartwatch", 3, 200),
    (6, "2024-09-06", "C005", "Keyboard", 4, 100),
    (7, "2024-09-07", "C006", "Mouse", 10, 20)
]

incremental_df = spark.createDataFrame(incremental_data, columns)
```

```
# Task 3 Implement Incremental load

# transactions after 2024-09-03
incremental_load_df = incremental_df.filter(incremental_df.TransactionDate > "2024-
09-03")

incremental_load_df.write.format("delta").mode("append").save("/delta/transactions"
)

full_data_df = spark.read.format("delta").load("/delta/transactions")
full_data_df.show()

history_df = spark.sql("DESCRIBE HISTORY delta.`/delta/transactions`")
history_df.show()
```