**Exercise 1: Creating a Complete ETL Pipeline using Delta Live Tables (DLT)**

Task 1: Create Delta Live Table (DLT) Pipeline:

TransactionID,TransactionDate,CustomerID,Product,Quantity,Price

1,2024-09-01,C001,Laptop,1,1200

2,2024-09-02,C002,Tablet,2,300

3,2024-09-03,C001,Headphones,5,50

4,2024-09-04,C003,Smartphone,1,800

5,2024-09-05,C004,Smartwatch,3,200

**Step 1: Ingest Raw Data from CSV Files**

*Python Implementation:*

import dlt

@dlt.table

def raw\_transactions():

return dlt.read\_csv("/content/transactions.csv")

*SQL Implementation:*

CREATE OR REFRESH LIVE TABLE raw\_transactions

AS

SELECT \* FROM read\_csv('/content/transactions.csv');

**Step 2: Apply Transformations**

*Python Implementation:*

Transform the data to calculate total transaction amount.

@dlt.table

def transformed\_transactions():

return (

dlt.read("raw\_transactions")

.select(

"TransactionID",

"TransactionDate",

"CustomerID",

"Product",

"Quantity",

"Price",

(dlt.col("Quantity") \* dlt.col("Price")).alias("TotalAmount")

)

)

*SQL Implementation:*

CREATE OR REFRESH LIVE TABLE transformed\_transactions

AS

SELECT

TransactionID,

TransactionDate,

CustomerID,

Product,

Quantity,

Price,

(Quantity \* Price) AS TotalAmount

FROM

raw\_transactions;

**Step 3: Write the Final Data into a Delta Table**

*Python Implementation:*

@dlt.table

def final\_transactions():

return dlt.read("transformed\_transactions")

*SQL Implementation:*

CREATE OR REFRESH LIVE TABLE final\_transactions

AS

SELECT \* FROM transformed\_transactions;

**Exercise 2: Delta Lake Operations - Read, Write, Update, Delete, Merge**

**1.Read Data from Delta Lake**

*Using PySpark:*

# Read the Delta table

df\_transactions = spark.read.format("delta").load("/content/delta\_table")

# Show the first 5 rows

df\_transactions.show(5)

*Using SQL:*

-- Create a temporary view for SQL queries

CREATE OR REPLACE TEMP VIEW transactions AS

SELECT \* FROM delta.`/content/delta\_table`;

-- Query the first 5 rows

SELECT \* FROM transactions LIMIT 5;

**2. Write Data to Delta Lake**

*Using PySpark:*

# Create a DataFrame for new transactions

new\_transactions = [

(6, '2024-09-06', 'C005', 'Keyboard', 4, 100),

(7, '2024-09-07', 'C006', 'Mouse', 10, 20)

]

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

df\_new\_transactions = spark.createDataFrame(new\_transactions, columns)

# Append new transactions to the Delta table

df\_new\_transactions.write.format("delta").mode("append").save("/content/delta\_table")

**3. Update Data in Delta Lake**

*Using PySpark:*

from delta.tables import DeltaTable

# Load the existing Delta table

delta\_table = DeltaTable.forPath(spark, "/content /delta\_table")

# Update the price of the Laptop

delta\_table.update(

condition="Product = 'Laptop'",

set={"Price": "1300"}

)

# Verify the update

delta\_table.toDF().show()

*Using SQL:*

-- Update the price of the Laptop

UPDATE transactions

SET Price = 1300

WHERE Product = 'Laptop';

-- Verify the update

SELECT \* FROM transactions WHERE Product = 'Laptop';

**4. Delete Data from Delta Lake**

*Using PySpark:*

# Delete transactions where Quantity is less than 3

delta\_table.delete("Quantity < 3")

*Using SQL:*

-- Delete transactions where Quantity is less than 3

DELETE FROM transactions WHERE Quantity < 3;

**5. Merge Data into Delta Lake**

*Using PySpark:*

# Create a DataFrame for merging updates

merge\_data = [

(1, '2024-09-01', 'C001', 'Laptop', 1, 1250), # Updated Price

(8, '2024-09-08', 'C007', 'Charger', 2, 30) # New Transaction

]

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

df\_merge\_data = spark.createDataFrame(merge\_data, columns)

# Merge the new data into the existing Delta table

delta\_table.alias("t").merge(

df\_merge\_data.alias("s"),

"t.TransactionID = s.TransactionID"

).whenMatchedUpdate(

condition="t.Product = 'Laptop'",

set={"Price": "s.Price"}

).whenNotMatchedInsertAll().execute()

**Exercise 3: Delta Lake - History, Time Travel, and Vacuum**

**1.View Delta Table History**

*Using PySpark:*

# Read the Delta table history

history\_df = spark.sql("DESCRIBE HISTORY delta.`/content/to/delta\_table`")

history\_df.show(truncate=False)

*Using SQL:*

-- Query the history of the Delta table

DESCRIBE HISTORY delta.`/content/to/delta\_table`;

**2. Perform Time Travel**

*Using PySpark:*

# Access the Delta table at the specified version

df\_time\_travel = spark.read.format("delta").option("versionAsOf", 5).load("/content/delta\_table")

df\_time\_travel.show()

*Using SQL:*

-- Query the Delta table at a specific version

SELECT \* FROM delta.`/content/delta\_table` VERSION AS OF 5;

**3. Vacuum the Delta Table**

*Using SQL:*

-- Clean up old data using VACUUM command

VACUUM delta.`/content/delta\_table` RETAIN 168 HOURS;

*Verify Current State:*

# Check the current state of the Delta table

df\_current\_state = spark.read.format("delta").load("/content/delta\_table")

df\_current\_state.show()

**4. Converting Parquet Files to Delta Files**

**Step 1: Create a Parquet-based Table from CSV**

# Create DataFrame from CSV data

raw\_data = [

(1, '2024-09-01', 'C001', 'Laptop', 1, 1200),

(2, '2024-09-02', 'C002', 'Tablet', 2, 300),

# Add more rows as needed

]

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

df\_raw = spark.createDataFrame(raw\_data, columns)

# Write DataFrame to Parquet

parquet\_path = "/content/parquet\_table"

df\_raw.write.mode("overwrite").parquet(parquet\_path)

**Step 2: Convert Parquet to Delta**

# Convert the Parquet table to Delta format

spark.read.parquet(parquet\_path).write.format("delta").mode("overwrite").save("/content/delta\_table\_converted")

# Optionally, create a Delta table from the converted data

spark.sql("CREATE TABLE transactions\_delta USING DELTA LOCATION '/content/delta\_table\_converted'")

**Exercise 4: Implementing Incremental Load Pattern using Delta Lake**

**1.Set Up Initial Data**

# Initial transactions 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"]

# Create DataFrame

df\_initial = spark.createDataFrame(initial\_data, columns)

# Write initial data to Delta table

delta\_table\_path = "/content/delta\_table"

df\_initial.write.format("delta").mode("overwrite").save(delta\_table\_path)

**2. Set Up Incremental Data**

# Incremental transactions 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)

]

# Create DataFrame

df\_incremental = spark.createDataFrame(incremental\_data, columns**)**

**3. Implement Incremental Load**

# Filter new transactions (after 2024-09-03)

new\_transactions = df\_incremental.filter(df\_incremental.TransactionDate > '2024-09-03')

# Append new transactions to the Delta table

new\_transactions.write.format("delta").mode("append").save(delta\_table\_path)

**4. Monitor Incremental Load**

# View the history of the Delta table

history\_df = spark.sql("DESCRIBE HISTORY delta.`/content/delta\_table`")

history\_df.show(truncate=False)