**Delta Lake Hands-on Labs**

**Lab 1: Convert Parquet to Delta**

**Objective:** Convert an existing Parquet dataset to Delta format and register it as a Delta table.

**Steps:**

1. Load an existing Parquet dataset into a Spark DataFrame:

df = spark.read.parquet("/mnt/data/parquet/customers")

1. Write the DataFrame into Delta format:

df.write.format("delta").save("/mnt/data/delta/customers")

1. Register this dataset as a managed Delta table:

CREATE TABLE customers\_delta

USING DELTA

LOCATION '/mnt/data/delta/customers';

1. Verify the table creation:

SELECT \* FROM customers\_delta LIMIT 10;

**Lab 2: MERGE Operation – SCD Type 2 Update**

**Objective:** Perform Slowly Changing Dimension (SCD) Type 2 using MERGE INTO.

**Scenario:** You maintain customer information and need to track changes in customer addresses while preserving history.

**Steps:**

1. Assume you have a target Delta table:

CREATE TABLE customers\_scd ( id INT, name STRING, address STRING,start\_date DATE, end\_date DATE,current BOOLEAN) USING DELTA;

1. Prepare a source DataFrame with new customer records (some existing, some new).

source = [(1, "Alice", "New Address A"),

(3, "Charlie", "Address C")]

sourceDF = spark.createDataFrame(source, ["id", "name", "address"])

sourceDF.createOrReplaceTempView("updates")

1. Run the MERGE operation:

MERGE INTO customers\_scd t

USING updates s

ON t.id = s.id AND t.current = true

WHEN MATCHED AND t.address <> s.address

THEN UPDATE SET t.current = false, t.end\_date = current\_date()

WHEN NOT MATCHED

THEN INSERT (id, name, address, start\_date, end\_date, current)

VALUES (s.id, s.name, s.address, current\_date(), NULL, true);

1. Verify the history:

SELECT \* FROM customers\_scd ORDER BY id, start\_date;

**Lab 3: Schema Evolution – Add a New Column**

**Objective:** Add a new column dynamically without dropping the table.

**Steps:**

1. Start with a simple Delta table:

df = spark.createDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

df.write.format("delta").save("/mnt/data/delta/users")

1. Add a new column to the DataFrame:

from pyspark.sql.functions import lit

df\_new = df.withColumn("country", lit("Unknown"))

1. Write the DataFrame back to the same Delta location with schema evolution:

df\_new.write.option("mergeSchema", "true") \

.format("delta") \

.mode("append") \

.save("/mnt/data/delta/users")

1. Verify the schema:

DESCRIBE DETAIL delta.`/mnt/data/delta/users`;

**Lab 4: Time Travel – Query a Previous Version**

**Objective:** Retrieve data from an earlier state of the Delta table.

**Steps:**

1. Check table history:

DESCRIBE HISTORY delta.`/mnt/data/delta/users`;

1. Query by version number:

df\_old = spark.read.format("delta") \

.option("versionAsOf", 1) \

.load("/mnt/data/delta/users")

df\_old.show()

1. Query by timestamp:

df\_old\_ts = spark.read.format("delta") \

.option("timestampAsOf", "2025-09-01 12:00:00") \

.load("/mnt/data/delta/users")

df\_old\_ts.show()

**Lab 5: Optimize & Vacuum**

**Objective:** Improve query performance and clean up old data files.

**Steps:**

1. Run OPTIMIZE with Z-Ordering to improve data skipping:

OPTIMIZE customers\_scd ZORDER BY (id);

1. Cache frequently queried tables:

CACHE TABLE customers\_scd;

1. Remove obsolete files using VACUUM:

VACUUM customers\_scd RETAIN 168 HOURS; -- 7 days

Note: VACUUM has a default retention of 7 days to prevent accidental data loss.

1. Verify optimized storage:

DESCRIBE DETAIL customers\_scd;