**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:
2. df = spark.read.parquet("/mnt/data/parquet/customers")
3. Write the DataFrame into Delta format:
4. df.write.format("delta").save("/mnt/data/delta/customers")
5. Register this dataset as a managed Delta table:
6. CREATE TABLE customers\_delta
7. USING DELTA
8. LOCATION '/mnt/data/delta/customers';
9. Verify the table creation:
10. 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:
2. CREATE TABLE customers\_scd (
3. id INT,
4. name STRING,
5. address STRING,
6. start\_date DATE,
7. end\_date DATE,
8. current BOOLEAN
9. ) USING DELTA;
10. Prepare a source DataFrame with new customer records (some existing, some new).
11. source = [(1, "Alice", "New Address A"),
12. (3, "Charlie", "Address C")]
13. sourceDF = spark.createDataFrame(source, ["id", "name", "address"])
14. sourceDF.createOrReplaceTempView("updates")
15. Run the MERGE operation:
16. MERGE INTO customers\_scd t
17. USING updates s
18. ON t.id = s.id AND t.current = true
19. WHEN MATCHED AND t.address <> s.address
20. THEN UPDATE SET t.current = false, t.end\_date = current\_date()
21. WHEN NOT MATCHED
22. THEN INSERT (id, name, address, start\_date, end\_date, current)
23. VALUES (s.id, s.name, s.address, current\_date(), NULL, true);
24. Verify the history:
25. 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:
2. df = spark.createDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])
3. df.write.format("delta").save("/mnt/data/delta/users")
4. Add a new column to the DataFrame:
5. from pyspark.sql.functions import lit
6. df\_new = df.withColumn("country", lit("Unknown"))
7. Write the DataFrame back to the same Delta location with schema evolution:
8. df\_new.write.option("mergeSchema", "true") \
9. .format("delta") \
10. .mode("append") \
11. .save("/mnt/data/delta/users")
12. Verify the schema:
13. 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:
2. DESCRIBE HISTORY delta.`/mnt/data/delta/users`;
3. Query by version number:
4. df\_old = spark.read.format("delta") \
5. .option("versionAsOf", 1) \
6. .load("/mnt/data/delta/users")
7. df\_old.show()
8. Query by timestamp:
9. df\_old\_ts = spark.read.format("delta") \
10. .option("timestampAsOf", "2025-09-01 12:00:00") \
11. .load("/mnt/data/delta/users")
12. 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:
2. OPTIMIZE customers\_scd ZORDER BY (id);
3. Cache frequently queried tables:
4. CACHE TABLE customers\_scd;
5. Remove obsolete files using VACUUM:
6. 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;