**Delta Lake Hands-on Labs**

**Lab 1: Introduction to Delta Lakes**

**Objective:** Understand what Delta Lake is and how it improves traditional data lakes.

**Steps:**

1. **Explore limitations of traditional data lakes (Parquet/CSV):**
   * Lack of transaction guarantees.
   * Hard to handle schema mismatches.
   * Difficulty in handling updates/deletes.
2. **Enable Delta Lake in Spark/Databricks:**  
   Delta Lake is already enabled in Databricks runtime. For open-source Spark:

--packages io.delta:delta-core\_2.12:2.4.0

1. **Read Parquet data as baseline:**

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

df.show()

1. **Write the same data in Delta format:**

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

**Lab 2: Delta Lake Concepts – ACID, Schema Enforcement, Time Travel**

**Objective:** Demonstrate ACID transactions, schema enforcement, and time travel.

**Steps:**

1. **ACID Transactions:**  
   Write to the same Delta table from two notebooks/jobs; Delta ensures consistency.

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

1. **Schema Enforcement:**  
   Try writing mismatched schema:

bad\_df = spark.createDataFrame([(1, "Alice", 30)], ["id", "name", "age"])

bad\_df.write.format("delta").mode("append").save("/mnt/data/delta/customers")

→ This will fail since age is not part of schema.

1. **Time Travel:**

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

spark.read.format("delta").option("versionAsOf", 0).load("/mnt/data/delta/customers").show()

**Lab 3: Creating Delta Tables from Existing Data**

**Objective:** Convert existing Parquet/CSV tables to Delta tables.

**Steps:**

1. **Read source data:**

parquet\_df = spark.read.parquet("/mnt/data/parquet/orders")

1. **Write as Delta:**

parquet\_df.write.format("delta").save("/mnt/data/delta/orders")

1. **Register as Delta table:**

CREATE TABLE orders\_delta

USING DELTA

LOCATION '/mnt/data/delta/orders';

**Lab 4: Upserts and Deletes with MERGE INTO**

**Objective:** Perform UPSERT and DELETE operations using MERGE INTO.

**Steps:**

1. **Create target table:**

CREATE TABLE customers\_delta (id INT, name STRING, city STRING)

USING DELTA;

1. **Insert base data:**

INSERT INTO customers\_delta VALUES (1, 'Alice', 'NY'), (2, 'Bob', 'LA');

1. **Prepare updates:**

updates = [(1, "Alice", "Boston"), (3, "Charlie", "Seattle")]

updatesDF = spark.createDataFrame(updates, ["id","name","city"])

updatesDF.createOrReplaceTempView("updates")

1. **Run MERGE (Upsert):**

MERGE INTO customers\_delta t

USING updates s

ON t.id = s.id

WHEN MATCHED THEN UPDATE SET t.city = s.city

WHEN NOT MATCHED THEN INSERT (id, name, city) VALUES (s.id, s.name, s.city);

1. **Run Delete:**

DELETE FROM customers\_delta WHERE city = 'LA';

**Lab 5: Schema Evolution – Adding/Removing Columns**

**Objective:** Allow table schema to evolve dynamically.

**Steps:**

1. Start with a simple Delta table:

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

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

1. Add a new column:

from pyspark.sql.functions import lit

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

new\_df.write.option("mergeSchema", "true").format("delta").mode("append").save("/mnt/data/delta/users")

1. Verify schema change:

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

1. Remove a column (by recreating table):

ALTER TABLE delta.`/mnt/data/delta/users` DROP COLUMN country;

**Lab 6: Performance Tuning – Z-Ordering, Partitioning, Caching**

**Objective:** Improve query performance with Delta optimizations.

**Steps:**

1. Partition large dataset by a key:

big\_df.write.partitionBy("region").format("delta").save("/mnt/data/delta/sales")

1. Optimize with Z-Ordering:

OPTIMIZE sales ZORDER BY (customer\_id);

1. Cache for repeated queries:

CACHE TABLE sales;

**Lab 7: Streaming with Delta Tables**

**Objective:** Use Delta with Structured Streaming.

**Steps:**

1. Write streaming data into Delta table:

stream\_df = spark.readStream.format("json").schema(schema).load("/mnt/stream/input")

stream\_df.writeStream.format("delta").option("checkpointLocation", "/mnt/checkpoints/orders") \

.outputMode("append").start("/mnt/data/delta/orders\_stream")

1. Query streaming table:

SELECT \* FROM delta.`/mnt/data/delta/orders\_stream`;

**Lab 8: Managing Delta Logs and Vacuuming Old Data**

**Objective:** Manage transaction logs and clean old data files.

**Steps:**

1. Inspect transaction history:

DESCRIBE HISTORY orders\_delta;

1. View Delta logs in storage:  
   Files stored under /mnt/data/delta/orders/\_delta\_log/
2. Run vacuum to clean old versions:

VACUUM orders\_delta RETAIN 168 HOURS; -- keep 7 days

1. Verify active files:

DESCRIBE DETAIL orders\_delta;