**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:
3. --packages io.delta:delta-core\_2.12:2.4.0
4. **Read Parquet data as baseline:**
5. df = spark.read.parquet("/mnt/data/parquet/customers")
6. df.show()
7. **Write the same data in Delta format:**
8. 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.
2. df.write.format("delta").mode("append").save("/mnt/data/delta/customers")
3. **Schema Enforcement:**  
   Try writing mismatched schema:
4. bad\_df = spark.createDataFrame([(1, "Alice", 30)], ["id", "name", "age"])
5. 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:**
2. DESCRIBE HISTORY delta.`/mnt/data/delta/customers`;
3. 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:**
2. parquet\_df = spark.read.parquet("/mnt/data/parquet/orders")
3. **Write as Delta:**
4. parquet\_df.write.format("delta").save("/mnt/data/delta/orders")
5. **Register as Delta table:**
6. CREATE TABLE orders\_delta
7. USING DELTA
8. 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:**
2. CREATE TABLE customers\_delta (id INT, name STRING, city STRING)
3. USING DELTA;
4. **Insert base data:**
5. INSERT INTO customers\_delta VALUES (1, 'Alice', 'NY'), (2, 'Bob', 'LA');
6. **Prepare updates:**
7. updates = [(1, "Alice", "Boston"), (3, "Charlie", "Seattle")]
8. updatesDF = spark.createDataFrame(updates, ["id","name","city"])
9. updatesDF.createOrReplaceTempView("updates")
10. **Run MERGE (Upsert):**
11. MERGE INTO customers\_delta t
12. USING updates s
13. ON t.id = s.id
14. WHEN MATCHED THEN UPDATE SET t.city = s.city
15. WHEN NOT MATCHED THEN INSERT (id, name, city) VALUES (s.id, s.name, s.city);
16. **Run Delete:**
17. 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:
2. df = spark.createDataFrame([(1, "Alice")], ["id", "name"])
3. df.write.format("delta").save("/mnt/data/delta/users")
4. Add a new column:
5. from pyspark.sql.functions import lit
6. new\_df = df.withColumn("country", lit("Unknown"))
7. new\_df.write.option("mergeSchema", "true").format("delta").mode("append").save("/mnt/data/delta/users")
8. Verify schema change:
9. DESCRIBE DETAIL delta.`/mnt/data/delta/users`;
10. Remove a column (by recreating table):
11. 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:
2. big\_df.write.partitionBy("region").format("delta").save("/mnt/data/delta/sales")
3. Optimize with Z-Ordering:
4. OPTIMIZE sales ZORDER BY (customer\_id);
5. Cache for repeated queries:
6. CACHE TABLE sales;

**Lab 7: Streaming with Delta Tables**

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

**Steps:**

1. Write streaming data into Delta table:
2. stream\_df = spark.readStream.format("json").schema(schema).load("/mnt/stream/input")
3. stream\_df.writeStream.format("delta").option("checkpointLocation", "/mnt/checkpoints/orders") \
4. .outputMode("append").start("/mnt/data/delta/orders\_stream")
5. Query streaming table:
6. 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:
2. DESCRIBE HISTORY orders\_delta;
3. View Delta logs in storage:  
   Files stored under /mnt/data/delta/orders/\_delta\_log/
4. Run vacuum to clean old versions:
5. VACUUM orders\_delta RETAIN 168 HOURS; -- keep 7 days
6. Verify active files:
7. DESCRIBE DETAIL orders\_delta;