# DELTA LAKE ASSIGNMENT

1. **Database vs datawareshouse vs data lake vs Delta Lake**

| Feature | Database | Data Warehouse | Data Lake | Delta Lake |
|---|---|---|---|---|
| **Purpose** | Store current, structured transactional data (OLTP) | Store historical, structured analytical data (OLAP) | Store raw, structured + semi/unstructured data | Store data lake data with ACID guarantees, time travel, and analytics support |
| **Data Type** | Structured only | Structured (mostly) | Structured, Semi-structured, Unstructured | Structured + Semi-structured (optimized) |
| **Storage Cost** | Higher (specialized storage) | Higher (optimized for queries) | Low (commodity storage) | Low (built on data lake storage) |
| **Schema** | Fixed schema | Fixed schema (star/snowflake) | Schema-on-read | Schema enforcement + evolution |
| **Performance** | High for small OLTP queries | Optimized for complex OLAP queries | Depends on query engine | Optimized with indexing, Z-ordering |
| **Updates/Deletes** | Easy | Not common (append-only) | Hard (requires rewrite) | Easy (MERGE, UPDATE, DELETE supported) |
| **ACID Transactions** | Yes | Yes | No | Yes |
| **Use Cases** | Banking systems, ERP | BI Reporting, Trend Analysis | Data Science, Big Data | Unified analytics, machine learning, BI |
| **Example Tools** | MySQL, PostgreSQL | Snowflake, Redshift | ADLS, S3 | Delta Lake on Databricks |

**What is Delta Lake?**

- Storage layer that brings ACID transactions to Apache Spark and big data workloads.
- Allows:
    - Schema enforcement & evolution

- Time travel (query past versions of data)
- Efficient upserts/merges
- Data optimization (compaction, Z-Ordering)
- Built-in support in Azure Databricks.

## 2. Creating Delta Table using 3 Methods

### Method 1: Create from DataFrame

```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

from pyspark.sql import SparkSession


spark = SparkSession.builder.getOrCreate()


schema = StructType([
   StructField("id", IntegerType(), True),
   StructField("name", StringType(), True),
   StructField("age", IntegerType(), True)
])


data = [(1, "John", 30), (2, "Alice", 25)]
df = spark.createDataFrame(data, schema)


df.write.format("delta").mode("overwrite").save("/mnt/delta/people")
```

### Method 2: Create Managed Table

```
df.write.format("delta").mode("overwrite").saveAsTable("default.people")
```

- Stored in the metastore location.
- Databricks manages the file path.

**Method 3: Using SQL**

```
CREATE TABLE default.people_sql (
  id INT,
  name STRING,
  age INT
) USING DELTA;
```

### 3. Merge & Upsert (SCD)

- Maintain history of changes.
- Steps:
  1. Add new rows when a new address is found.
  2. Mark old records as inactive.

```
from delta.tables import DeltaTable
delta_table = DeltaTable.forPath(spark, "/mnt/delta/customers")
updates_df = spark.createDataFrame([
  (1, "John", "New York", "2024-01-01", None),
  (2, "Alice", "Boston", "2024-02-01", None)
], ["id", "name", "address", "start_date", "end_date"])
(delta_table.alias("t")
 .merge(
   updates_df.alias("u"),
   "t.id = u.id AND t.end_date IS NULL"
 )
 .whenMatchedUpdate(set={
   "end_date": "current_date()"
 })
 .whenNotMatchedInsert(values={
   "id": "u.id",
   "name": "u.name",
   "address": "u.address",
```

```
    "start_date": "u.start_date",
    "end_date": "u.end_date"
  })
  .execute()
)
```

**4. Internals of Delta Table**

A Delta Table =

- Data files (Parquet format) + Transaction log (_delta_log/ folder).
- _delta_log folder:
  - JSON log files → metadata changes.
  - Parquet checkpoints → faster queries.
- Atomicity: All-or-nothing writes.
- Version history: Every write = new table version.

Benefits of Delta Lake

- ACID transactions.
- Time travel.
- Schema enforcement + evolution.
- Merge/upsert support.
- Handles streaming + batch.
- Low-cost data lake storage + high query performance.

## Delta Lake Hands-on tutorial

### Step 1 – Load CSV Data into a Delta Table

```
from pyspark.sql.types import *
schema = StructType([
  StructField("id", IntegerType(), True),
  StructField("firstName", StringType(), True),
  StructField("middleName", StringType(), True),
  StructField("lastName", StringType(), True),
```

```
  StructField("gender", StringType(), True),
  StructField("birthDate", TimestampType(), True),
  StructField("ssn", StringType(), True),
  StructField("salary", IntegerType(), True)
])
df = spark.read.format("csv") \
   .option("header", True) \
   .schema(schema) \
   .load("/Volumes/main/default/my-volume/export.csv")
df.writeTo("main.default.people_10m").createOrReplace()
```

**Step 2 – Insert or Upsert Data**

Append:

```
df.write.mode("append").saveAsTable("main.default.people_10m")
```

Merge/Upsert:

```
MERGE INTO main.default.people_10m t
USING main.default.more_people s
ON t.id = s.id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *
```

**Step 3 – View Data**

```
people_df = spark.read.table("main.default.people_10m")
display(people_df)
```

**Step 4 – View Table History**

```
from delta.tables import DeltaTable
deltaTable = DeltaTable.forName(spark, "main.default.people_10m")
display(deltaTable.history())
```

**Step 5 – Time Travel**

```
df_v0 = spark.read.option("versionAsOf",
0).table("main.default.people_10m")

df_time = spark.read.option("timestampAsOf", "2024-05
15T22:43:15.000+00:00").table("main.default.people_10m")

display(df_v0)
```

**Step 6 – Optimize Delta Table**

```
OPTIMIZE main.default.people_10m;
```

With Z-Ordering for faster filtering:

```
OPTIMIZE main.default.people_10m ZORDER BY (gender);
```

**Step 7 – Clean Up Old Data**

```
VACUUM main.default.people_10m RETAIN 168 HOURS;
```