**Delta Lake: Advanced Concepts – Study Document**

**1. Delta Log Anatomy**

Delta Lake stores data in **Apache Parquet** format and uses a **transaction log (\_delta\_log)** to manage metadata and changes. The \_delta\_log folder exists at the root of a Delta table and contains:

**File Types:**

* .json files – atomic commits, one file per transaction (e.g., 00000000000000000010.json)
* .checkpoint.parquet – performance-optimized checkpoints combining multiple JSONs into a binary Parquet format
* \_last\_checkpoint – points to the latest checkpoint

**JSON Transaction File Format:**

Each JSON contains actions like:

* add: New data file added
* remove: File logically deleted
* metaData: Schema, table properties
* protocol: Reader/writer protocol version
* txn: App-level transaction identifier (e.g., for idempotency)

**Commit Flow:**

1. Transaction is staged
2. Changes are written as a new JSON file (atomic rename)
3. Checkpoints are written every N commits (default 10)

**2. Schema Evolution**

Delta supports **schema evolution** during:

* Writes (append) – with option mergeSchema = true
* Merges – with mergeSchema or autoMerge
* Overwrites – full replacement of schema

**Examples:**

python

# Append with evolution

df.write.option("mergeSchema", "true").format("delta").mode("append").save(path)

# Overwrite with schema change

df.write.option("overwriteSchema", "true").format("delta").mode("overwrite").save(path)

**Considerations:**

* Schema changes logged as a new metaData entry in Delta log
* Strict enforcement: Column type widening (e.g., int → long) allowed, but not narrowing

**3. OPTIMIZE ZORDER**

Used to **compact small files** and improve **data skipping** performance using **Z-ordering**.

**What is ZORDER?**

Z-ordering co-locates related column values into the same files for efficient filtering and skipping.

**Syntax:**

sql

OPTIMIZE delta.`/path/to/table` ZORDER BY (user\_id, country)

**Benefits:**

* Reduces scan time for common filters
* Ideal for large fact tables with selective queries

**Notes:**

* Compaction works best after heavy streaming or small-file ingestion
* Works only on Delta Lake tables (Databricks Runtime)

**4. VACUUM & Retention**

**VACUUM** removes old files no longer needed due to updates/deletes, optimizing storage.

**Syntax:**

sql

VACUUM delta.`/path/to/table` RETAIN 168 HOURS

**Safe Default:**

* Default retention = 7 days (168 hours)
* Must set:

sql

SET spark.databricks.delta.retentionDurationCheck.enabled = false

to run VACUUM with lower values in non-prod/testing.

**Caution:**

Running VACUUM improperly can **break time travel** if old files are deleted.

**5. Time Travel & Restore**

Delta supports querying historical data using **version number** or **timestamp**.

**Query Time Travel:**

sql

-- By version

SELECT \* FROM delta.`/path/to/table` VERSION AS OF 5;

-- By timestamp

SELECT \* FROM delta.`/path/to/table` TIMESTAMP AS OF '2023-01-01 00:00:00';

**Restore:**

sql

RESTORE TABLE my\_table TO VERSION AS OF 3;

**Limitations:**

* Limited by VACUUM and file retention period
* Requires checkpoint + JSON log to reconstruct past state

**6. Generated Columns**

**Computed columns** derived from other columns, materialized for performance.

**Syntax:**

sql

CREATE TABLE sales (

id INT,

sale\_date DATE,

year INT GENERATED ALWAYS AS (YEAR(sale\_date))

) USING DELTA;

**Benefits:**

* Stored physically, avoids recomputation
* ZORDER & partitioning can leverage them

**Limitations:**

* Cannot be updated directly
* Must be deterministic

**7. GDPR Delete Pattern**

Under GDPR, users can request **deletion of their data**.

**Soft Delete Pattern:**

1. Mark records as deleted:

sql

UPDATE users SET deleted = true WHERE user\_id = 'abc123';

1. Filter in downstream:

sql

SELECT \* FROM users WHERE deleted = false;

**Hard Delete (GDPR-compliant):**

1. Physically delete:

sql

DELETE FROM users WHERE user\_id = 'abc123';

1. VACUUM after retention period:

sql

VACUUM delta.`/users` RETAIN 0 HOURS;

1. Override safety check:

sql

SET spark.databricks.delta.retentionDurationCheck.enabled = false;

**Ensure:**

* Deleted files no longer exist in \_delta\_log or filesystem after vacuum

**8. Delta Metadata Tables**

Delta exposes **system tables** for metadata inspection (Databricks or OSS Delta v2.3+).

**System Tables:**

sql

DESCRIBE DETAIL delta.`/path` -- full table metadata

DESCRIBE HISTORY delta.`/path` -- commit history

**Sample Output:**

* operation (WRITE, DELETE, MERGE, OPTIMIZE, etc.)
* userName
* notebook or job info
* timestamp, version

**Use Cases:**

* Auditing changes
* Debugging schema evolution
* Analyzing writer patterns

**Summary Table**

| **Feature** | **Key Benefit** | **Command/Syntax Example** |
| --- | --- | --- |
| Delta Log | ACID, audit trail, performance | \_delta\_log/000000.json |
| Schema Evolution | Flexible appends and merges | mergeSchema = true |
| OPTIMIZE ZORDER | Better skipping and compaction | OPTIMIZE ZORDER BY |
| VACUUM | Clean unused files, save space | VACUUM RETAIN 168 HOURS |
| Time Travel | Query/restore historical data | VERSION AS OF, RESTORE |
| Generated Columns | Derived, indexable, storable fields | GENERATED ALWAYS AS |
| GDPR Delete Pattern | Full compliance with deletion requests | DELETE + VACUUM |
| Metadata Tables | Track operations and changes | DESCRIBE HISTORY |