Cheatsheet – Day 2

Delta Optimization, Streaming & Incremental, Advanced Transforms

---

Delta Optimization Commands

```
-- Compact small files
OPTIMIZE my_table;

-- Compact + co-locate data for filtered columns
OPTIMIZE my_table ZORDER BY (col1, col2);

-- Remove stale files (default retention: 7 days)
VACUUM my_table;
VACUUM my_table RETAIN 168 HOURS;  -- explicit
VACUUM my_table DRY RUN;           -- preview only

-- Inspect table metadata
DESCRIBE DETAIL my_table;
DESCRIBE HISTORY my_table;
```

Liquid Clustering (replaces partitioning + ZORDER)

```
CREATE TABLE t CLUSTER BY (col1, col2) AS SELECT ...;
ALTER TABLE t CLUSTER BY (col3);  -- change clustering keys
```

Auto Loader (cloudFiles)

```python
df = (spark.readStream
    .format("cloudFiles")
    .option("cloudFiles.format", "json")
    .option("cloudFiles.schemaLocation", checkpoint_path)
    .load("/source/path"))

(df.writeStream
    .option("checkpointLocation", checkpoint_path)
    .trigger(availableNow=True)
    .toTable("catalog.schema.target"))
```

COPY INTO (SQL idempotent ingestion)

```sql
COPY INTO catalog.schema.target
FROM '/source/path'
FILEFORMAT = JSON;
```

COPY INTO vs Auto Loader

Feature	COPY INTO	Auto Loader (cloudFiles)
Language	SQL	Python (PySpark)
Scale	Thousands of files	Millions+ of files
Schema evolution	Manual	Automatic (`schemaLocation`)
File tracking	SQL-state based	Checkpoint-based
Idempotent	Yes	Yes
Best for	Simple SQL pipelines	Production streaming ETL

MERGE INTO (Upsert)

```
MERGE INTO target t
USING source s
ON t.id = s.id
WHEN MATCHED THEN UPDATE SET t.name = s.name, t.updated =
        s.updated
WHEN NOT MATCHED THEN INSERT (id, name, updated) VALUES
        (s.id, s.name, s.updated)
WHEN NOT MATCHED BY SOURCE THEN DELETE;
```

- Atomic upsert: insert + update + delete in one statement
- Exactly-once semantics with Delta Lake

CLONE (Table Copy)

```
-- Deep Clone: full physical copy of data + metadata
CREATE TABLE backup DEEP CLONE source_table;

-- Shallow Clone: copy metadata only, reference source data
        files
CREATE TABLE test_copy SHALLOW CLONE source_table;
```

	Deep Clone	Shallow Clone
Data copied?	Yes (full copy)	No (references source)
Independent?	Fully independent	Depends on source
Best for	Backups, migration	Testing, experiments
Cost	Storage × 2	Minimal

Change Data Feed (CDF)

```
-- Enable CDF on table
ALTER TABLE my_table SET TBLPROPERTIES
        (delta.enableChangeDataFeed = true);
```

```sql
-- Read changes (batch)
SELECT * FROM table_changes('my_table', 2);       -- from
        version 2
SELECT * FROM table_changes('my_table', 2, 5);     --
        versions 2-5

-- Read changes (streaming)
```

```python
df = (spark.readStream
    .option("readChangeFeed", "true")
    .option("startingVersion", 2)
    .table("my_table"))
```

`_change_type`	**Meaning**
`insert`	New row
`update_preimage`	Row before update
`update_postimage`	Row after update
`delete`	Deleted row

Structured Streaming Triggers

Trigger	**Behavior**
(default)	Continuous micro-batches
`trigger(availableNow=True)`	Process all available, then stop
`trigger(processingTime='10 seconds')`	Micro-batch every 10s
`trigger(once=True)`	One micro-batch, then stop (legacy)

Structured Streaming Output Modes

Mode	Behavior	Use Case
`append`	Only new rows	Raw ingestion, logs
`complete`	Full result rewrite	Aggregations (groupBy)
`update`	Only changed rows	Stateful updates

```
(df.writeStream
    .outputMode("append")          # or "complete", "update"
    .option("checkpointLocation", path)
    .trigger(availableNow=True)
    .toTable("target"))
```

Window Functions

```
-- Ranking
ROW_NUMBER() OVER (PARTITION BY cat ORDER BY val DESC) AS rn
RANK() OVER (PARTITION BY cat ORDER BY val DESC) AS rnk
DENSE_RANK() OVER (...)

-- Running total
SUM(amount) OVER (
    PARTITION BY category
    ORDER BY date
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
) AS running_total

-- Lead / Lag
LAG(price, 1) OVER (ORDER BY date) AS prev_price
LEAD(price, 1) OVER (ORDER BY date) AS next_price
```

Higher-Order Functions

```
-- Array operations
explode(array_col)                    -- array → rows
TRANSFORM(arr, x -> x * 2)           -- map each element
FILTER(arr, x -> x > 0)              -- filter elements
EXISTS(arr, x -> x > 100)            -- boolean check
```

CTEs (Common Table Expressions)

```
WITH
    step1 AS (SELECT ... FROM raw),
    step2 AS (SELECT ... FROM step1 WHERE ...),
    step3 AS (SELECT ... FROM step2 JOIN ...)
SELECT * FROM step3;
```

Key Exam Points

- `OPTIMIZE` compacts files; does NOT remove old versions
- `VACUUM` removes old files; breaks time travel for removed versions
- `VACUUM` default retention: 7 days (168 hours)
- Checkpoint = streaming state (exactly-once guarantee)
- `availableNow=True` = best for scheduled incremental jobs
- `CLUSTER BY` = Liquid Clustering (modern replacement for partitioning)
- `explode()` turns array elements into rows
- Window functions need `OVER (PARTITION BY ... ORDER BY ...)`
- `MERGE INTO` = atomic upsert (insert + update + optional delete)
- Deep Clone = full copy; Shallow Clone = metadata only
- CDF: `_change_type` column tracks insert/update/delete
- CDF must be enabled with `delta.enableChangeDataFeed = true`
- `table_changes()` reads CDF history (SQL)

- Output mode `append` = default for streaming tables
- `DESCRIBE DETAIL` → file count, size, location
- `DESCRIBE HISTORY` → version history, operations, timestamps