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)

```
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)

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

- Idempotent: skips already-processed files
- Good for thousands of files; Auto Loader better for millions

---

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	Behavior
`trigger(once=True)`	One micro-batch, then stop (legacy)

Window Functions

```sql
-- 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

```sql
-- 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
- 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 ...)`