LAB 07: Lakeflow Declarative Pipeline

Duration: ~45 min | **Day:** 3 | **After module:** M07: Medallion & Lakeflow Pipelines | **Difficulty:** Advanced

---

Scenario

> *“Build a complete Medallion Pipeline using Lakeflow Declarative Pipelines. First you’ll create a pipeline via the Databricks UI (Workshop), then practice writing SQL declarations for Bronze, Silver, and Gold layers (Practice).”*

Objectives

After completing this lab you will be able to: - Create and configure a Lakeflow Declarative Pipeline in the Databricks UI - Upload SQL source files and set pipeline variables - Write `STREAMING TABLE` declarations for Bronze - Add data quality expectations (`ON VIOLATION DROP ROW`) - Create `MATERIALIZED VIEW` declarations for Gold - Verify pipeline results and query the Event Log

---

Prerequisites

- Cluster running and attached to notebook
- SQL source files available in `materials/lakeflow/`

Section 1: Workshop — Building the Pipeline in UI

Follow the trainer's instructions step by step.

Step 1: Upload SQL Files

- Upload the SQL transformation files from `materials/lakeflow/` via Databricks UI or Git Folders

Step 2: Create Pipeline

1. Go to **Workflows** → **Pipelines** → **Create Pipeline**
2. Set **Pipeline name**: `lakeflow_pipeline_<your_name>`
3. Set **Catalog**: `retailhub_<your_name>`
4. Set **Target schema**: `<your_name>_lakeflow`
5. Add the uploaded SQL files as source code

Step 3: Configure Variables

- Add pipeline configuration keys:
    - `customer_path` → path to customer data
    - `order_path` → path to order data
    - `product_path` → path to product data

Step 4: Run the Pipeline

1. Click **Start** to run the pipeline
2. After first run completes, add a new file to `orders/stream/`
3. Run the pipeline again — observe incremental processing
4. Check the **Event Log** tab for processing metrics

Step 5: Verify Results

- Query `fact_sales` with joins to `dim_customer` , `dim_product` , `dim_date`

- Check SCD Type 2 history in `silver_customers`

Section 2: Practice — Lakeflow SQL Declarations

Open **`LAB_07_code.ipynb`** and complete the `# TODO` cells.

Task	What to do	Key concept
Task 1	Write Bronze Declaration	`CREATE OR REFRESH STREAMING TABLE` + `read_files()`
Task 2	Write Silver with Expectations	`ON VIOLATION DROP ROW` for constraints
Task 3	Write Gold Declaration	`CREATE OR REFRESH MATERIALIZED VIEW`
Task 4	Compare ST vs MV	Fill comparison table (Streaming Table vs Materialized View)
Task 5	Verify Pipeline Results	Query bronze, silver, gold tables
Task 6	Check Pipeline Event Log	`event_log(TABLE(...))` for data quality metrics

---

Detailed Hints

Task 1: Bronze — STREAMING TABLE

- `CREATE OR REFRESH STREAMING TABLE bronze_orders`
- Source: `STREAM read_files('{path}', format => 'json')`

Task 2: Silver — Expectations

- Constraints use `EXPECT (condition) ON VIOLATION DROP ROW`
- First constraint: `order_id IS NOT NULL`

- Second constraint: `total_price > 0`
- Source: `STREAM(bronze_orders)`

Task 3: Gold — MATERIALIZED VIEW

- `CREATE OR REFRESH MATERIALIZED VIEW gold_daily_revenue`
- Materialized Views are recalculated from scratch each run

Task 4: ST vs MV

Feature	Streaming Table	Materialized View
Processing mode	Incremental (append)	Full recompute
Best for	Raw/Bronze ingestion	Aggregations/Gold

Task 6: Event Log

- `SELECT * FROM event_log(TABLE(catalog.schema.table))`
- Filter by `event_type = 'flow_progress'` for data quality metrics

Summary

In this lab you: - Created a complete Lakeflow Pipeline via Databricks UI - Wrote SQL declarations for Bronze (STREAMING TABLE), Silver (with expectations), and Gold (MATERIALIZED VIEW) - Verified incremental processing with streaming sources - Queried the Event Log for pipeline monitoring

> **Exam Tip:** `STREAMING TABLE` processes data incrementally (append-only). `MATERIALIZED VIEW` recalculates fully each run. Use `ON VIOLATION DROP ROW` to silently filter invalid rows. `FAIL` stops the pipeline. `EXPECT` without action only logs warnings.

> **What’s next:** In LAB 08 you will create multi-task Jobs with dependencies and triggers for orchestrating pipelines.