Building a Declarative Sales Data Warehouse with Databricks Delta Live Tables

Author: Meet Nirav Zaveri

Date: September 9, 2025

Version: 1.0

Table of Contents

1. Introduction

1.1. Purpose

This document provides a comprehensive, step-by-step guide for implementing an end-to-end data pipeline using Databricks Delta Live Tables (DLT). The objective is to build a scalable and maintainable Sales Data Warehouse that follows the modern Medallion Architecture, transforming raw data into business-ready insights.

1.2. What are Delta Live Tables (DLT)?

Delta Live Tables is a declarative framework for building reliable, maintainable, and testable data processing pipelines. Instead of defining your pipeline's execution steps, you *declare* the desired end state of your data transformations. DLT then automatically manages task orchestration, cluster management, monitoring, data quality, and error handling.

1.3. Key Benefits

- **Automated Data Quality:** Define data quality rules ("expectations") directly in your pipeline code to prevent bad data from flowing downstream.
- **Simplified Dependency Management:** DLT automatically infers the relationships between datasets, building and maintaining the Directed Acyclic Graph (DAG) for you.
- **Unified Batch and Streaming:** Use the same code to process data in both batch and streaming modes. DLT handles the complexity of incremental processing.
- **Automatic Maintenance:** DLT automates complex tasks like OPTIMIZE and VACUUM to ensure tables are performant.

1.4. Project Overview

We will ingest sales data from two different regions, along with dimension data for products and customers. This data will be processed through a three-tiered Medallion Architecture:

- **Bronze Layer:** Ingests raw source data.
- **Silver Layer:** Creates cleansed, conformed, and enriched tables using upserts (SCD Type 1).
- **Gold Layer:** Builds a dimensional model with Slowly Changing Dimensions (SCD Type 2) and aggregated business views for analytics.

2. Prerequisites & Environment Setup

2.1. Required Skills

- Basic understanding of data warehousing concepts (ETL, Dimensional Modeling).
- Familiarity with the Databricks workspace.
- Foundational knowledge of SQL and Python (PySpark).

2.2. Databricks Account Setup

This guide uses the Databricks Free Edition, which can be set up with a standard Gmail account and requires no subscription or credit card.

1. Navigate to the Databricks Try page.
2. Locate the link for the **Free Edition** and sign up using your email.
3. Once registered, log in to your Databricks workspace.

2.3. Enabling the Lakeflow Pipeline Editor

To use the modern, file-based DLT editor, you must enable the experimental feature.

1. In the left sidebar, navigate to **Settings** > **Developer**.
2. Scroll down to the **Experimental features** section.
3. Toggle the **Lakeflow pipeline editor** to the "On" position. This provides a purpose-built IDE for creating DLT pipelines with folder-based organization.

3. Phase 1: Source Data Setup

3.1. Create Catalog and Schema

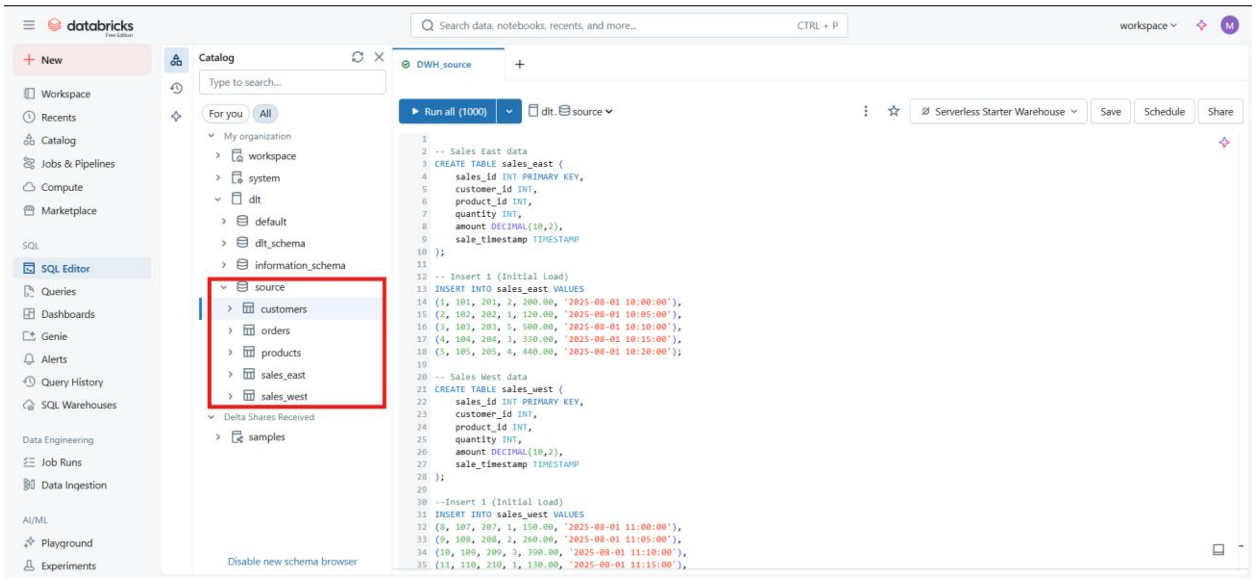
First, we establish a dedicated location for our pipeline's data.

1. Navigate to the **Catalog** explorer.
2. Click **Create Catalog**.
   - **Name:** dlt
   - Follow the prompts, granting access to your workspace.
3. Select the newly created dlt catalog.
4. Click **Create Schema**.
   - **Name:** source
   - This schema will hold our raw source tables.

3.2. Create and Populate Source Tables

Next, we use the SQL Editor to create and load our initial source data.

1. Navigate to the **SQL Editor**.
2. In the editor's context dropdowns, select the dlt catalog and the source schema.
3. Execute the DDL and DML statements provided to create and populate the following tables:
   - sales_east
   - sales_west
   - products
   - customers



```sql

-- Sales East data
CREATE TABLE sales_east (
    sales_id INT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    quantity INT,
    amount DECIMAL(10,2),
    sale_timestamp TIMESTAMP
);

-- Insert 1 (Initial Load)
INSERT INTO sales_east VALUES
(1, 101, 201, 2, 200.00, '2025-08-01 10:00:00'),
(2, 102, 202, 1, 120.00, '2025-08-01 10:05:00'),
(3, 103, 203, 5, 500.00, '2025-08-01 10:10:00'),
(4, 104, 204, 3, 330.00, '2025-08-01 10:15:00'),
(5, 105, 205, 4, 440.00, '2025-08-01 10:20:00');

-- Sales West data
CREATE TABLE sales_west (
    sales_id INT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    quantity INT,
    amount DECIMAL(10,2),
    sale_timestamp TIMESTAMP
);

--Insert 1 (Initial Load)
INSERT INTO sales_west VALUES
(8, 107, 207, 1, 150.00, '2025-08-01 11:00:00'),
(9, 108, 208, 2, 260.00, '2025-08-01 11:05:00'),
(10, 109, 209, 3, 390.00, '2025-08-01 11:10:00'),
(11, 110, 210, 1, 130.00, '2025-08-01 11:15:00'),
```

4. Phase 2: Building the DLT Pipeline

4.1. Pipeline and Folder Structure Setup

1. From the Databricks homepage, click **New** > **ETL Pipeline**.
2. In the catalog dropdown, select dlt.
3. Click **Create Schema** and name it dlt_pipeline. This schema will store the output tables (Bronze, Silver, Gold).
4. Select **Start with sample code in Python**.
5. Ensure the **Lakeflow pipelines editor** toggle is **On** and click **Create**.
6. You are now in the DLT editor. In the left-hand assets browser:
    - Rename the pipeline at the top to Sales Warehouse Pipeline if needed.
    - Rename the default transformations folder to source_code if needed. This folder will contain all our pipeline logic.
    - Inside source_code, create three new folders: bronze, silver, and gold.
    - Delete any sample files created by default.

4.2. The Bronze Layer: Raw Data Ingestion

The Bronze layer ingests data with minimal transformation while ensuring basic data quality.

Objective: Combine sales_east and sales_west into a single table and ingest product and customer data.

1. **Create Files:** Inside the bronze folder, create three new Python files:
    - 1_ingest_sales.py
    - 2_ingest_products.py
    - 3_ingest_customers.py
2. **Implement Ingestion Logic:**
    - In 1_ingest_sales.py, use dlt.create_streaming_table() to define an empty target table. Then, use two separate functions decorated with @dlt.append_flow, one for each source table (sales_east, sales_west), to stream data into the target table. This pattern is ideal for unifying multiple sources.
    - In the other files, use a standard @dlt.table decorator to create streaming tables directly from the products and customers source tables.
    - **Data Quality:** Add data quality rules using @dlt.expect_all_or_drop. For example, ensure that key columns like sales_id and product_id are not null.

4.3. The Silver Layer: Cleansed and Enriched Data

The Silver layer refines the data, applies business logic, and creates an enterprise view of key entities.

Objective: Create enriched tables for Sales, Products, and Customers with upsert logic (SCD Type 1).

1. **Create Files:** Inside the silver folder, create three new Python files:
    - 1_transform_sales.py
    - 2_transform_products.py
    - 3_transform_customers.py
2. **Implement Transformation Logic (SCD Type 1):**

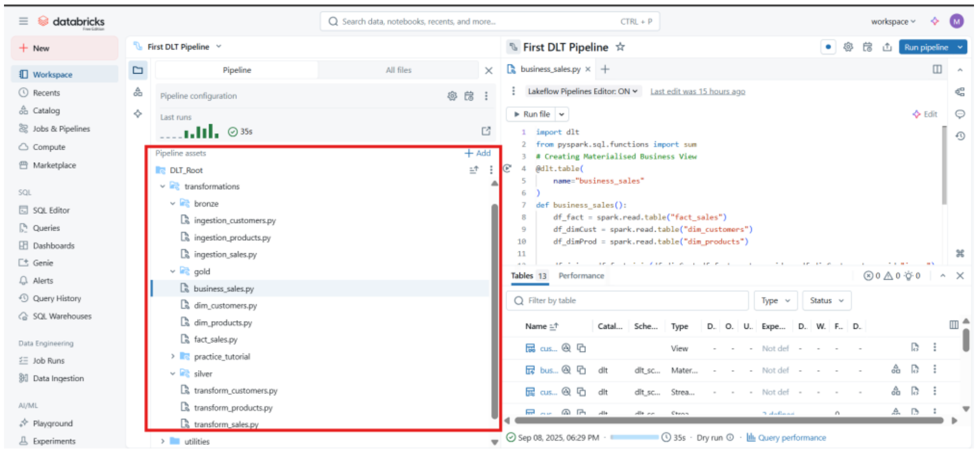
- For each entity, first create a streaming view (@dlt.view) that reads from the corresponding Bronze table. Apply transformations within this view (e.g., type casting, creating new columns like total_amount).
- Next, use the dlt.create_auto_cdc_flow() function. This powerful feature handles the complexity of upserts.
  - **target**: The name of the Silver table (e.g., products_enriched).
  - **source**: The name of the transformation view created above.
  - **keys**: The primary key of the table (e.g., ["product_id"]).
  - **sequence_by**: A column like last_updated to resolve duplicates, ensuring only the latest record is processed.
  - **scd_type**: Set to "1" for upsert behavior.

4.4. The Gold Layer: Business-Ready Dimensional Model

The Gold layer creates data models optimized for analytics and reporting.

Objective: Build a dimensional model with SCD Type 2 dimensions and a final aggregated business view.

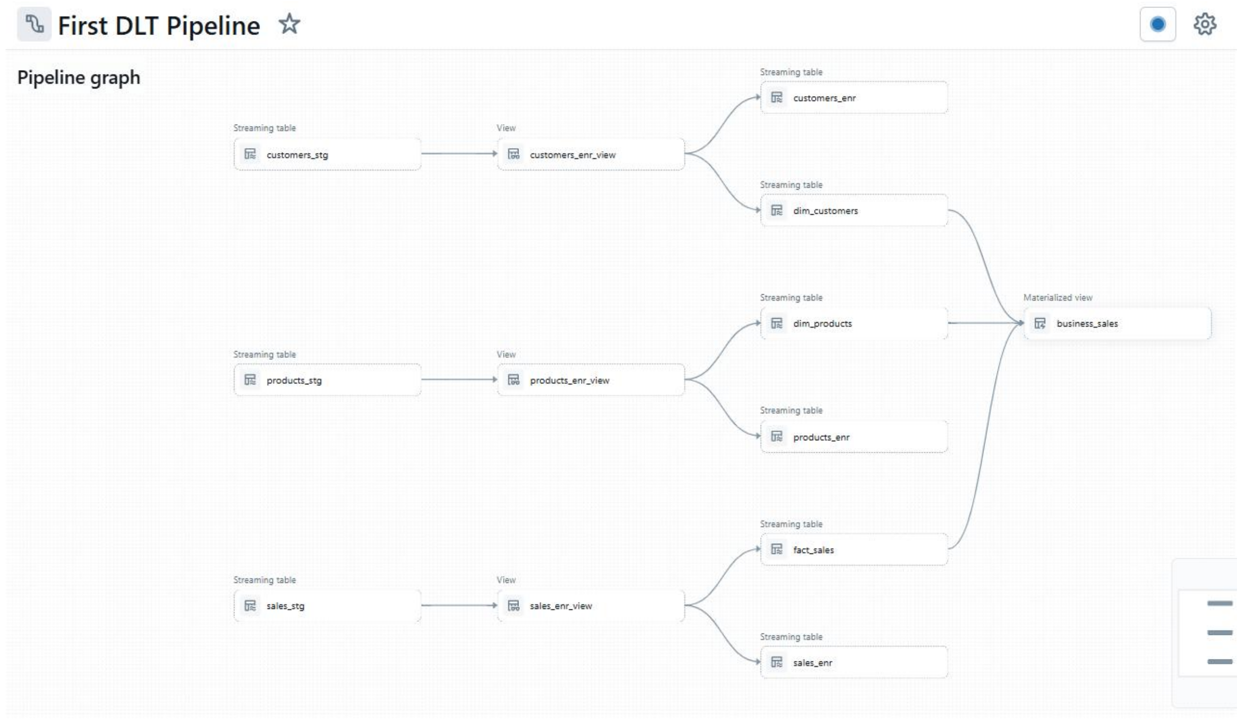
1. **Create Files:** Inside the gold folder, create four new Python files:
   - 1_dim_products.py
   - 2_dim_customers.py
   - 3_fact_sales.py
   - 4_business_view_sales.py
2. **Implement Dimensional Model Logic:**
   - **Dimensions (SCD Type 2):** In 1_dim_products.py and 2_dim_customers.py, use dlt.create_auto_cdc_flow() again. This time, set **scd_type to "2"**. DLT will automatically manage the historical records by adding __START_AT, __END_AT, and __IS_CURRENT columns to track the full history of changes.
   - **Fact Table:** In 3_fact_sales.py, create the fact table using SCD Type 1, as facts are typically updated, not versioned.
   - **Business View:** In 4_business_view_sales.py, create a final materialized view (@dlt.table). This view joins the fact table with the dimension tables and calculates key business metrics, such as total sales by region and product category.



```
import dlt
from pyspark.sql.functions import sum
# Creating Materialised Business View
@dlt.table(
    name="business_sales"
)
def business_sales():
    df_fact = spark.read.table("fact_sales")
    df_dimCust = spark.read.table("dim_customers")
    df_dimProd = spark.read.table("dim_products")

```

Name	Catal...	Sche...	Type	D..	O..	U..	Expe...	D..	W..	F...	D..
cus...			View	-	-	-	Not def	-	-	-	-
bus...	dlt	dlt_sc...	Mater...	-	-	-	Not def	-	-	-	-
cus...	dlt	dlt_sc...	Strea...	-	-	-	Not def	-	-	-	-



5. Phase 3: Pipeline Execution & Validation

5.1. Initial Pipeline Run

1. Before the first run, click **Dry Run**. DLT will validate the code, resolve dependencies, and display the full pipeline DAG without processing data. This is a crucial step for catching errors early.
2. Once the dry run is successful, click **Run Pipeline** to perform the initial data load.

5.2. Incremental Data Load and Validation

To test the pipeline's incremental and change data capture capabilities:

1. Use the SQL Editor to INSERT new sales records into the sales_east and sales_west tables.
2. UPDATE a few records in the products and customers source tables to simulate changes.
3. Re-run the pipeline.
4. Observe the pipeline graph and event logs. Note that only the new and changed records are processed, demonstrating the efficiency of incremental loads. The record counts will be significantly smaller than the initial full load.

5.3. Verifying SCD Type 2 History

Query the Gold layer dimension tables (e.g., dim_products) to validate the SCD Type 2 implementation.

```
SELECT * FROM dlt.source.dlt_pipeline.dim_products
WHERE product_id = '[ID of a product you updated]';
```

The result will show two rows for the updated product.

6. Phase 4: Monitoring & Orchestration

6.1. Monitoring Pipeline Health

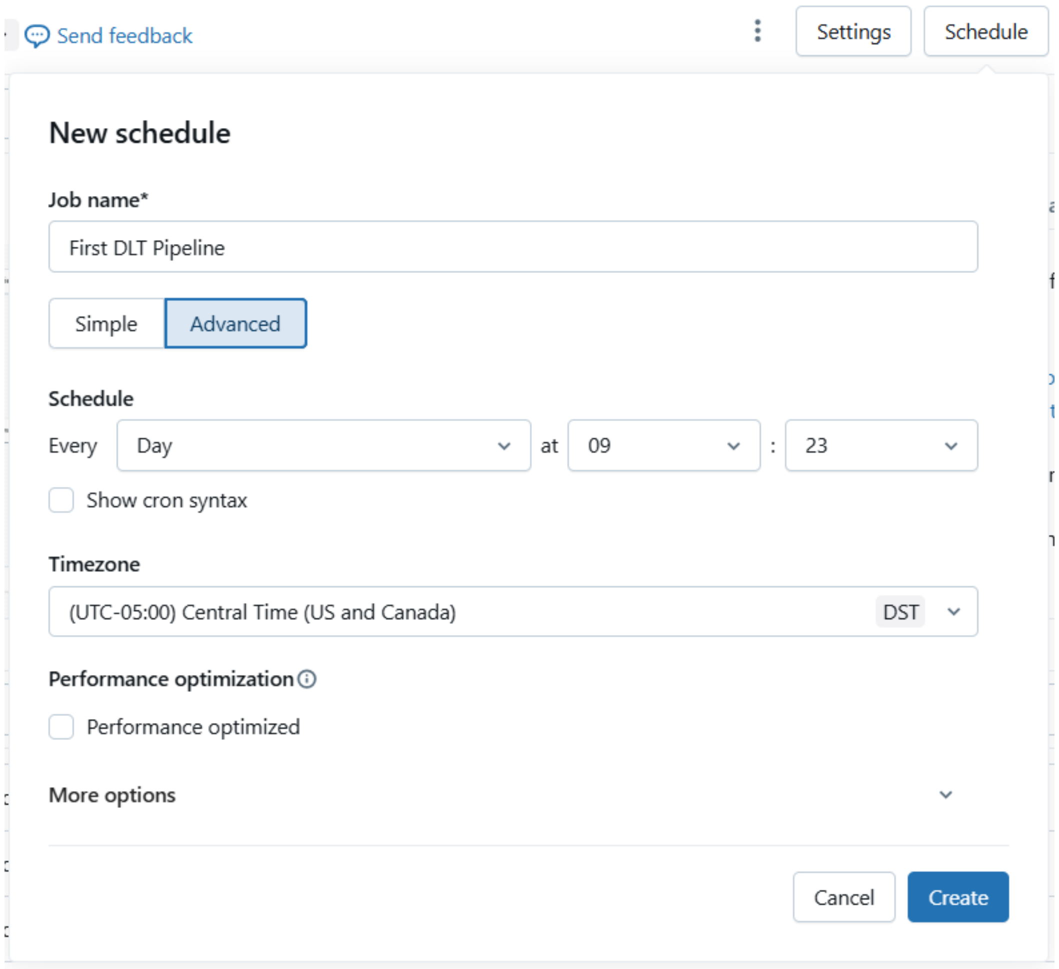
The DLT UI provides a comprehensive dashboard for each pipeline run.

- **Pipeline Graph:** Visually inspect the flow and status of each table.
- **Data Quality:** Click on any table to view the results of your data quality expectations (number of records passed, failed, or dropped).
- **Event Log:** Access detailed logs for troubleshooting and auditing.

6.2. Scheduling the Pipeline

To automate execution:

1. In the pipeline view, click **Schedule**.
2. Define a schedule using cron syntax (e.g., to run daily, hourly).
3. Click **Create**. The pipeline will now run automatically without manual intervention.



New schedule

Job name*
First DLT Pipeline

Simple [✓ Advanced]

Schedule
Every Day at 09 : 23
Show cron syntax

Timezone
(UTC-05:00) Central Time (US and Canada) DST

Performance optimization
Performance optimized