**Incremental Data Loading in Microsoft Fabric**

**What is Incremental Loading?**

Incremental Data Loading is a method where only new or updated data is loaded into your system instead of loading the entire data every time. This helps save time, improve performance, and reduce cost.

**Why Use Incremental Load in Microsoft Fabrics?**

1. Improves data loading speed by processing only new or updated records.
2. Reduces load on source and target systems.
3. Minimizes storage and compute costs.
4. Ensures reports and dashboards have up-to-date data.
5. Ideal for handling large and growing datasets.

**Where is Incremental Loading Used in Microsoft Fabric?**

* **Source Systems (like SQL, Oracle, Excel):**  
  Incremental loading starts by pulling only newly added or changed records from databases, cloud sources, or files.
* **Lakehouse (Fabric’s Storage):**  
  The new or changed data is stored in the Lakehouse, usually in a structured folder format like Bronze (raw), Silver (cleaned), and Gold (final).
* **Pipelines:**  
  Data pipelines automate the movement of incremental data from the source to the destination, ensuring scheduled or trigger-based loading.
* **Notebooks or Dataflows:**  
  These are used to clean, combine, or merge new data with existing data, especially in the silver layer where deduplication and transformation happen.

**When to Use Incremental Load?**

* **Daily or Weekly Batch Jobs:**  
  When data needs to be refreshed regularly without reloading everything, like daily sales or weekly updates.
* **Operational Reports or Dashboards (e.g., Power BI):**  
   To keep dashboards updated with only the latest changes without slowing down performance.
* **ETL Pipelines with Historical + New Data:**  
   When pipelines need to preserve history but only process recent data during each run.
* **Change Data Capture (CDC) Systems:**  
  When you're tracking insertions, updates, or deletions in real time or near real time and loading only those changes.

**What is Change Data Capture (CDC)?**

Change Data Capture (CDC) is a data management technique used toidentify and capture changes (INSERTS, UPDATES, DELETES) that happen in a source database, so only the changed data can be loaded into the target system.

Instead of loading the full table again, CDC tracks what changed and when it changed, making data pipelines more efficient and real-time ready.

## **Benefits of CDC in Data Engineering:**

* Real-time or near-real-time reporting
* Faster ETL workflows
* Reduced data latency
* Scalable for big data use cases

## **Where is CDC Used in Microsoft Fabric?**

In Microsoft Fabric, CDC can be applied in the following ways:

* **Source Side (SQL/Oracle):**  
   Track changes using last\_updated\_date or audit columns.
* **Fabric Pipelines:**  
   Use filters in the copy activity to load only new or changed records.
* **Lakehouse Silver Layer:**  
   Use notebooks to merge new changes with existing data while avoiding duplicates.

**How Does CDC Work?**

* **Timestamp Column**
  + Uses a column like last\_updated\_date or modified\_time.
  + Only records changed after the last load timestamp are selected.
  + Simple and widely used in incremental loading.
* **Triggers or Audit Tables**
  + Database triggers capture changes (INSERT, UPDATE, DELETE).
  + Changes are written to a separate audit or log table.
  + Pipelines read from this table to fetch only the new/changed data.
* **Built-in CDC Feature (in Databases like SQL Server or Oracle)**
  + Native CDC capabilities track row-level changes automatically.
  + Changes are stored in system-managed CDC tables.
  + Offers detailed change tracking without custom logic.

## **Example Use Case:**

Let’s say our Sales table is updated every hour.

* Instead of loading the entire table, CDC lets us pull only the records where last\_updated\_date > last\_load\_time.
* This filtered data is merged with the existing data in your Fabric Lakehouse.
* Dashboards get refreshed with only the newly changed data.

**Approaches to Perform Incremental Loading in Fabric:**

### **Using a Single Pipeline:**

* A single pipeline is created to handle both full load (initial load) and incremental load (daily or regular updates).
* Inside the pipeline, conditions or filters (like last\_updated\_date) are used to decide whether to load all data or only new data.
* This method is simpler to build, especially for small datasets or short-term projects.
* However, as the project grows, the logic becomes more complex and harder to manage (e.g., handling multiple branches inside the same pipeline).
* Best suited for quick use cases or simple source systems.

### **Using Two Pipelines (Recommended for Professionals)**

* This method uses two separate pipelines for better clarity and control:
  + **Pipeline 1:**  
    Responsible for pulling data from the source (SQL, Oracle, etc.) It loads both raw full data (initial) and incremental updates (daily/hourly changes) into the bronze layer of the Lakehouse.
  + **Pipeline 2:**  
    Reads data from the bronze layer and performs cleaning, deduplication, and merging with existing data. This cleaned and updated data is stored in the silver layer.
* This approach is modular, meaning each pipeline has a specific job — making it easier to debug, maintain, and scale.
* Highly recommended for enterprise-level, production-ready environments where data quality and manageability are crucial.

## **How Medallion Architecture Works in Incremental Data Loading (Microsoft Fabric):**

The Medallion Architecture is a layered design approach in Microsoft Fabric used to organize and process data in stages. It works perfectly with incremental data loading to ensure clean, optimized, and up-to-date data for reporting and analytics.

### **1. Bronze Layer – Raw & Incremental Data Storage:**

* This is the landing zone where data is first ingested from the source systems.
* In incremental loading, only new or changed data is loaded here daily or hourly.
* Both initial full loads and ongoing incremental loads are stored here in raw form.
* Data is stored in Delta Tables inside the Lakehouse.

### **2. Silver Layer – Cleaned & Combined Data:**

* In this layer, data from the bronze layer is cleaned, transformed, and deduplicated.
* Merging logic is applied to combine new changes with historical data.
* This step ensures that the data is reliable, consistent, and ready for analysis.
* It handles issues like duplicates, nulls, and schema alignment.

### **3. Gold Layer – Final Analytical Data:**

* The Gold layer contains business-ready, aggregated, and optimized data.
* Only necessary columns and tables are kept for reporting and dashboards.
* Ideal for Power BI and other visualization tools to use with high performance.
* Gold data is refreshed with only the changed parts, thanks to the incremental logic applied earlier.

### **What is SCD (Slowly Changing Dimension)?**

* SCD refers to a technique used to manage and track changes in dimension data (like customer, product, employee) over time.
* These changes don’t happen frequently (hence “slowly”), but it’s important to decide whether to overwrite, track history, or keep both versions.
* SCD logic is commonly used in data warehouses, lakehouses, and business reporting setups, especially during incremental loading.

### **Types of SCD:**

#### **SCD Type 0 – Fixed Dimension:**

* No changes are allowed; the data remains as it was originally loaded.
* Rarely used in dynamic systems.

#### **SCD Type 1 – Overwrite Old Data:**

* When a change happens, the old data is simply updated/overwritten.
* No history is preserved.
* Simple to implement but not suitable when historical tracking is needed.

***SCD Type 2 – Keep Full History:***

* A new row is added for every change, and the old row is retained.
* Allows you to track the entire history of changes.
* Most common in incremental data loading and widely used by data engineers in enterprise environments.
* Requires extra columns like effective\_date, end\_date, is\_current.

#### **SCD Type 3 – Keep Limited History:**

* Only the previous and current values are stored in the same row.
* Useful when only 1-step change tracking is needed (e.g., current and previous address).
* Not widely used in large-scale systems.

#### **Hybrid Types (e.g., Type 6):**

* Combines Type 1, Type 2, or Type 3 depending on business needs.
* Rare and complex; used in special modeling cases.

### **Which SCD Type is Mostly Used in Microsoft Fabric?**

* SCD Type 2 is the most commonly used in incremental data loading scenarios.
* It's ideal when you use Bronze → Silver → Gold layers and want to track changes over time.
* In Microsoft Fabric, this logic is often handled in Notebooks (PySpark) or Dataflows, especially in the silver layer where merging and deduplication happens.

Using the above approach to incremental loading, the following end-to-end process was implemented to build a scalable data pipeline in Microsoft Fabric.

## **Project Title:**

Incremental Data Analysis Using Medallion Architecture in Microsoft Fabric.

**Objective:**

To simulate daily sales data ingestion from Oracle DB, apply transformations using PySpark in Fabric notebooks, and create a Power BI dashboard with real-time alerts. This project uses Microsoft Fabric’s **Lakehouse architecture** following the **Bronze-Silver-Gold** model.

## **Business Goal:**

* Analyze **gender distribution** and **product category sales.**
* Automate **incremental data ingestion** from Oracle DB.
* Provide a **Power BI report** with real-time alerts.
* Follow **best practices in data lakehouse architecture.**

## **Technology Stack:**

* Microsoft Fabric (Lakehouse, Notebooks, Pipelines)
* Oracle Database (source)
* PySpark
* Power BI
* Medallion Architecture (Bronze → Silver → Gold)

**Architecture Diagram:**

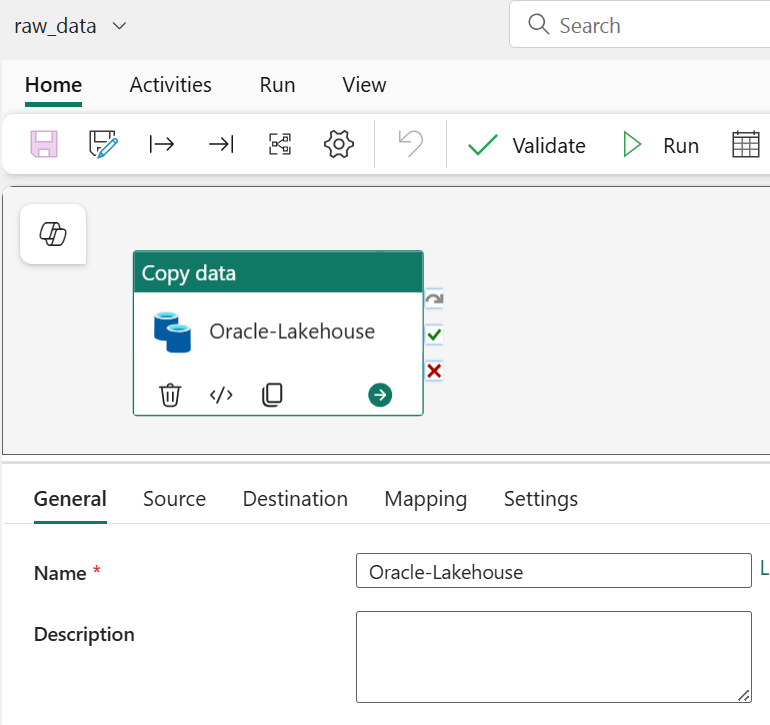
**Step-by-Step Process:**

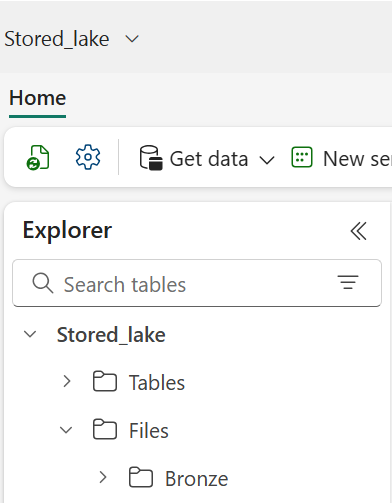
### **Step 1: Simulate Sales Data in Oracle**

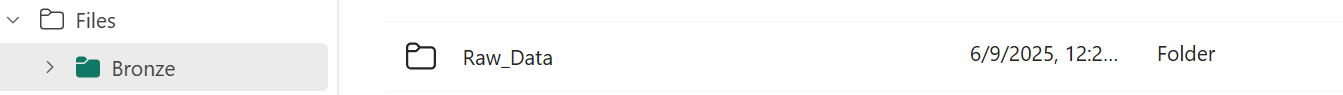
* Generated a Python script to insert 5,000 new records daily into the Oracle DB.
* Created the required **Sales table** schema in Oracle.

### **Step 2: Create Lakehouse and Pipeline in Fabric**

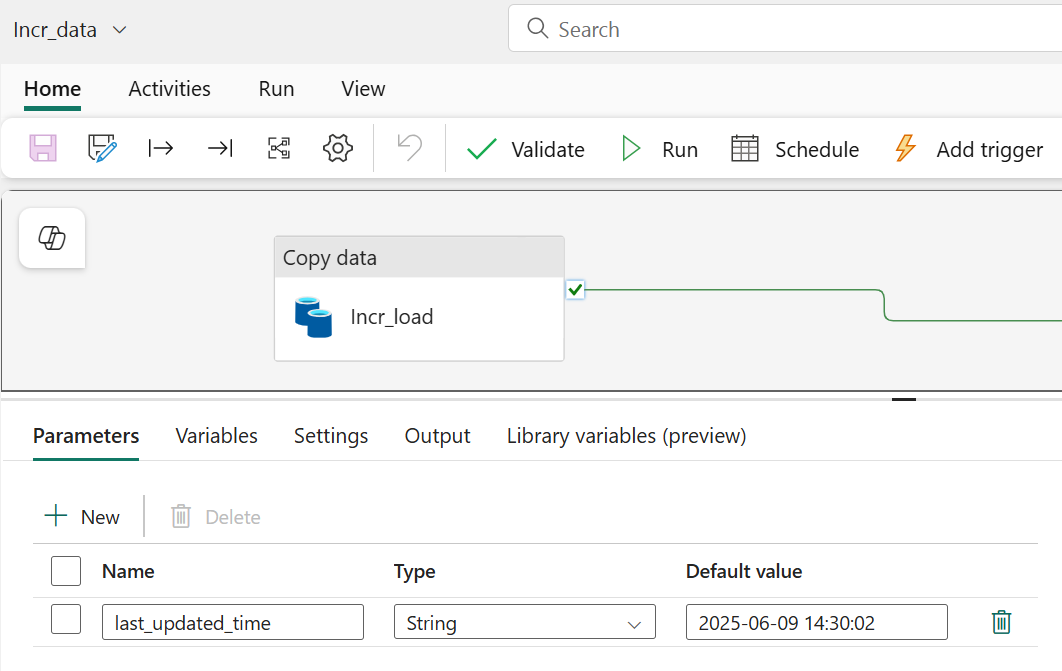
* Created a **Lakehouse** in Fabric with folders: Bronze, Silver, and Gold.
* Built **Pipeline 1** to copy raw full-load data from Oracle to Lakehouse Files (Bronze layer).

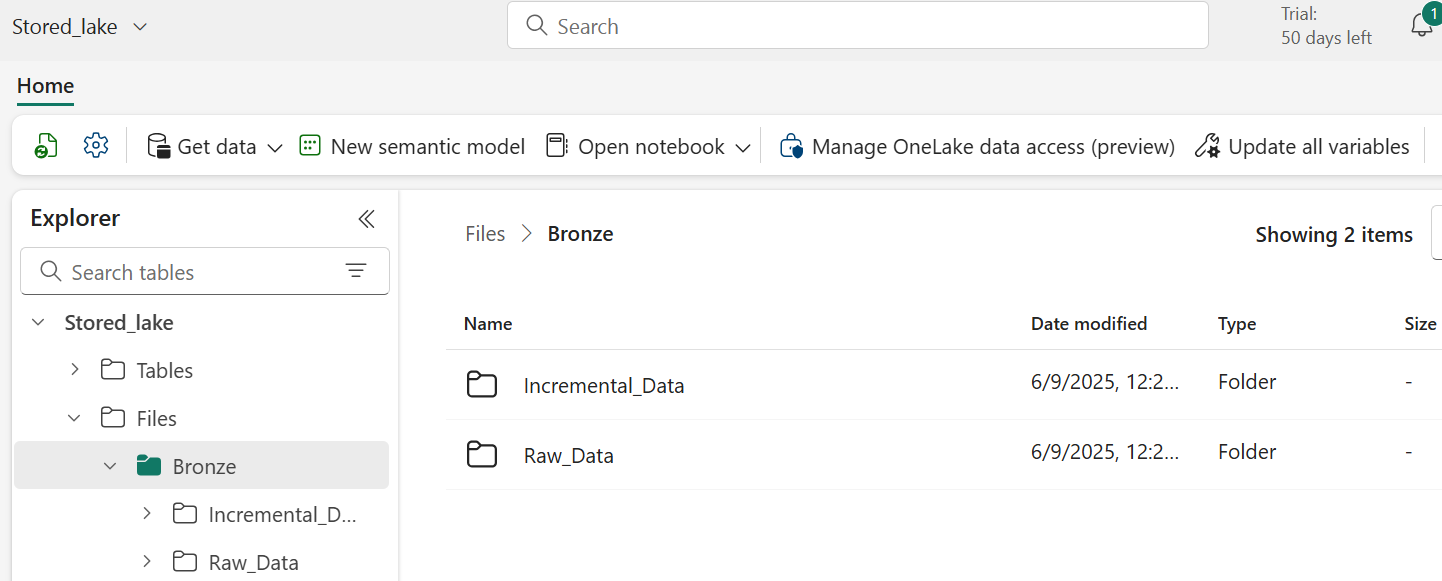






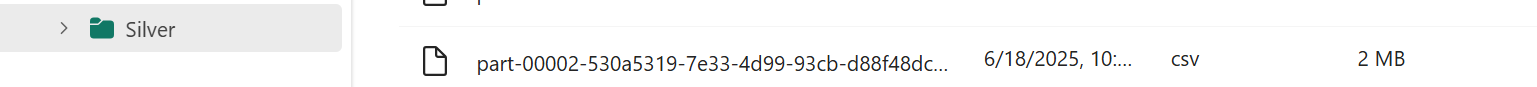
* Built **Pipeline 2** to load only **incremental data** using a **timestamp parameter** in the query.

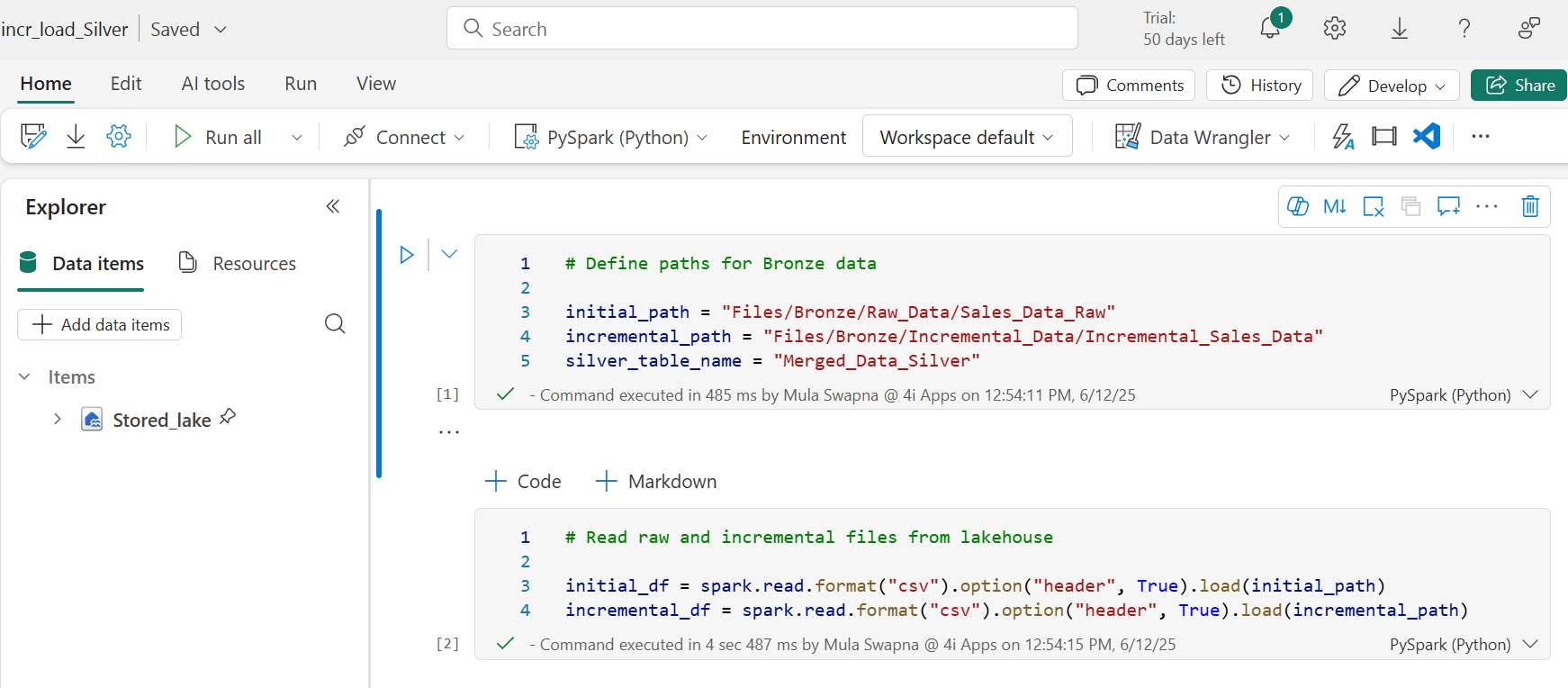


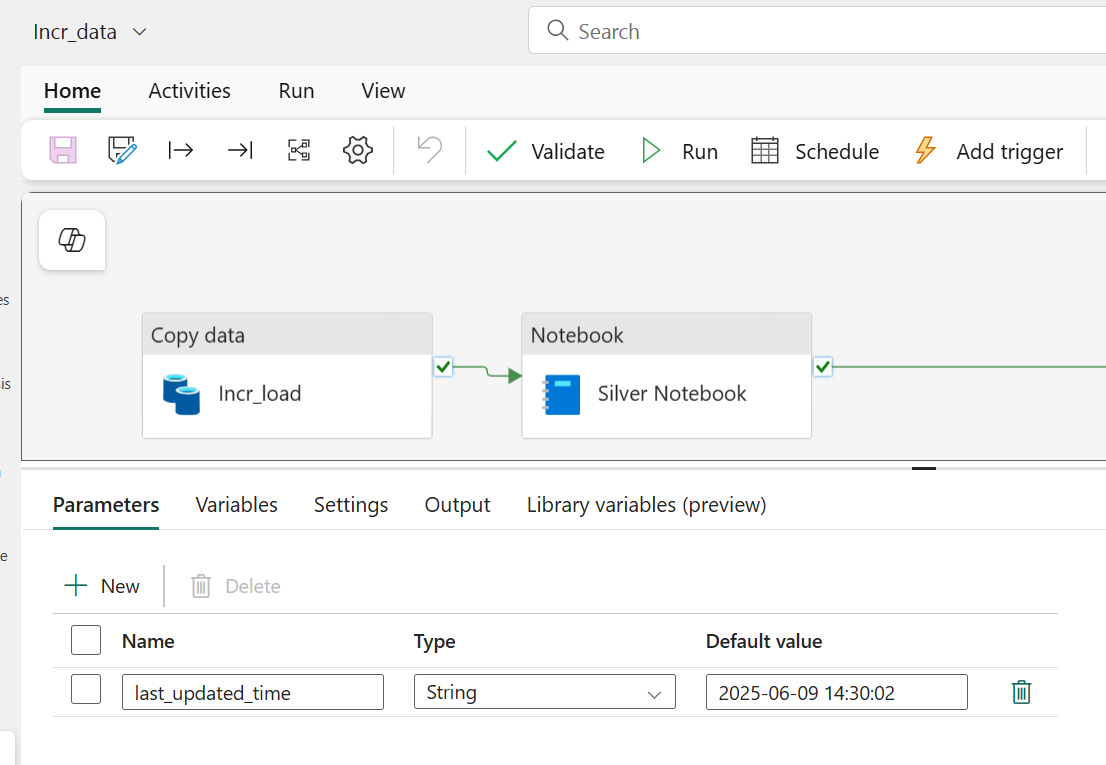


### **Step 3: Bronze to Silver Transformation**

* Created a **Notebook** to:
  + Read both raw and incremental files from Lakehouse.
  + Merge data using **PySpark**.
  + Deduplicate based on a primary key.
  + Write the cleaned dataset to **Silver layer**.

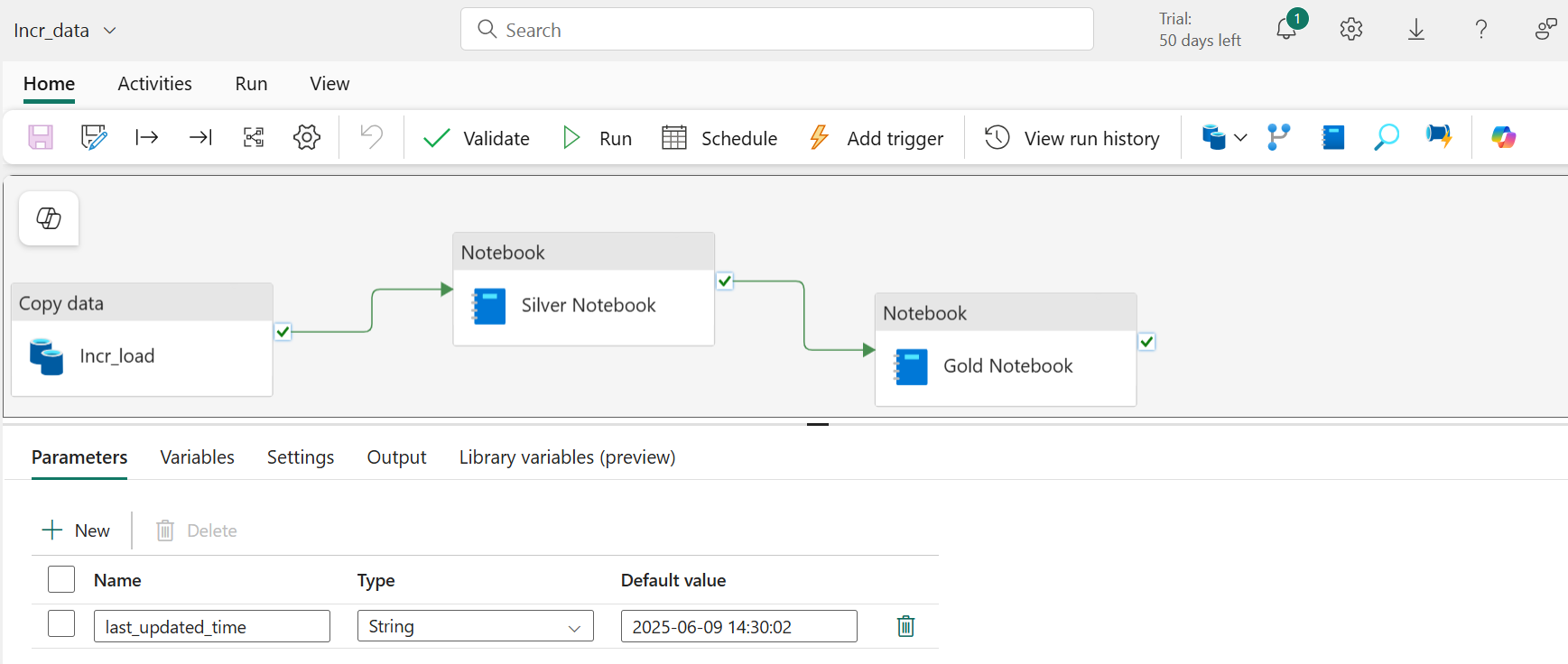


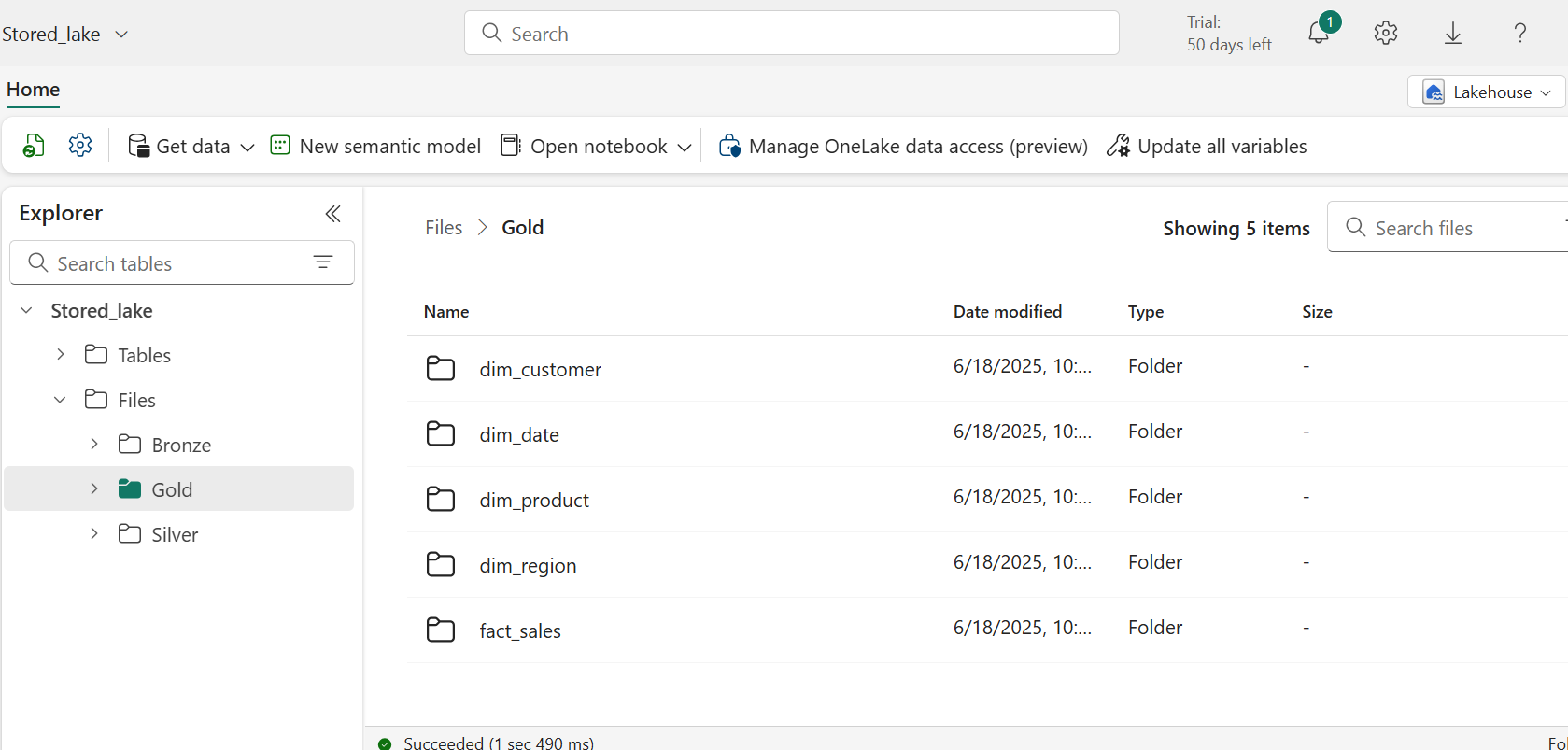


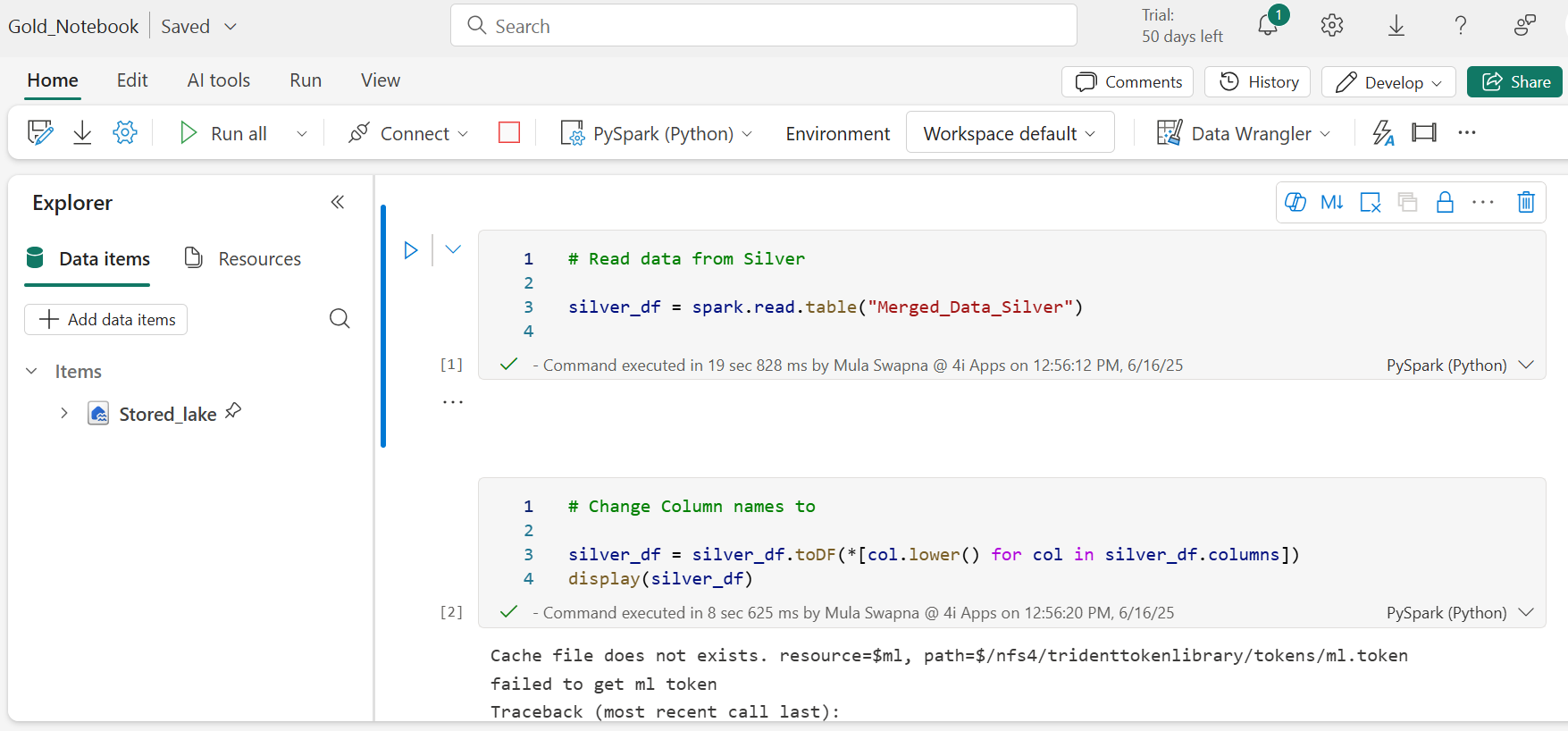


### **Step 4: Silver to Gold Transformation**

* Another **Notebook** for:
  + Changing column data types
  + Adding new calculated columns
  + Removing nulls
  + Splitting data into **Fact (Sales)** and **Dimensions (Product, Date, Customer, Region)**
* Saved outputs as both:
  + **Lakehouse Tables**
  + **Lakehouse Files** (with custom filenames like dim\_customer.csv)







### **Step 5: Power BI Reporting**

* Connected to **Lakehouse Tables** via SQL Analytics Endpoint.
* Built visuals for:
  + Sales by gender
  + Sales by product category
  + Filters for date, gender, category
* Published the report to Power BI Service.

### **Step 6: Alerts & Automation**

* Set up **Power BI data alerts** (e.g., email when sales cross a threshold).
* Used **Fabric pipeline triggers** to schedule daily incremental runs.

**Medallion Architecture in Action:**

|  |  |  |
| --- | --- | --- |
| **Layer** | **Purpose** | **Storage Format** |
| **Bronze** | Raw Ingested Data from Oracle | CSV in Lakehouse Files |
| **Silver** | Merged + Deduplicated Data | CSV in Lakehouse Files |
| **Gold** | Transformed Fact & dim Tables | Lakehouse Tables + CSVs |

This project demonstrates a **real-time incremental data pipeline** using **Microsoft Fabric**. By combining Oracle DB, PySpark notebooks, Lakehouse layers, and Power BI, it ensures efficient data flow, high data quality, and actionable business insights.