Sora Union Data Engineering Task

# Data Warehousing & ETL Process

The project provided us with two CSV files in a Google Sheet, in which we are expected to design a data warehouse and create an ETL process.

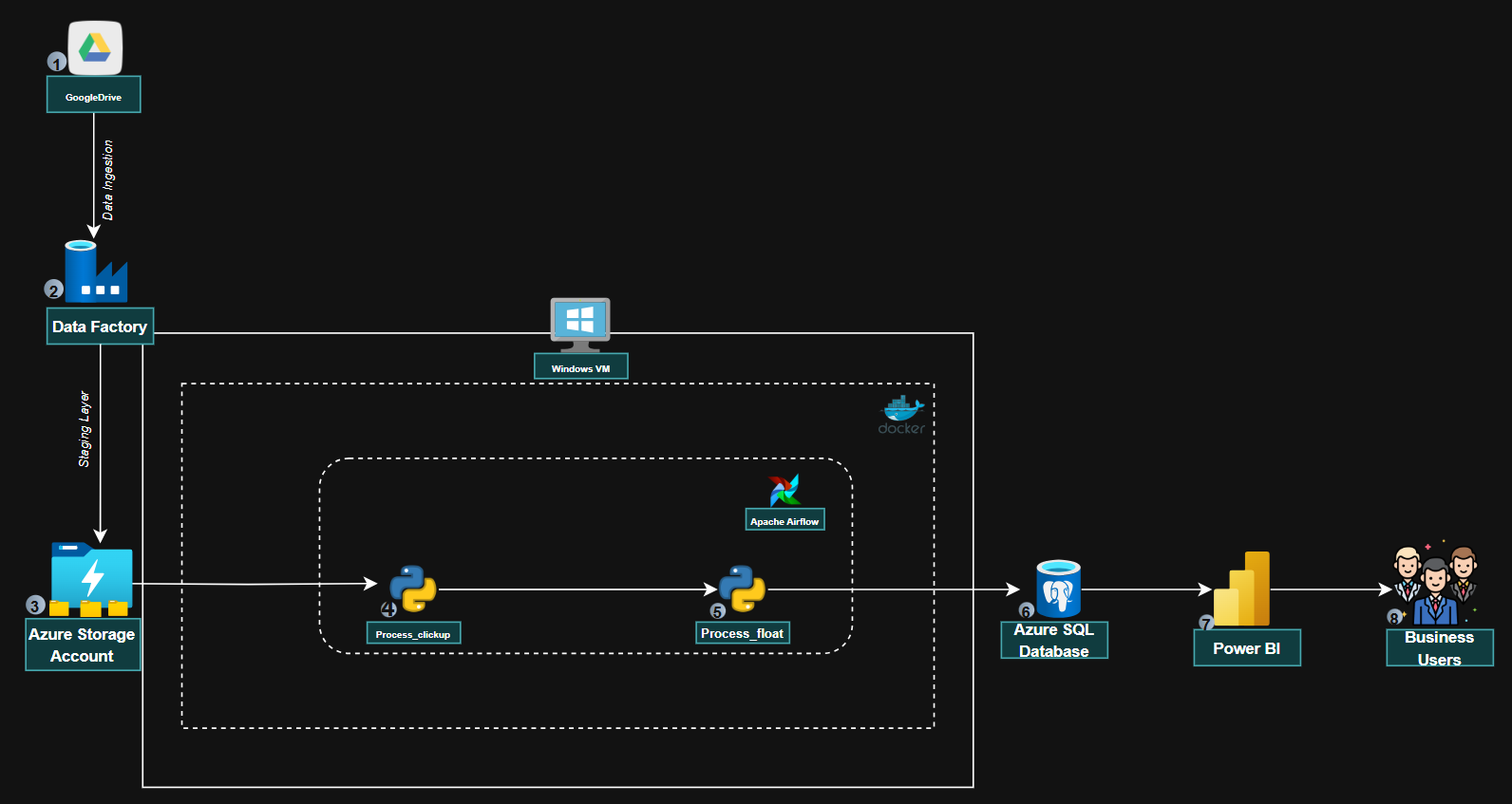
## Assumption

Assuming the data comes in a batch process daily to Google Drive and are ingested to a storage account like ***S3 bucket, Google Cloud Storage, or Azure Storage Account (ADLS)*** using ETL tools like Fivetran, AWS Glue, Azure Data Factory or Dataproc***.*** For this project, we will be using the ***Azure Storage Account, data Lake Gen 2*** for storing the ***Float and Clickup*** which will be ingested using the Azure Data Factory.

Since the data gets ingested to the storage account daily, we will be using the concept of getting the latest file (Last Modified Date) in the storage account, performing the necessary transformation and modeling before inserting it into the Azure PostgreSQL Database.

# Project Architecture

Based on our early assumption you will notice that data from the Google Drive are ingested to Azure Storage Account (adls) using the Azure Data Factory, we then created a ***DAG*** process in ***Apache Airflow*** for Orchestrating the whole process from ingestion of data to transformation and finally loading into Azure PostgreSQL Database (Flexible Server). This process was set to run at ***7:00 WAT/ GMT+ 1/ UTC +1*** daily



*Project Architecture*

# Data Modeling and Warehouse Design

For this reason, we are going to break it into two parts:

* *Table Creating with Relationship*
* *Entity Relationship Diagram*

## Table Creation

Following the Start Schema design approach we are going to break the data into two fact tables and five-dimension tables.

### Fact Tables

This contains the incremental tables that will continue to grow on a steady basis. Optimized for analytical queries.

* **fact\_time\_tracking:** Stores actual time entries from ClickUp
* **fact\_allocation:** Stores resource allocation/planning data from Float

### Dimension Tables

This is information about the fact table containing necessary information used in the modeling process.

* **dim\_client:** Client information
* **dim\_project:** Project details with client relationships
* **dim\_employee:** Employee information
* **dim\_role:** Role definitions
* **dim\_date:** Date dimension for time-based analysis

### Data Integrity

For Data Integrity we ensured following the best practice by creating references and using the indexing approach for query optimization.

**a) Referential Integrity:**

* Foreign key constraints on all dimension references
* Ensures data consistency across the warehouse

**b) Indexing Strategy:**

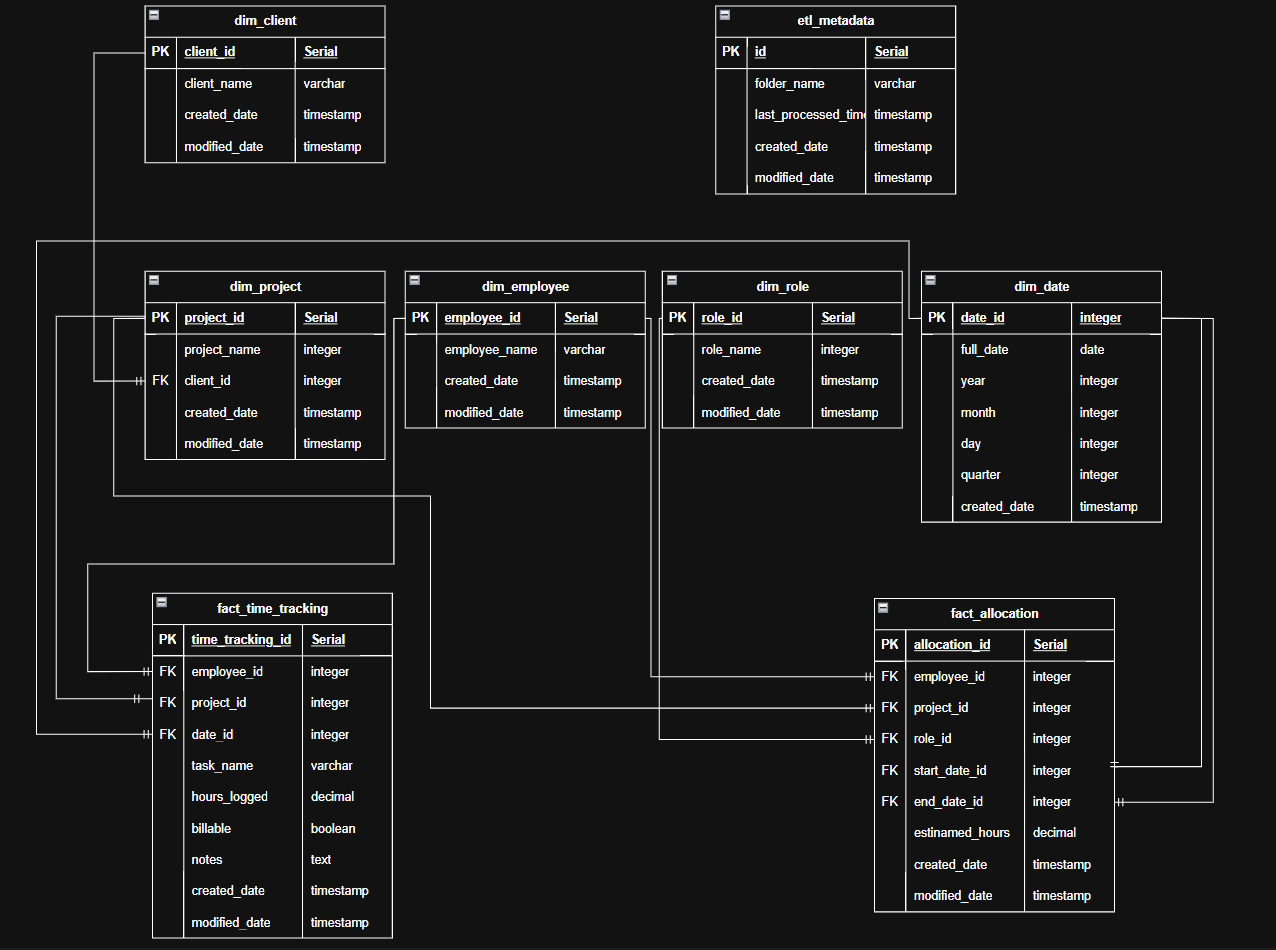
* Primary keys on all tables
* Foreign key indexes for joining optimization
* Composite index on allocation dates
* Created\_date and modified\_date tracking

**c) ETL Metadata**

These are standard data used in keeping records and changes.

## Entity Relationship Diagram

Entity-relationship (ER) modeling is a visual approach to data modeling used to represent the structure of a database. It is used to identify the "things" (entities) in a system and how they relate to each other.



*Project ER Diagram*

### One-to-Many Relationships:

* A client can have multiple projects (1:M)
* A project can have multiple time tracking entries (1:M)
* A project can have multiple allocations (1:M)
* An employee can have multiple time tracking entries (1:M)
* An employee can have multiple allocations (1:M)
* A role can be used in multiple allocations (1:M)
* A date can be referenced by multiple time tracking entries (1:M)
* A date can be the start or end date for multiple allocations (1:M)

### Fact Table Relationships:

**fact\_time\_tracking connects to:**

* dim\_employee
* dim\_project
* dim\_date

**fact\_allocation connects to:**

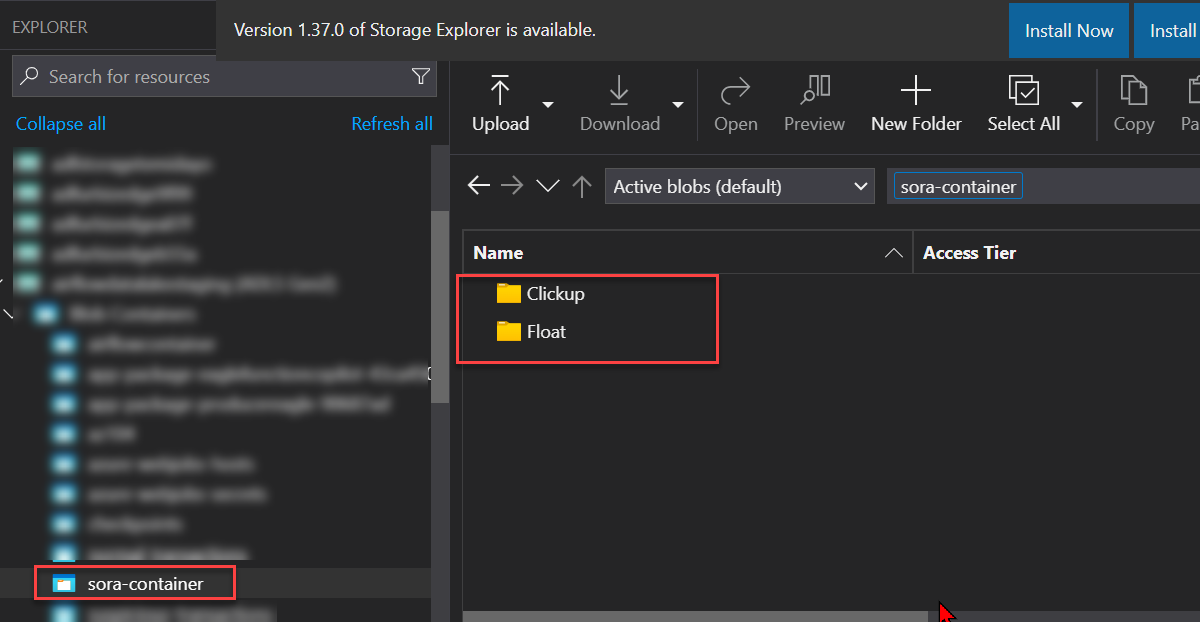
* dim\_employee
* dim\_project
* dim\_role
* dim\_date (twice, for start and end dates)

**Independent Tables:**

* etl\_metadata is independent and used for ETL process tracking

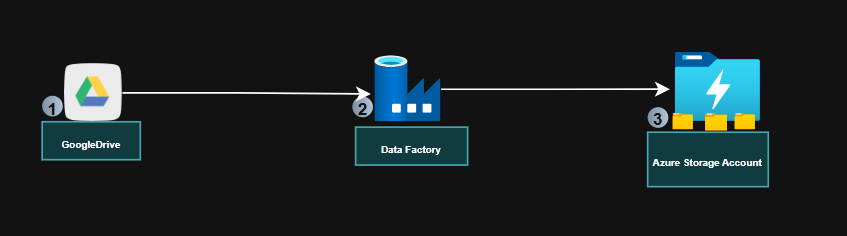
# ETL Process and Pipeline Development

This section focuses on the ETL process and pipeline development. Firstly let start by creating two empty folders in our ***container(prefix- adls).*** These are the containers where the ingestion process will start from.



## Data Ingestion with Data Factory

At this point we are going to create a pipeline using Azure Data Factory that will ingest the data from Google Drive and store it in our staging layer which is the adls.



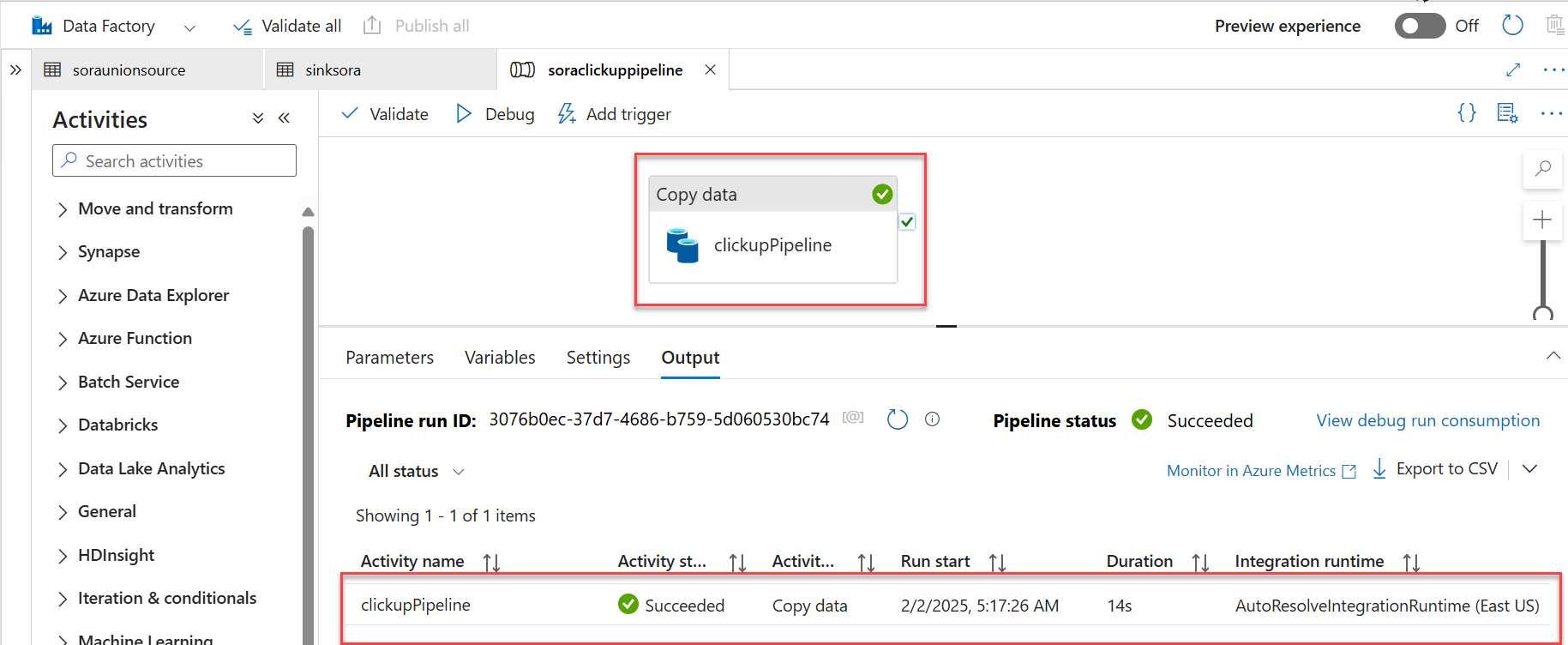
*Ingestion Phase*

## Pipeline Creation

In the Azure Data Factory the following components are created to achieve this.

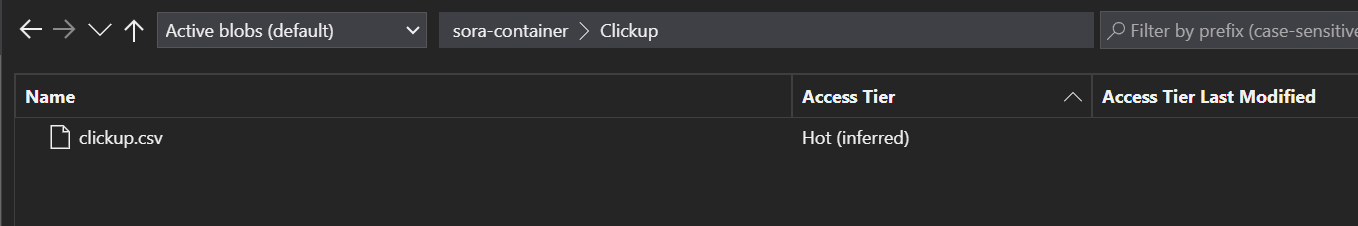
* ***Source Dataset***
* ***Source Linked Service***
* ***Activity (Copy Activity - Google Drive to adls)***
* ***Sink Linked Service***
* ***Sink Dataset***

From the image below you will notice the data copied successfully from Google Drive to ADLS which serves as our storage layer.



*Successful Pipeline Run*

We can confirm the data in the storage account.



## Transformation and Loading Preparation

At this stage we will be developing the transformation logic and creating and orchestration process using Apache Airflow

## Pre-requisite

* ***Docker Desktop (Using Windows)***
* ***Virtual Machine/ EC2 Instant (Windows Server)***
* ***WSL - Windows Subsystem for Linux (Applicable for only Windows Users)***
* ***Preferred IDE - VSCode***

## Setting Airflow on Docker

By going through the official documentation on Airflow Docker was a straight process.

<https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#running-airflow-in-docker>

### Folder Breakdown

**Sora Union/**

>> **config**

*-- .env*

*-- config.py*

>> **dags**  
*-- etl\_dag.py*

*-- etl\_process.py*

*-- utils.py*

>> **logs**

>> **plugins**

* **.env:** Environment variables and sensitive configuration. Stores sensitive configuration like:

Azure Storage credentials, PostgreSQL database connection details.

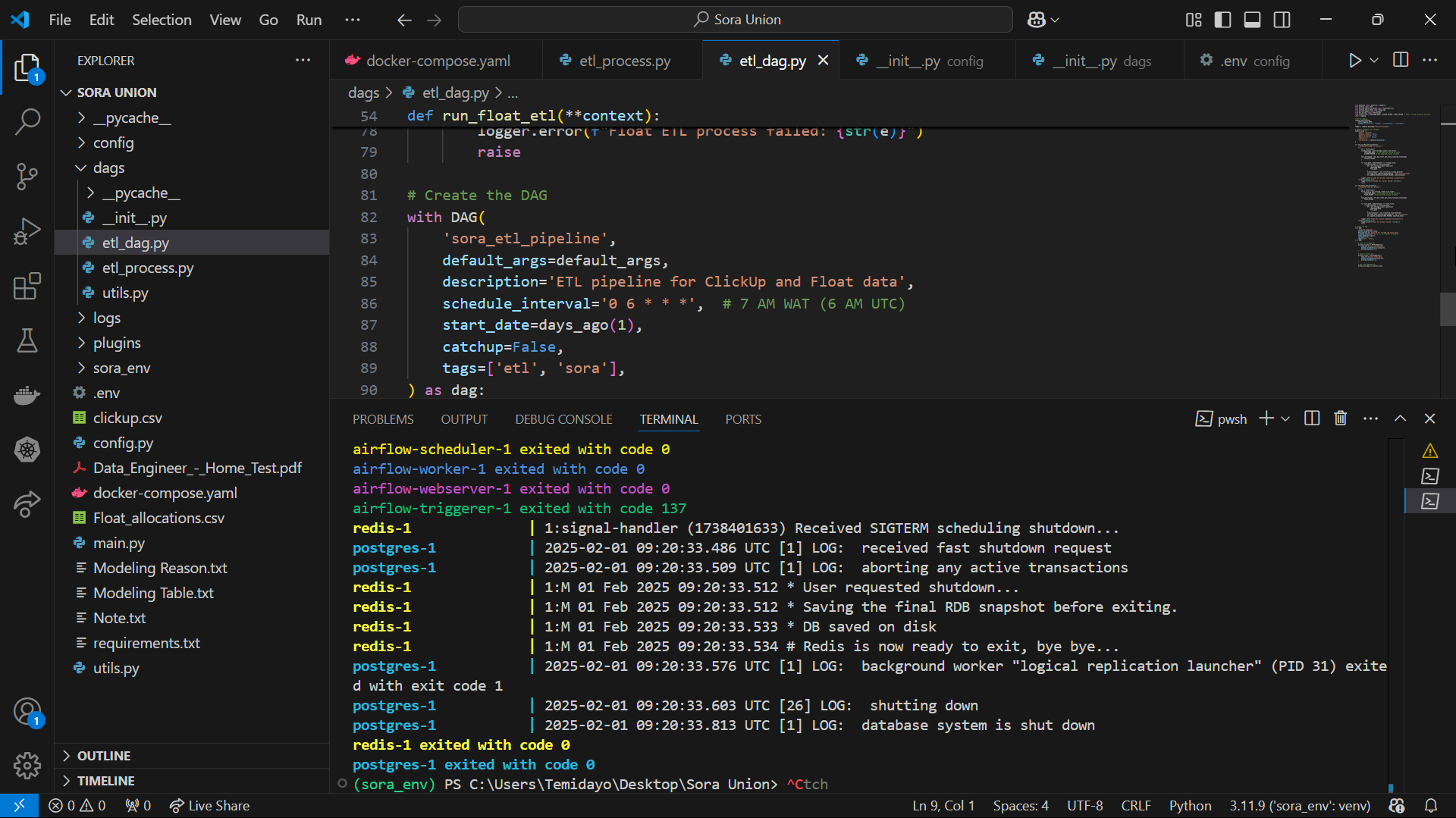
* **config.py:** Configuration loader and settings management. Loads environment variables and defines constants used across the application.
* **etl\_dag.py:** Airflow DAG definition and task orchestration.
  + **process\_clickup:** Processes ClickUp time tracking data
  + **process\_float:** Processes Float resource allocation data
  + Configures scheduling (runs daily at 6 AM UTC)
* **etl\_process.py:** Core ETL transformation logic. Contains the ETLProcess class which handles:
  + Dimension table management (clients, projects, employees, roles)
  + Data transformation logic for both ClickUp and Float data
  + Fact table loading for time tracking and resource allocation
  + Incremental loading through metadata tracking
* **utils.py:** Utility classes for Azure and PostgreSQL connections.

**Provides utility classes:**

* + **AzureStorageClient:**
  + Handles connections to Azure Data Lake Storage
  + Manages file listing and reading
  + **PostgresClient:**
  + Manages database connections
  + Provides methods for querying and data loading

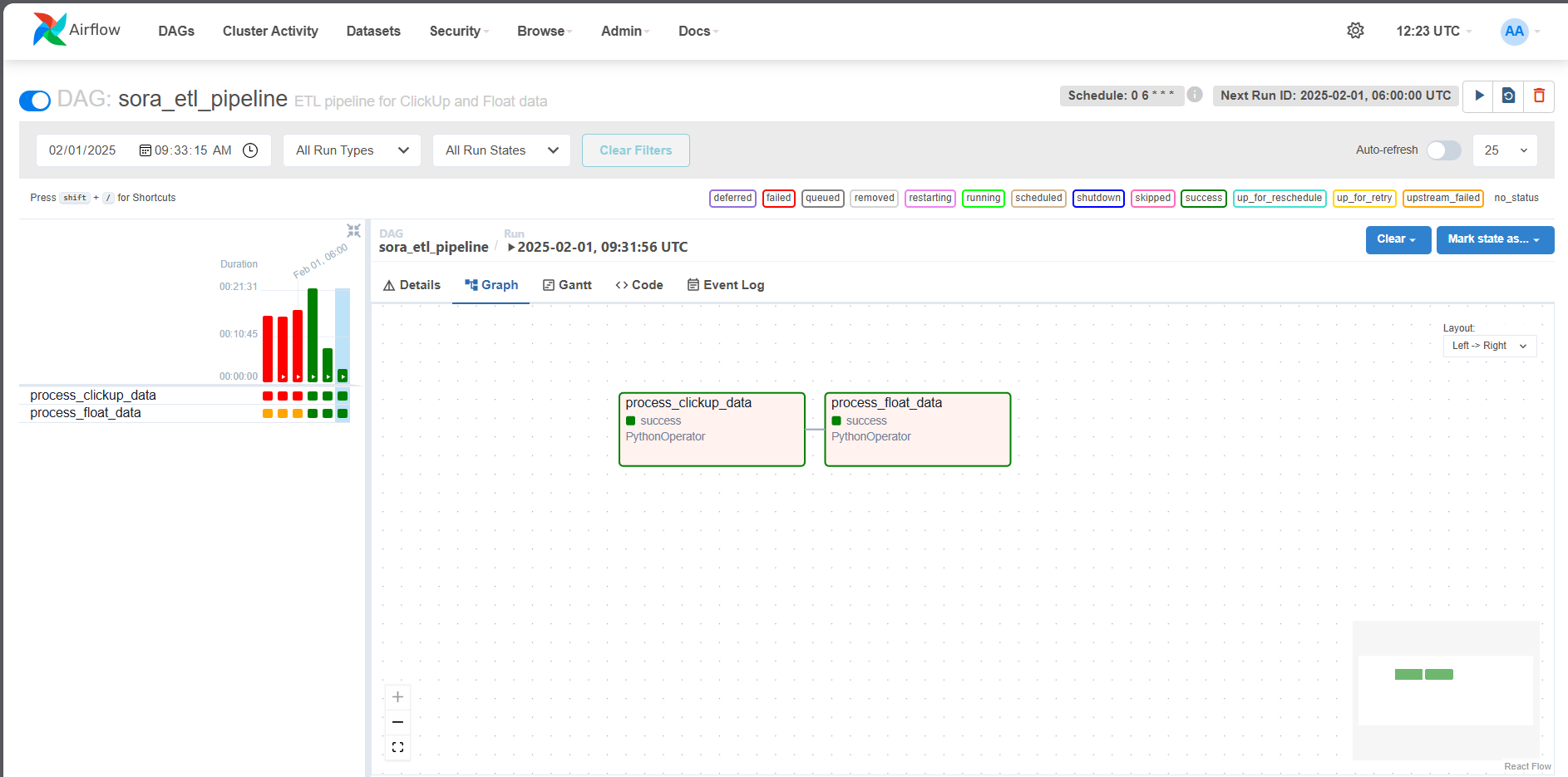
## Airflow Orchestration

After successfully setting up your code you can run it on the Airflow Webserver UI and see it works as expected.



docker-compose up

**Process\_clickup\_data >> process\_float\_data**

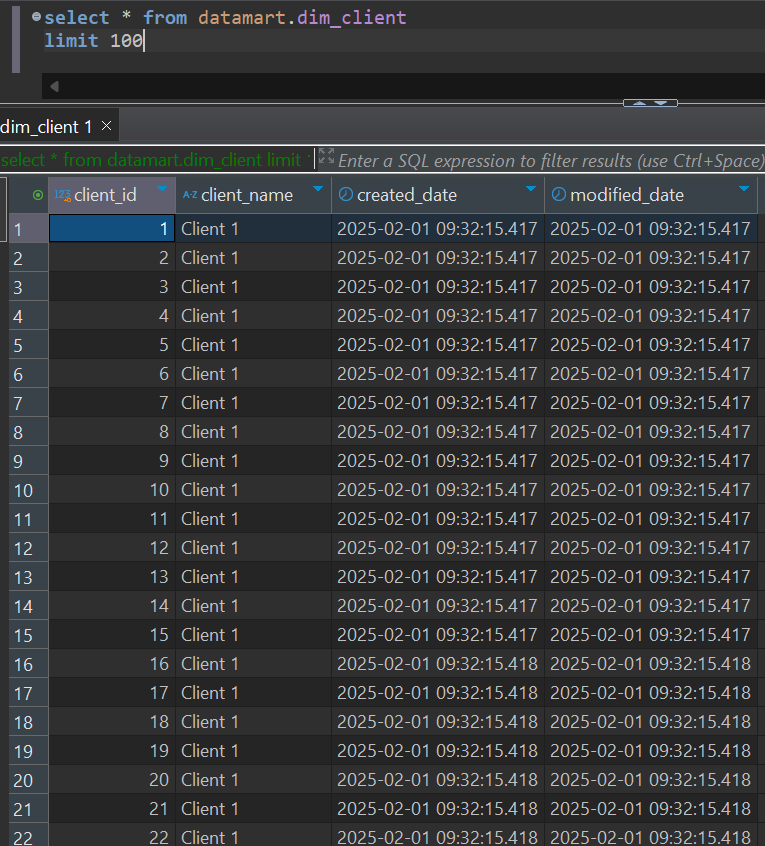


*Airflow Orchestration*

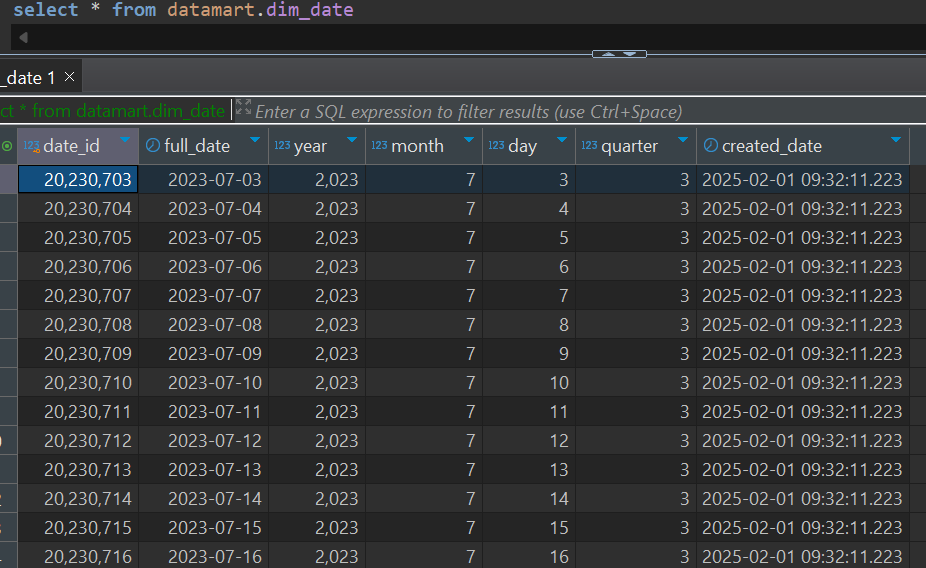
# Test and Validate ETL Process

After successfully running the entire ETL process we can test it by using the SELECT statement in PostgreSQL Database.

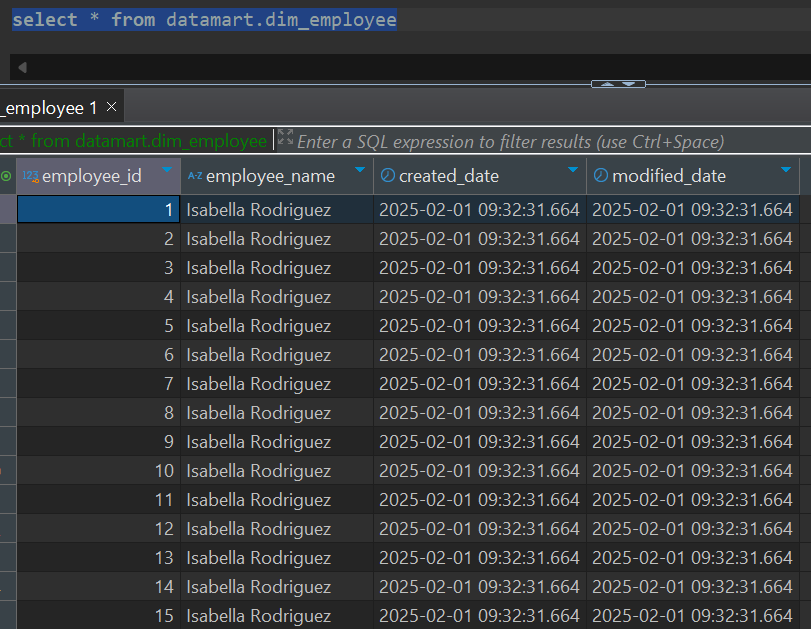
**select** \* **from** datamart.dim\_client

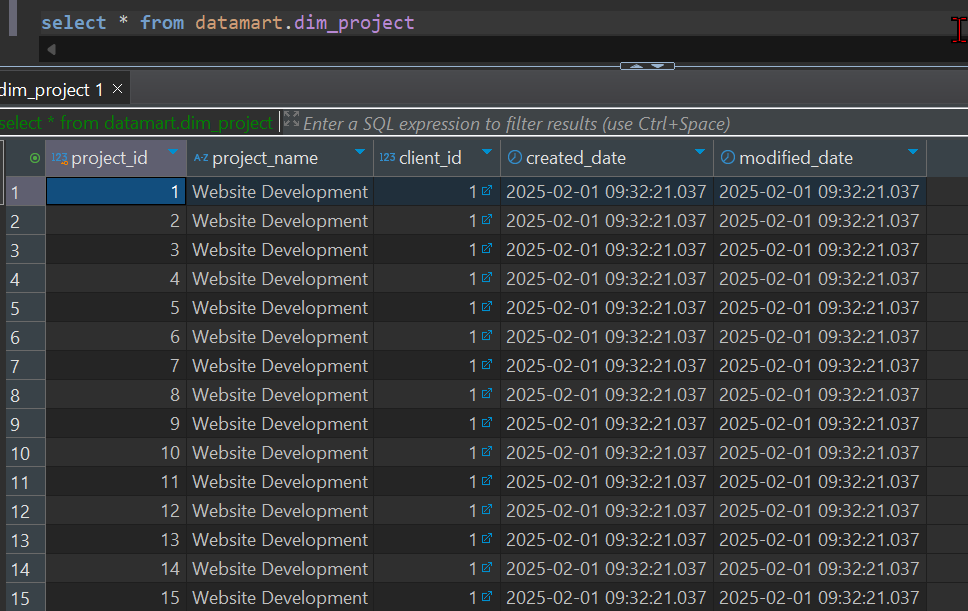


**select** \* **from** datamart.dim\_date

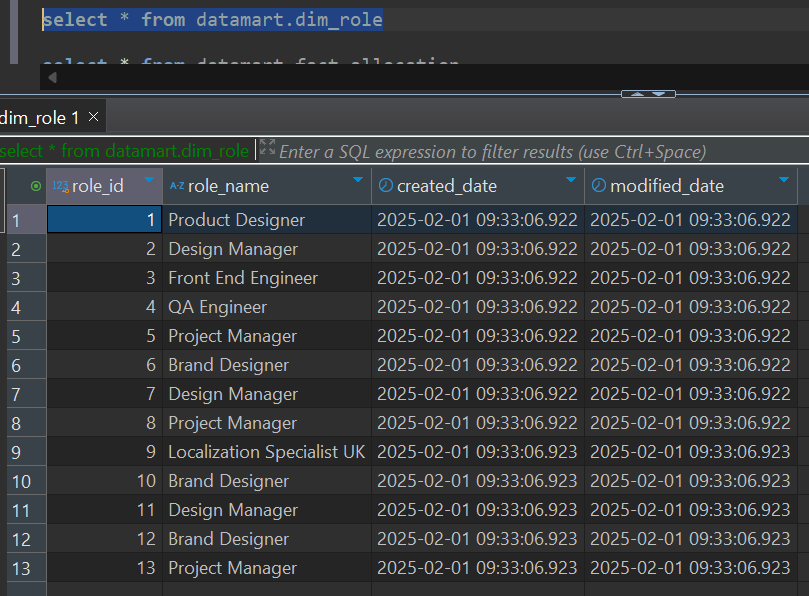


**select** \* **from** datamart.dim\_employee

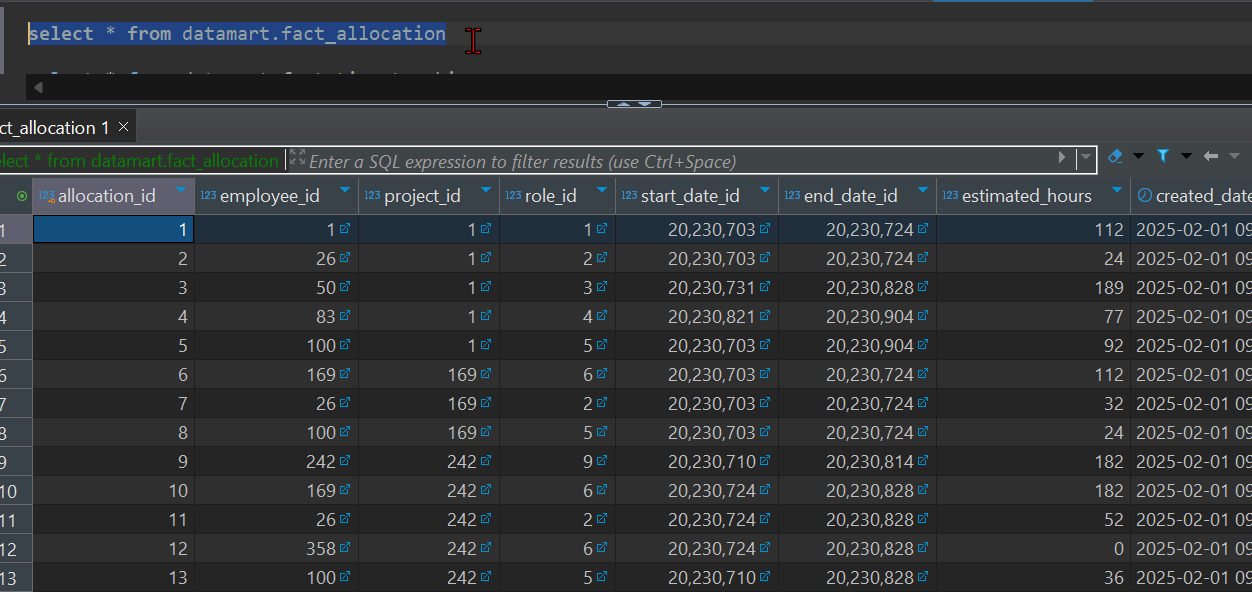


**select** \* **from** datamart.dim\_project  


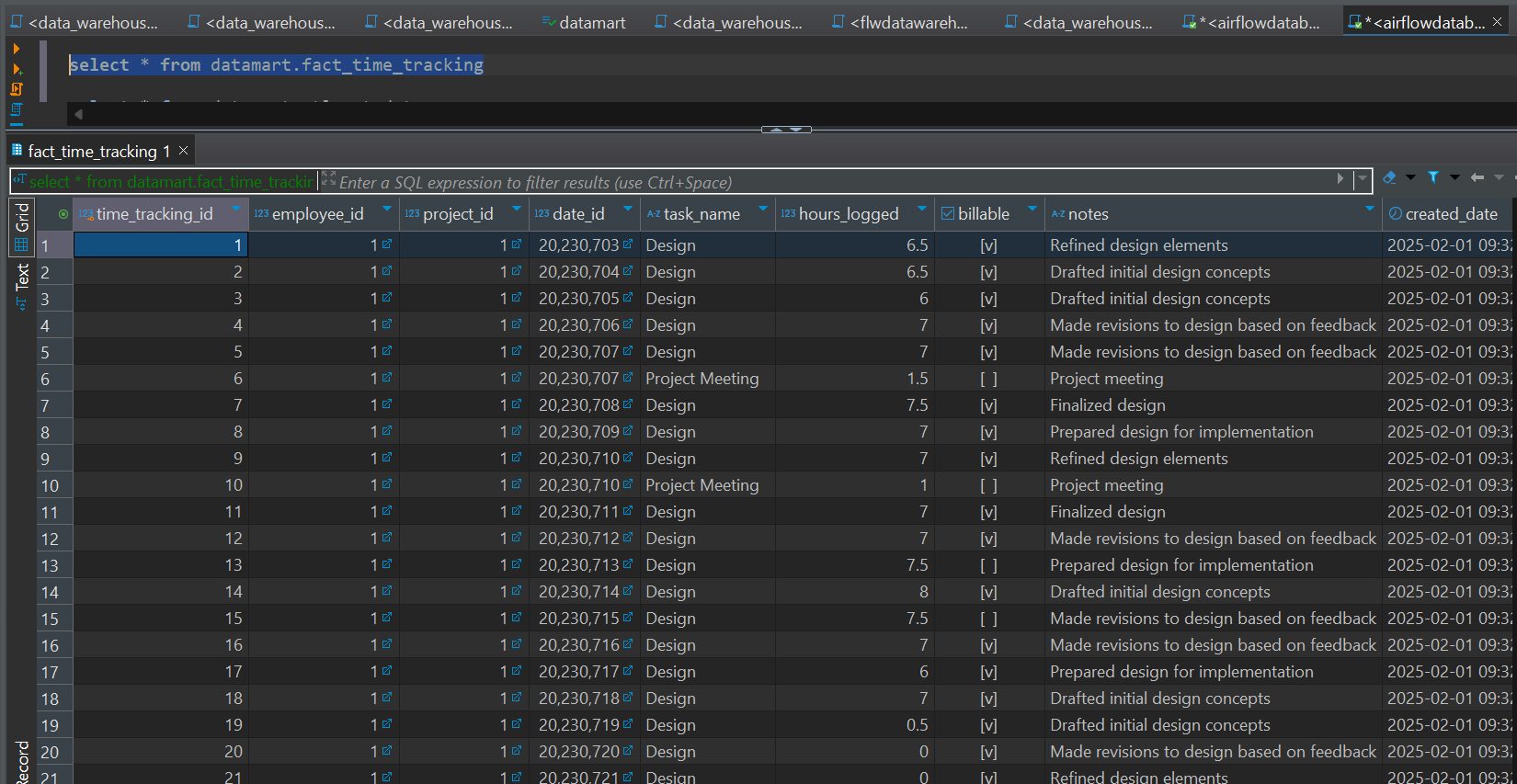
**select** \* **from** datamart.dim\_role



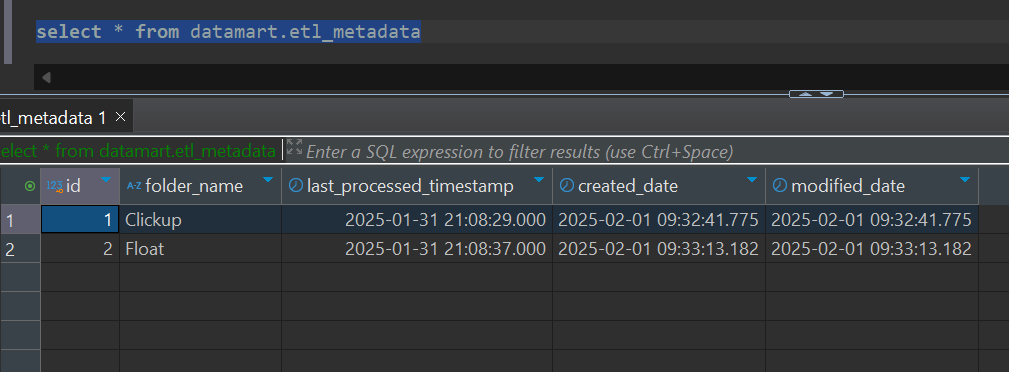
**select** \* **from** datamart.fact\_allocation



**select** \* **from** datamart.fact\_time\_tracking



**select** \* **from** datamart.etl\_metadata



# Conclusion

To get a high level of accuracy the following process where followed, having designed the model, had a high level of data quality change and improved the query by optimizing using index on certain columns.