**ETL Pipeline Documentation**

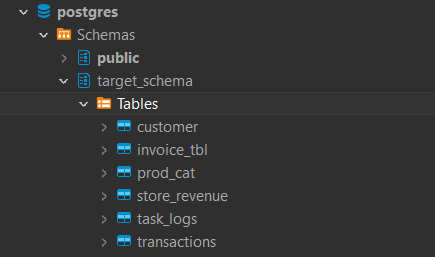
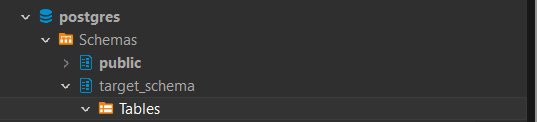
**Flow of ETL pipeline**

Loaded the data into Postgres DB by integrating it.

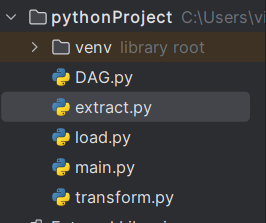
Data Extraction from CSV files to pandas DF.

Transformed & cleaned the pandas DF data.

Firstly, I have created a target schema in “postgres” DB and named it as “target\_schema”. I have used “DBEAVER” IDE for querying data and creation of tables into postgres DB.



Then, I have created 3 different python scripts for extraction, transformation and loading purpose using Pycharm IDE. And then I have created another script named as “main” which has a function “run\_etl” which I am using to call my ETL script’s functions.



In each of the file I have added comments so that anyone can get through it easily.

**Function of Extract:**

Into the **“extract.py”** what I am doing is that I have defined two functions into it one for setting up the connection with **postgres DB** named as **“con”** and another function defined is “**extract\_data**” that is being used to extract the data from the CSV files into **pandas DF**.

I have created a **json config** file so that I can pass the csv file paths dynamically. Then I’m reading the json file using the json library.

After that I’m reading the CSV files into the pandas DF. And then logging the status of extraction into the **“task\_logs”** table that has been created by me in **postgres DB**.

I have used **TRY** and **EXCEPT** so that I can raise the exception in case of any failure.

**Function of Transform:**

**“transform.py”** script is being used for the transformation of the data present in the **Pandas DF.**

Firstly, I have imported all the necessary libraries and files and then I have defined the function **“transform\_data”**, in which I have created lists of columns present in all of the pandas DF and then I am renaming it to lowercase so that there should not be any case-sensitivity issue.

Further, I’m filling NaN values using fillna() function because I don’t want to drop that NaN values.

And then for quantitative fields I don’t want any negative values and ideally the quantitative fields like quantity, rate and tax could not be negative so I have used “abs()” function to get the absolute values. After that I have removed duplicate data using “drop\_duplicates()” function to ensure data redundancy.

And then I have re-formatted date columns into the required format.

I did only required data transformations, further I can also add few more things into the transformation as per the need. And then I can refactor it easily.

Transformation part also has the logging part in case of success or failure of the job.

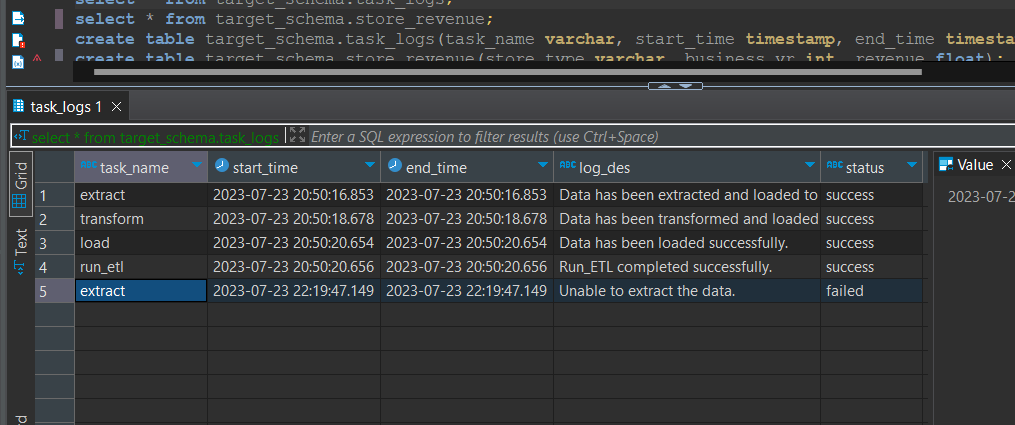
**Function of Load:**

Into the “**load.py”** script I just connected the **postgres** **DB** and **SQLAlchemy** engine to create the tables and insert the data into target tables.

Firstly, I have created 3 different tables(**customer table, prod\_cat table and transactions table**) in **postgres DB** from the data of **source DF**.

And then I am using these 3 tables as source tables for my target tables. I have created two target tables namely **invoice\_tbl** and **store\_revenue**. I wanted the data in certain data type and only the necessary data so I have written respective SQL queries to perform the insertion of the data into the target tables.

And this function load\_data also has the logging part for logging the data in case of failure or success. And by querying the logs from “**task\_logs**” table anyone can get the logs by stages.



I’m calling all of these functions into “main.py” file.

**DAG function:**

I have an airflow DAG which is scheduled on daily basis @7:00 pm.

Where I have defined 3 tasks Start, run\_etl and End respectively.



