**Sai Varun Kumar Namburi**

**Assignment – 5**

**Code Repo Link:** [**https://github.com/SaivarunNamburi/Customers-Debit-PurchasesDataProcessing**](https://github.com/SaivarunNamburi/Customers-Debit-PurchasesDataProcessing)

Steps involved in the completion of this assignment:

Step 1:

1. Created a Python script that generates mock transaction data.
2. Each record should include customer\_id, name, debit\_card\_number, debit\_card\_type, bank\_name, transaction\_date, and amount\_spent.

A screen shot of a computer

Description automatically generated

Step 2: Upload the data to S3

1. Created an AWS S3 bucket for storing the daily transaction CSV files.
2. Upload the daily CSV files to the S3 bucket using Python Script, and utilized a Hive-style partitioning scheme like "date=yyyy-mm-dd" in the S3 bucket structure to organize the data by date.
3. Execute the Python script periodically, generating and uploading new CSV files for each day's data.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 3:

1. Created a Postgres database instance in Amazon RDS
2. Design and create a table to store aggregated transaction data.

**Create Table Script**:

create table customer\_debitcard\_puchases(

customer\_id Int primary key,

debit\_card\_number varchar(255),

bank\_name varchar(255),

total\_amount\_spend float

);

A screenshot of a computer

Description automatically generated

Step 4:

1. Created Crawlers in AWS Glue, to read and store the metadata in the Metadata database
2. We need 2 crawlers here, one is for Source and one for Target which stores the metadata in the database.

Crawler for S3 datasource

A screenshot of a computer

Description automatically generated

Crawler for Postgres Database

A screenshot of a computer

Description automatically generated

Glue Database for S3 data source which contains metadata

A screenshot of a computer

Description automatically generated

Glue Metastore for Postgres Database

A screenshot of a computer

Description automatically generated

To establish a successful connection for Postgres in Glue, there needs to be attached S3 permissions within the VPC endpoints like below

A screenshot of a computer

Description automatically generated

After the connection is successful, then you can run a Crawler for the PostgreSQL database

A screenshot of a computer

Description automatically generated

After everything was set up, now I created an ETL job using visual ETL from the dashboard

1. Add the Data source
2. Add aggregation transform
3. Change Schema based on your destination
4. Connect with the target database

A screenshot of a computer

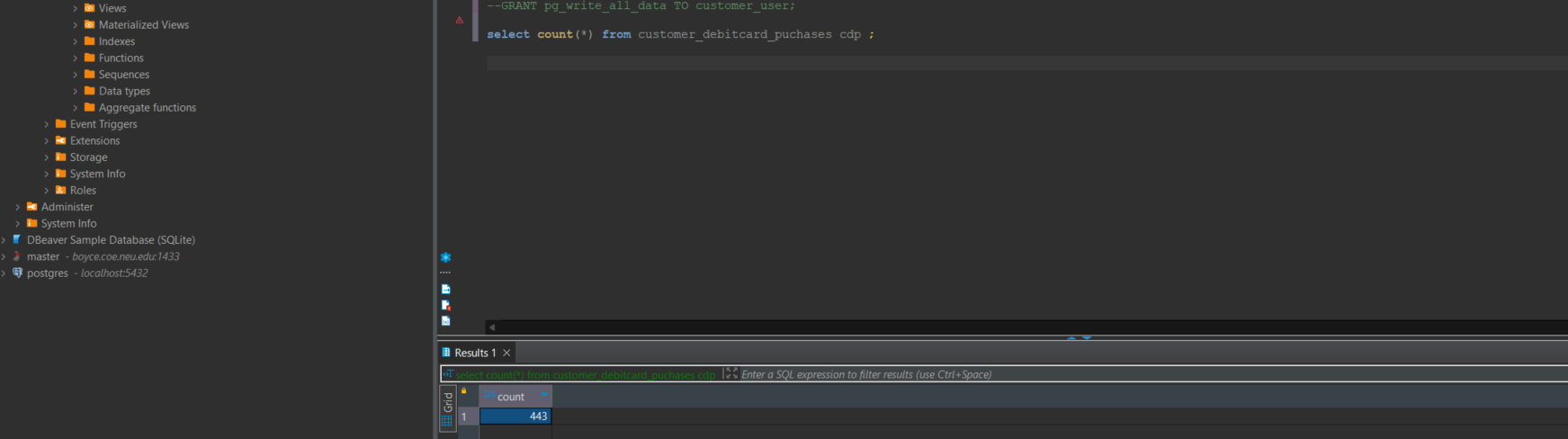
Description automatically generated

Go to jobdetails section in the GUI, then enable the Job bookmark part, which will help us for the incremental load of the daily data into postgres

A screenshot of a computer

Description automatically generated

Before Incremental Load, it has only 443 records



Below is the current Hive structure after adding today's data

A screenshot of a computer

Description automatically generated

Again we need to run the crawler to identify if we have any changes

A screenshot of a computer

Description automatically generated

Once again run the ETL job, then see if the new data has been added or not. I have verified it by running count(\*) and 4 new records have been added

A screen shot of a computer

Description automatically generated

**Data Processing Architecture Diagram**:

A screenshot of a computer

Description automatically generated

**Summary**:

The process encompasses setting up a streamlined data pipeline to manage daily transaction data. This involves generating mock transaction data in CSV format, organizing and storing it within an S3 bucket using a Hive-style partitioning approach. Additionally, an RDS PostgreSQL instance is established, equipped with a structured table schema for aggregating transactional data. Furthermore, an AWS Glue job is developed to handle incremental data processing, seamlessly integrating with both the S3 bucket and the PostgreSQL database. This orchestrated workflow ensures efficient management and processing of transactional data, facilitating seamless updates and data integrity.