

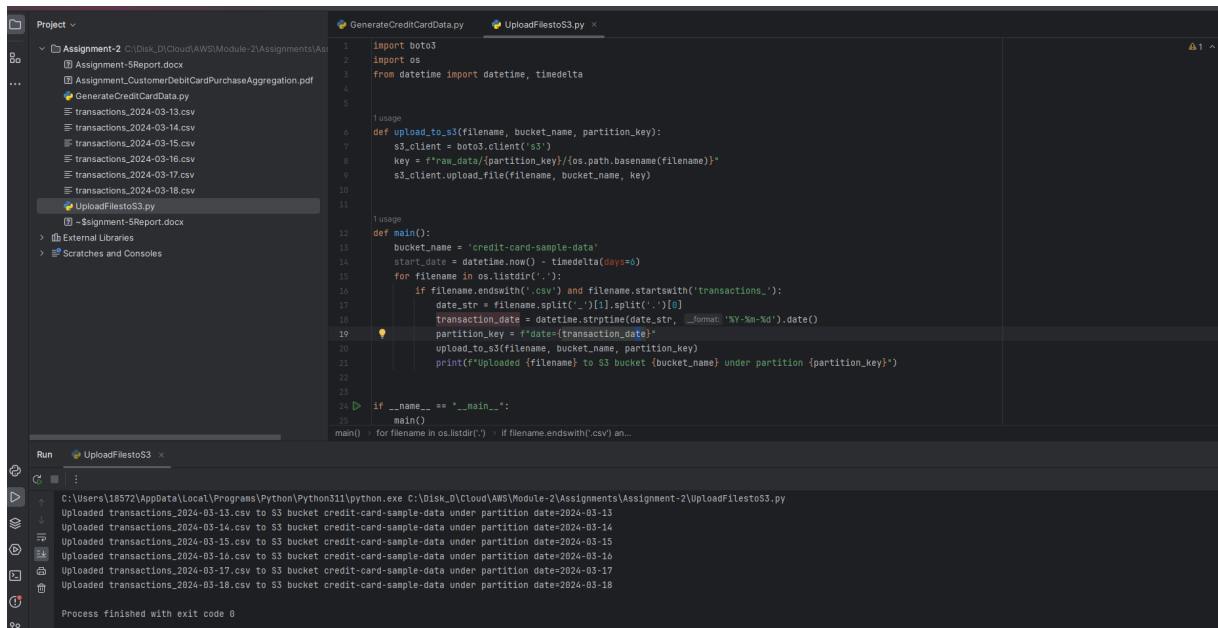
Assignment – 5

Code Repo Link: <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.



```
Project
├── Assignment-2
│   ├── Assignment-5Report.docx
│   ├── Assignment-CustomerDebitCardPurchaseAggregation.pdf
│   ├── GenerateCreditCardData.py
│   ├── transactions_2024-03-13.csv
│   ├── transactions_2024-03-14.csv
│   ├── transactions_2024-03-15.csv
│   ├── transactions_2024-03-16.csv
│   ├── transactions_2024-03-17.csv
│   ├── transactions_2024-03-18.csv
│   ├── UploadFiletoS3.py
│   └── Assignment-5Report.docx
├── External Libraries
└── Scratches and Consoles

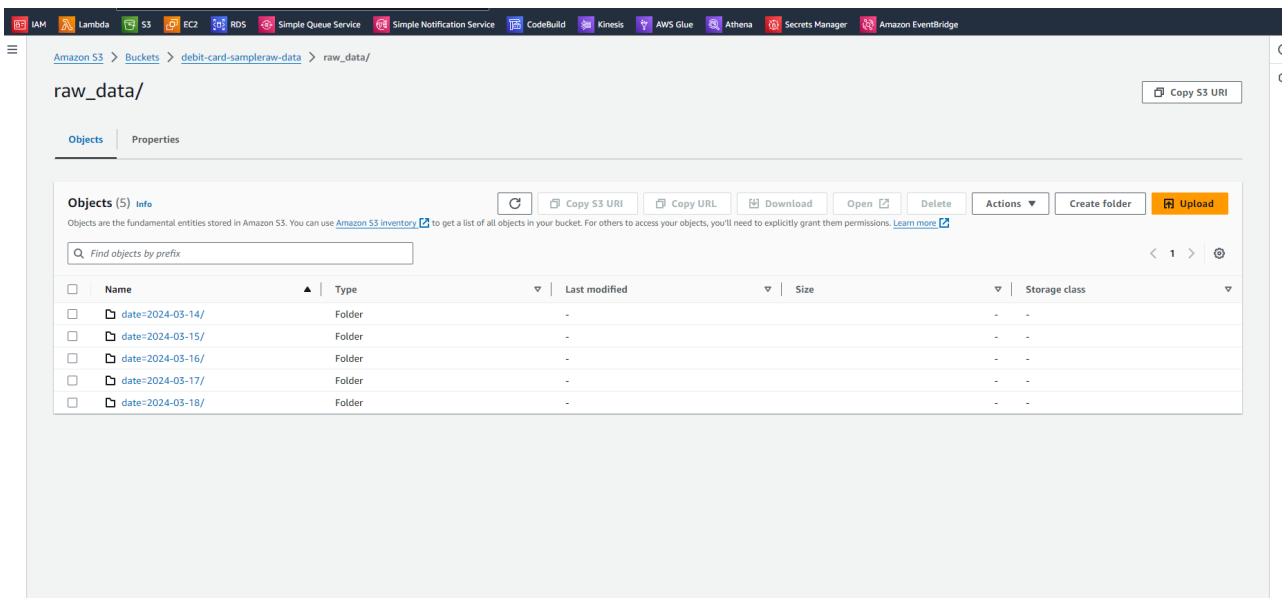
GenerateCreditCardData.py
1 import boto3
2 import os
3 from datetime import datetime, timedelta
4
5
6
7 usage
8
9 def upload_to_s3(filename, bucket_name, partition_key):
10     s3_client = boto3.client('s3')
11     key = f'raw_data/{partition_key}/{os.path.basename(filename)}'
12     s3_client.upload_file(filename, bucket_name, key)
13
14
15 usage
16
17 def main():
18     bucket_name = 'credit-card-sample-data'
19     start_date = datetime.now() - timedelta(days=5)
20     for filename in os.listdir('.'):
21         if filename.endswith('.csv') and filename.startswith('transactions_'):
22             date_str = filename.split('_')[1].split('.')[0]
23             transaction_date = datetime.strptime(date_str, '%Y-%m-%d').date()
24             partition_key = f'date={transaction_date}'
25             upload_to_s3(filename, bucket_name, partition_key)
26             print(f'Uploaded {filename} to S3 bucket {bucket_name} under partition {partition_key}')
27
28
29 if __name__ == '__main__':
30     main()
31
32 main() for filename in os.listdir('.') if filename.endswith('.csv') an...
```

```
Run
UploadFiletoS3
C:\Users\18572\AppData\Local\Programs\Python\Python311\python.exe C:\Disk_D\Cloud\AWS\Module-2\Assignments\Assignment-2\UploadFiletoS3.py
Uploaded transactions_2024-03-13.csv to S3 bucket credit-card-sample-data under partition date=2024-03-13
Uploaded transactions_2024-03-14.csv to S3 bucket credit-card-sample-data under partition date=2024-03-14
Uploaded transactions_2024-03-15.csv to S3 bucket credit-card-sample-data under partition date=2024-03-15
Uploaded transactions_2024-03-16.csv to S3 bucket credit-card-sample-data under partition date=2024-03-16
Uploaded transactions_2024-03-17.csv to S3 bucket credit-card-sample-data under partition date=2024-03-17
Uploaded transactions_2024-03-18.csv to S3 bucket credit-card-sample-data under partition date=2024-03-18
Process finished with exit code 0
```

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.

```
1 import boto3
2 import os
3 from datetime import datetime, timedelta
4
5
6 usage
7 def upload_to_s3(filename, bucket_name, partition_key):
8     s3_client = boto3.client('s3')
9     key = f"raw_data/{partition_key}/{os.path.basename(filename)}"
10    s3_client.upload_file(filename, bucket_name, key)
11
12 usage
13 def main():
14     bucket_name = 'debit-card-sampleraw-data'
15     for filename in os.listdir('.'):
16         if filename.endswith('.csv') and filename.startswith('transactions_'):
17             date_str = filename.split('_')[1].split('.')[0]
18             transaction_date = datetime.strptime(date_str, '%Y-%m-%d').date()
19             partition_key = f"date={transaction_date}"
20             upload_to_s3(filename, bucket_name, partition_key)
21             print(f"Uploaded {filename} to S3 bucket {bucket_name} under partition {partition_key}")
22
23 if __name__ == "__main__":
24     main()
25
```

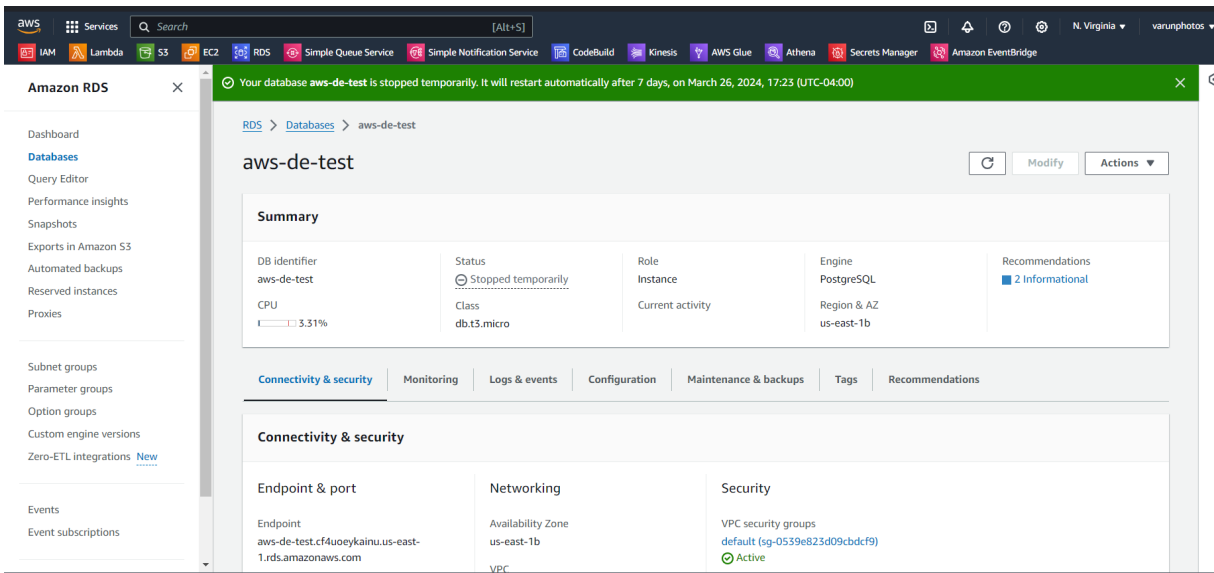


Step 3:

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

Create Table Script:

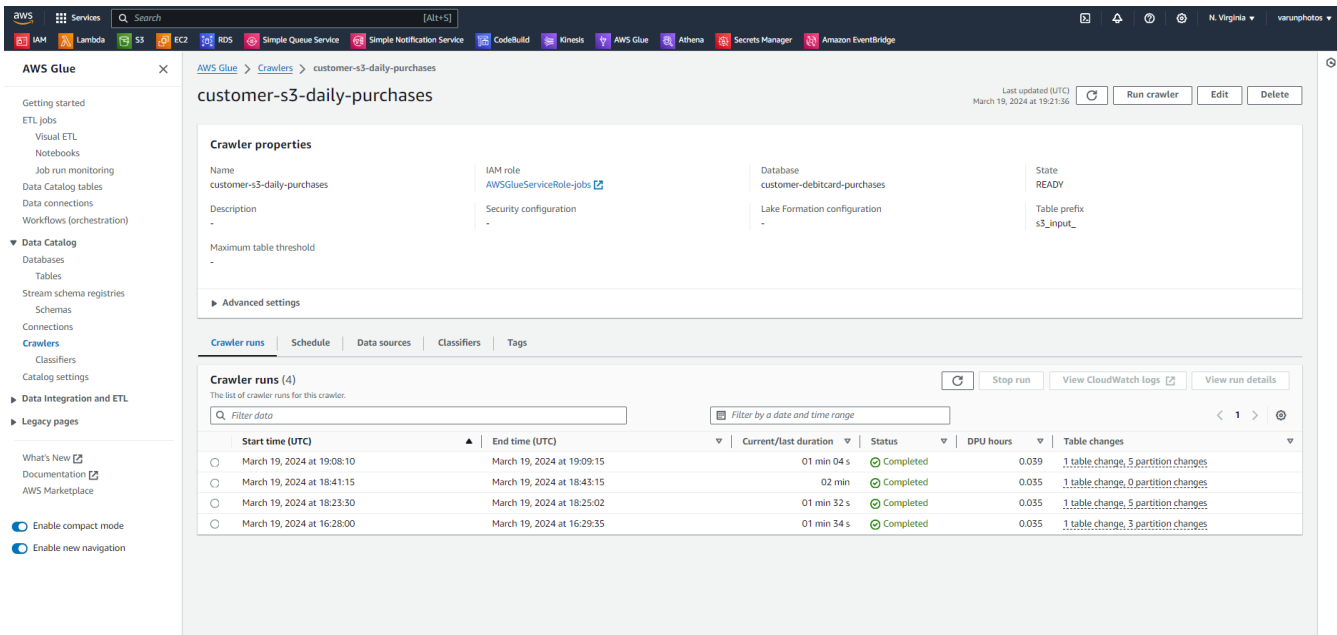
```
create table customer_debitcard_purchases(
customer_id Int primary key,
debit_card_number varchar(255),
bank_name varchar(255),
total_amount_spend float
);
```



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



Crawler for Postgres Database

postgresSQL-table-crawler
Last updated (UTC) March 19, 2024 at 19:22:09
Run crawler
Edit
Delete

Crawler properties

| | | | | | | | |
|-------------|---------------------------|------------------------|-------------------------|--------------|------------------------------|-------|-------|
| Name | postgresSQL-table-crawler | IAM role | AWSGlueServiceRole-jobs | Database | customer-debitcard-purchases | State | READY |
| Description | - | Security configuration | - | Table prefix | postgres_output | | |

Advanced settings

Crawler runs (1)
Stop run
View CloudWatch logs
View run details

Filter data
 Filter by a date and time range
 < 1 >

| Start time (UTC) | End time (UTC) | Current/last duration | Status | DPU hours | Table changes |
|----------------------------|----------------------------|-----------------------|-----------|-----------|-------------------------------------|
| March 19, 2024 at 17:05:44 | March 19, 2024 at 17:08:53 | 03 min 09 s | Completed | 0.232 | 1 table change, 0 partition changes |

Glue Database for S3 data source which contains metadata

s3_input_raw_data
Last updated (UTC) March 19, 2024 at 19:09:13
Version 4 (Current version)
Actions

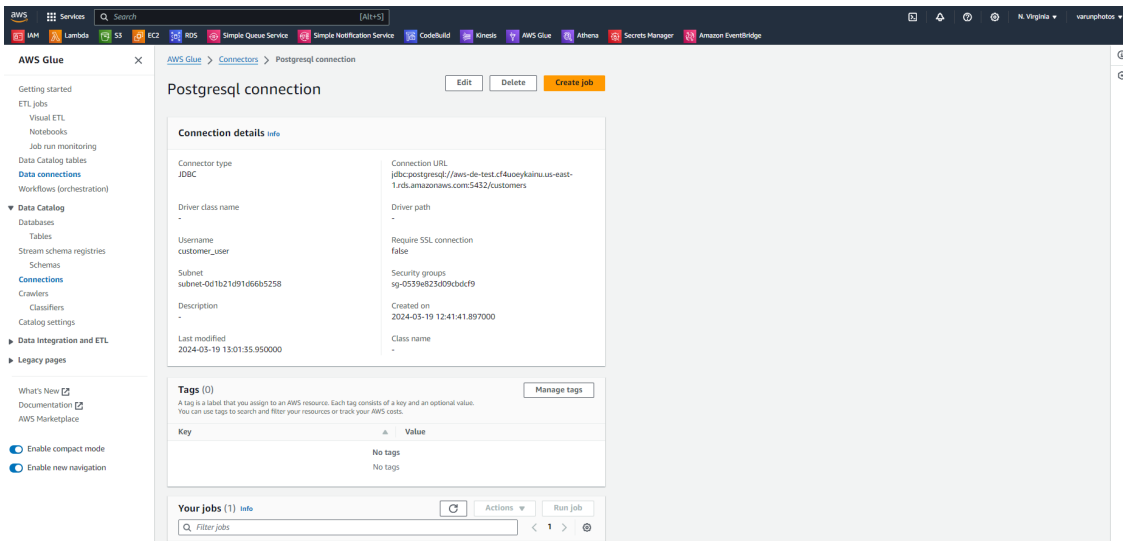
Table details
Advanced properties

| | | | | | | | |
|--------------|--|---------------|--|---------------|---|----------------|----------------------------|
| Name | s3_input_raw_data | Description | - | Database | customer-debitcard-purchases | Classification | CSV |
| Location | s3://debit-card-sampleraw-data/raw_data/ | Connection | - | Deprecated | - | Last updated | March 19, 2024 at 19:09:13 |
| Input format | org.apache.hadoop.mapred.TextInputFormat | Output format | org.apache.hadoop.hive.qjo.HiveIgnoreKeyTextOutputForm | Serialize lib | org.apache.hadoop.hive.serde2.LazySimpleSerDe | | |

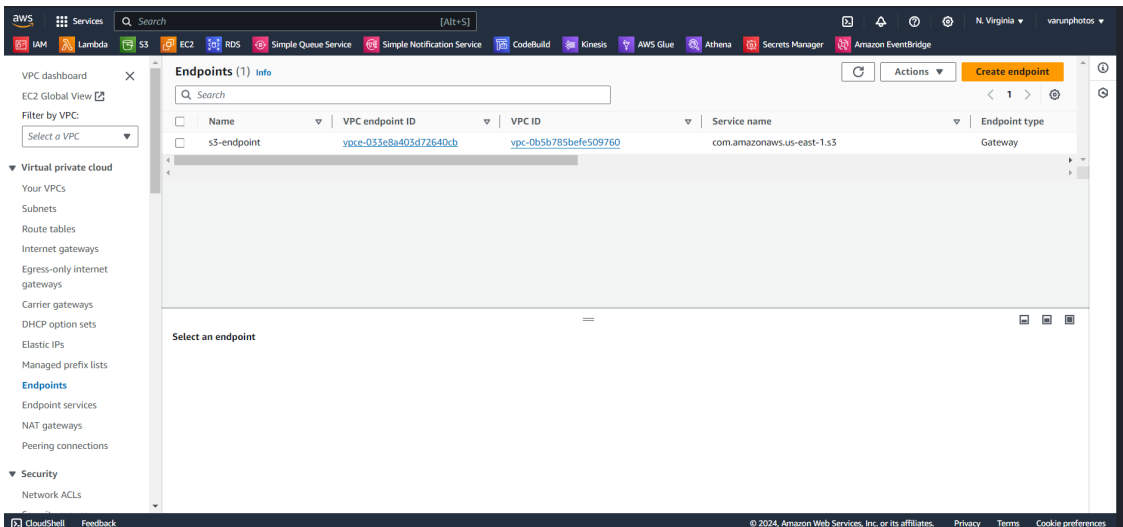
Schema (8)
Edit schema as JSON
Edit schema

| # | Column name | Data type | Partition key | Comment |
|---|-------------------|-----------|---------------|---------|
| 1 | customer_id | bigint | - | - |
| 2 | name | string | - | - |
| 3 | debit_card_number | bigint | - | - |
| 4 | debit_card_type | string | - | - |
| 5 | bank_name | string | - | - |
| 6 | transaction_date | string | - | - |
| 7 | amount_spent | double | - | - |

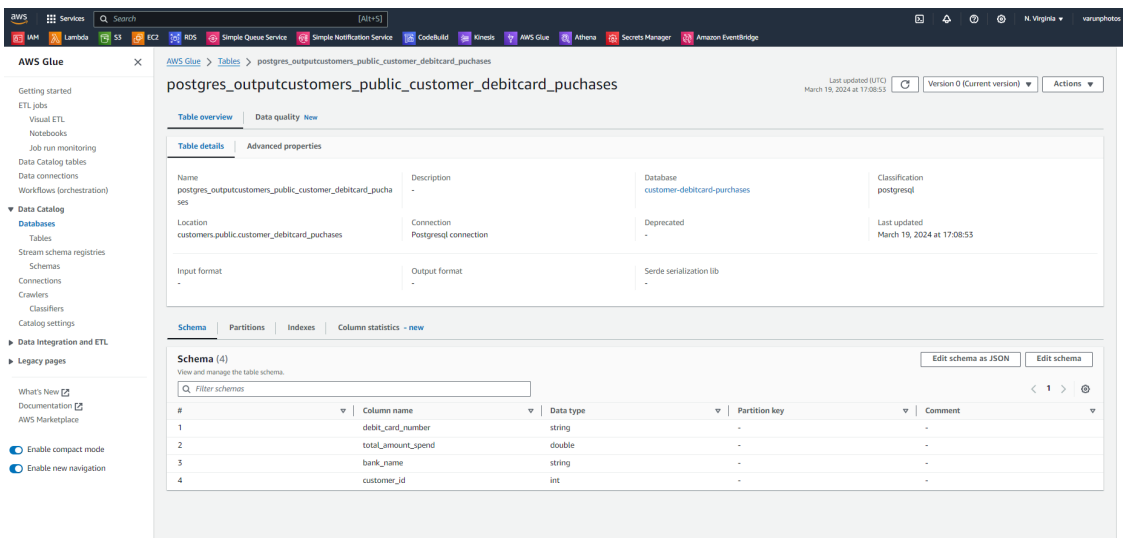
Glue Metastore for Postgres Database



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

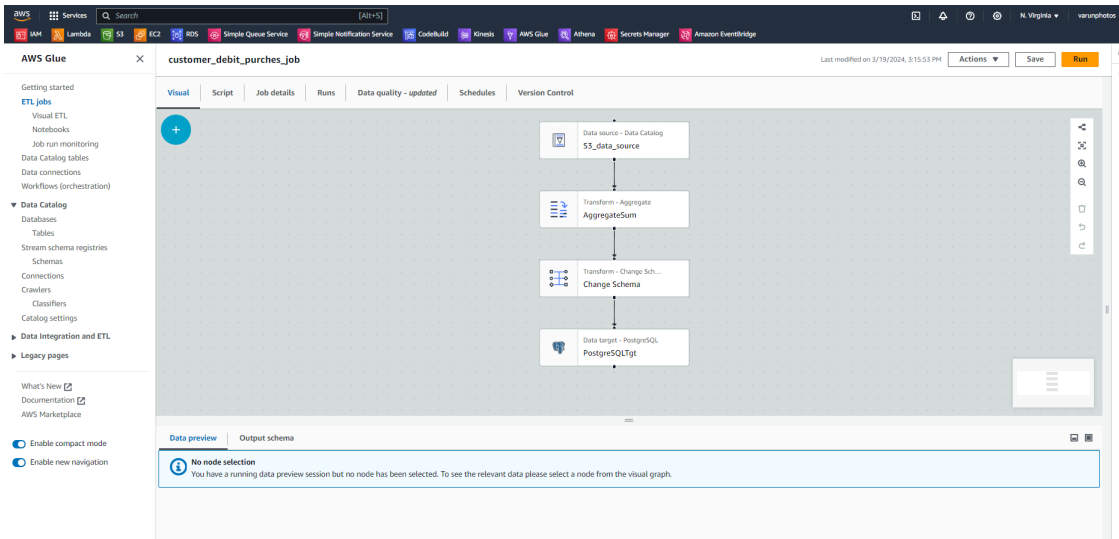


After the connection is successful, then you can run a Crawler for the PostgreSQL database



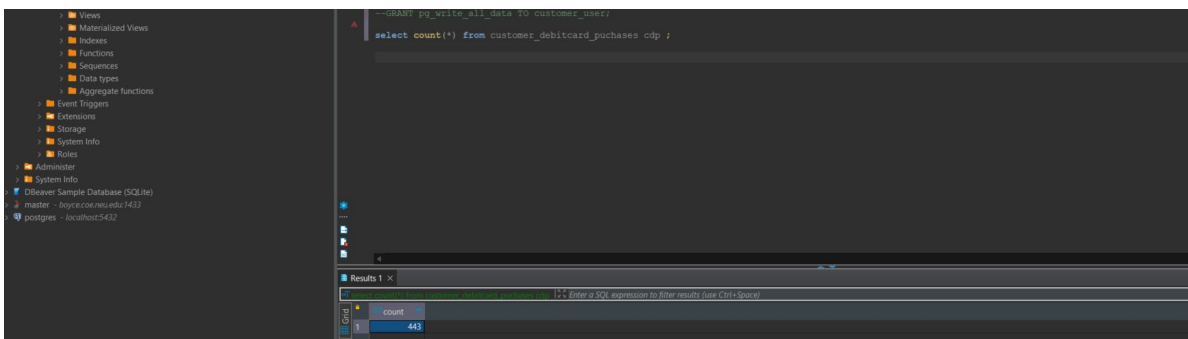
After everything was set up, now I created an ETL job using visual ETL from the dashboard

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

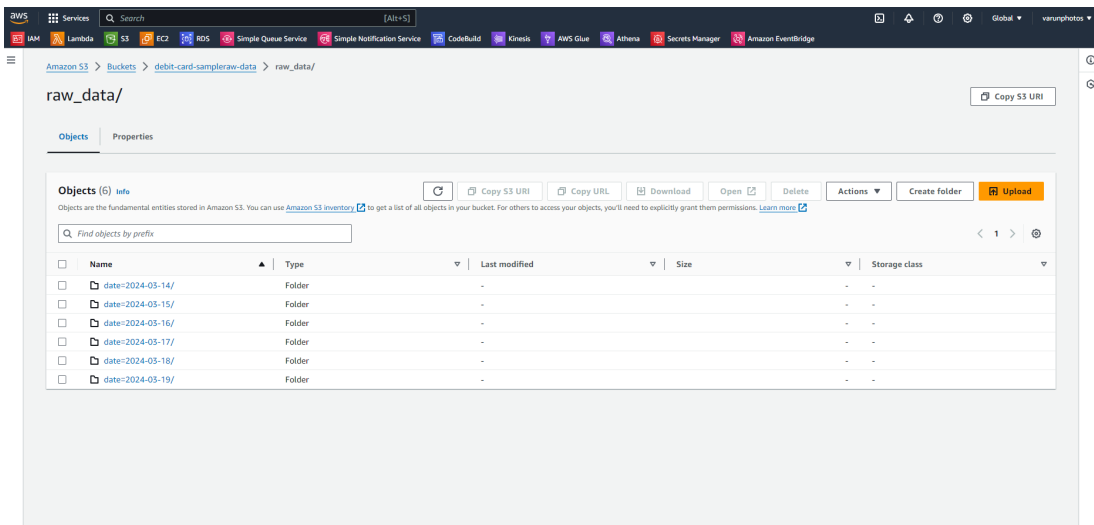


Go to jobdetails section in the GUI, then enable the Job bookmark part, which will help us for the incremental load of the daily data into postgres

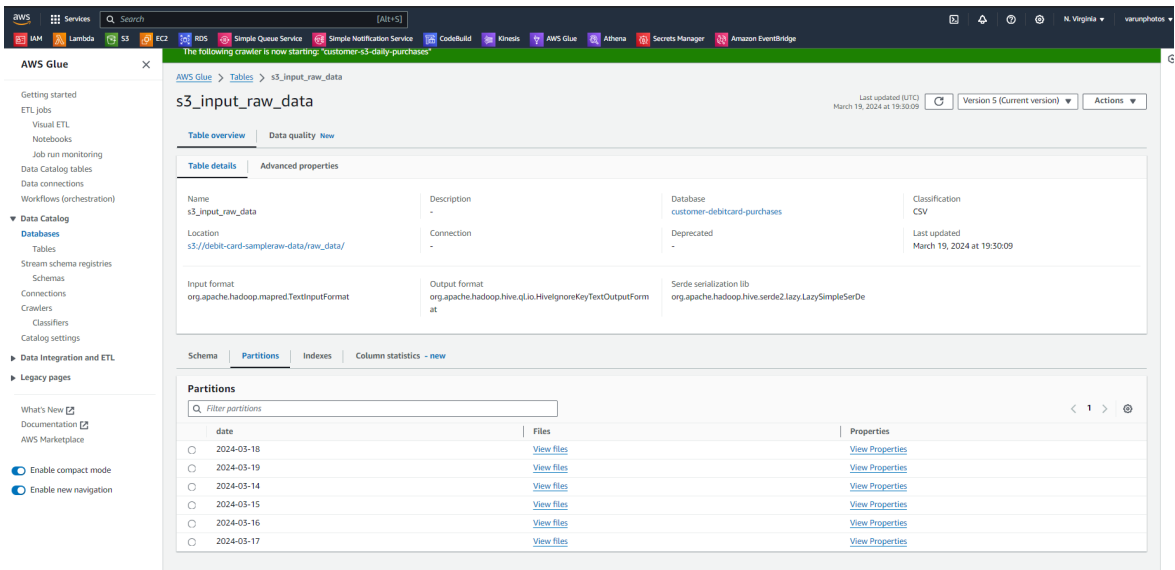
Before Incremental Load, it has only 443 records



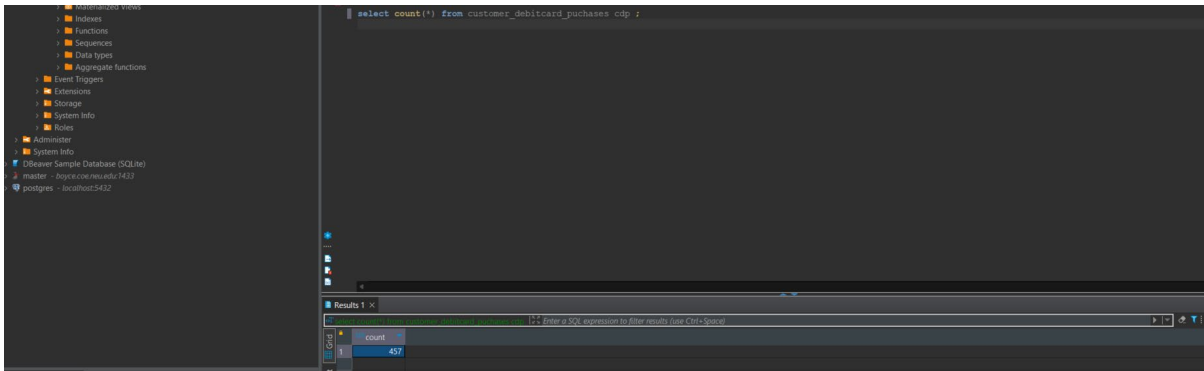
Below is the current Hive structure after adding today's data



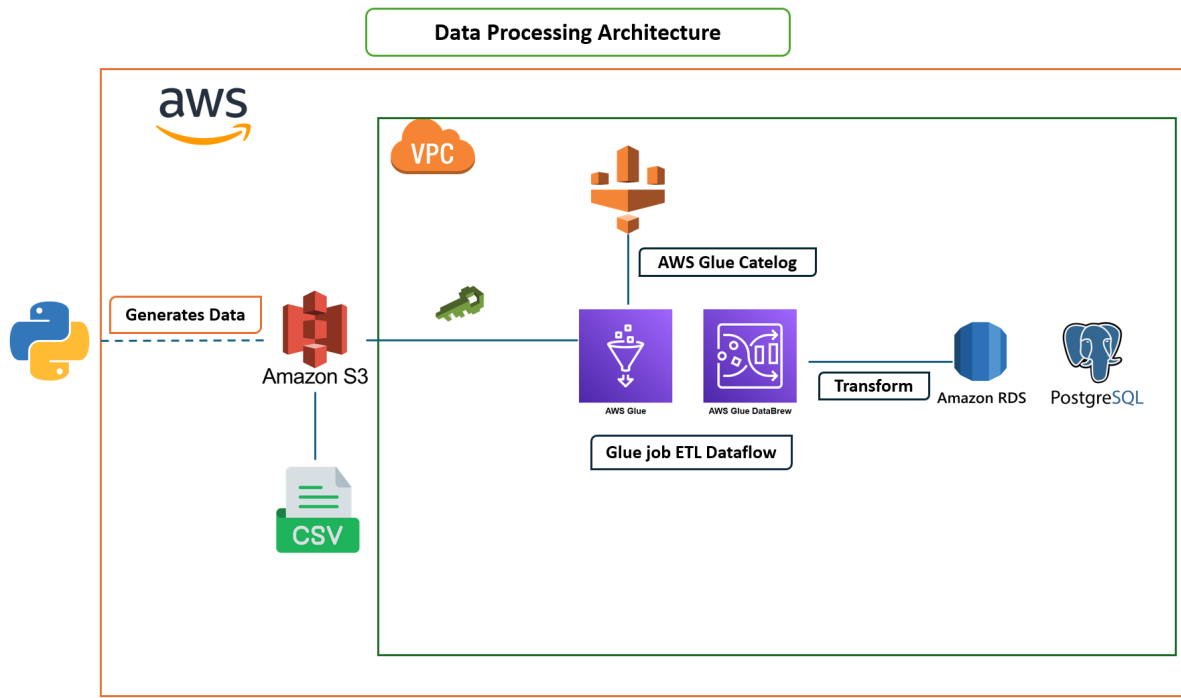
Again we need to run the crawler to identify if we have any changes



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



Data Processing Architecture Diagram:



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.