

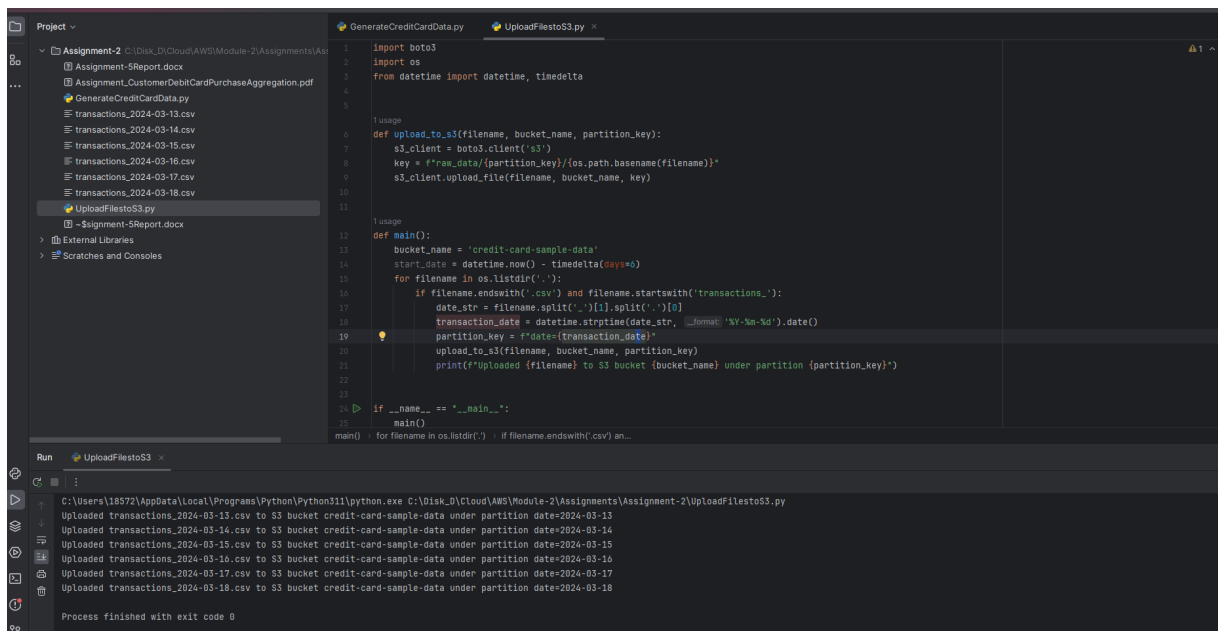
Sai Varun Kumar Namburi

Assignment – 5

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.



```
1 import boto3
2 import os
3 from datetime import datetime, timedelta
4
5 #usage
6 def upload_to_s3(filename, bucket_name, partition_key):
7     s3_client = boto3.client('s3')
8     key = f"raw_data/{partition_key}/{os.path.basename(filename)}"
9     s3_client.upload_file(filename, bucket_name, key)
10
11 #usage
12 def main():
13     bucket_name = 'credit-card-sample-data'
14     start_date = datetime.now() - timedelta(days=6)
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
26 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

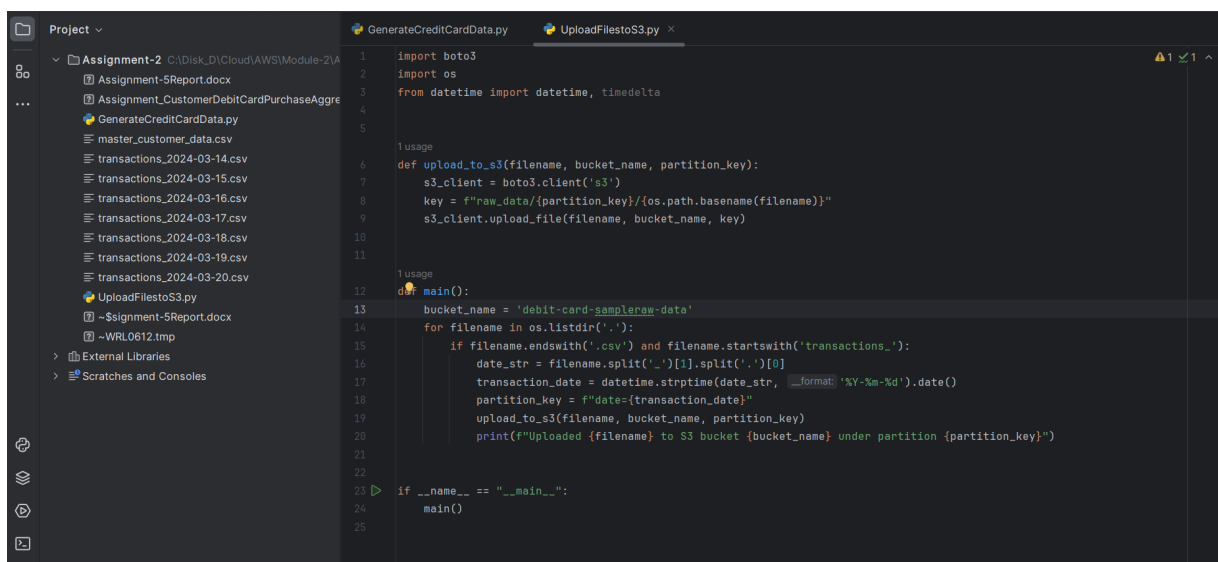
Uploaded transactions_2024-03-17.csv to S3 bucket credit-card-sample-data under partition date=2024-03-17

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

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_2024-03-14.csv to S3 bucket debit-card-sampleraw-data under partition date=2024-03-14

Uploaded transactions_2024-03-15.csv to S3 bucket debit-card-sampleraw-data under partition date=2024-03-15

Uploaded transactions_2024-03-16.csv to S3 bucket debit-card-sampleraw-data under partition date=2024-03-16

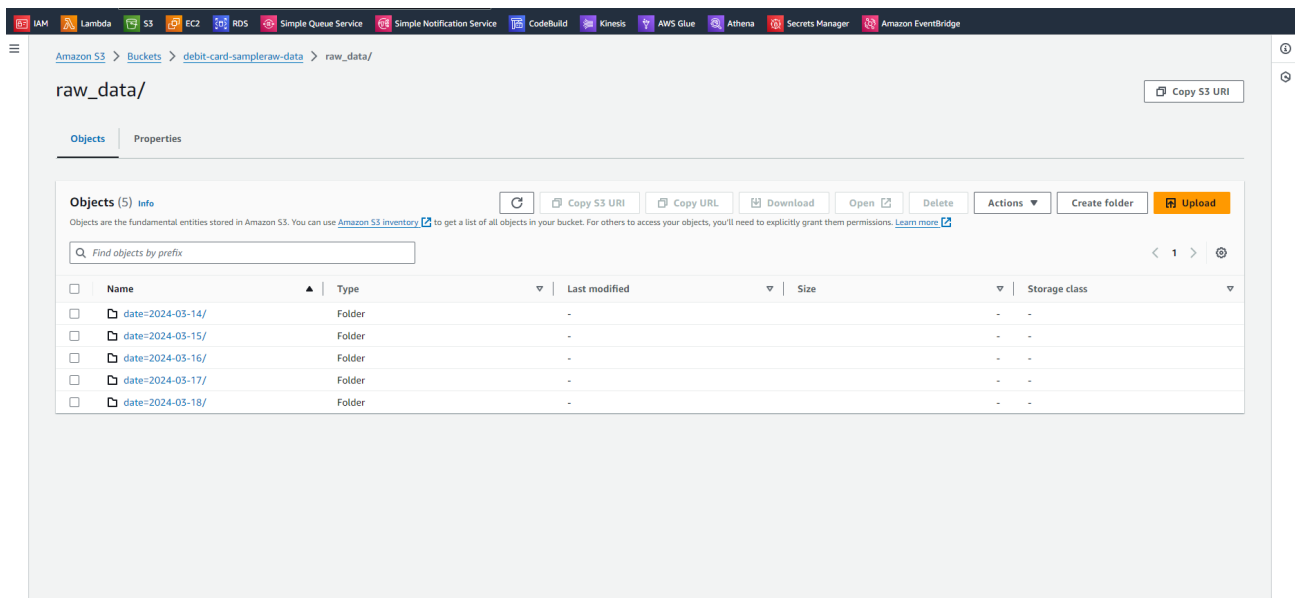
Uploaded transactions_2024-03-17.csv to S3 bucket debit-card-sampleraw-data under partition date=2024-03-17

Uploaded transactions_2024-03-18.csv to S3 bucket debit-card-sampleraw-data under partition date=2024-03-18

Uploaded transactions_2024-03-19.csv to S3 bucket debit-card-sampleraw-data under partition date=2024-03-19

Uploaded transactions_2024-03-20.csv to S3 bucket debit-card-sampleraw-data under partition date=2024-03-20

Process finished with exit code 0

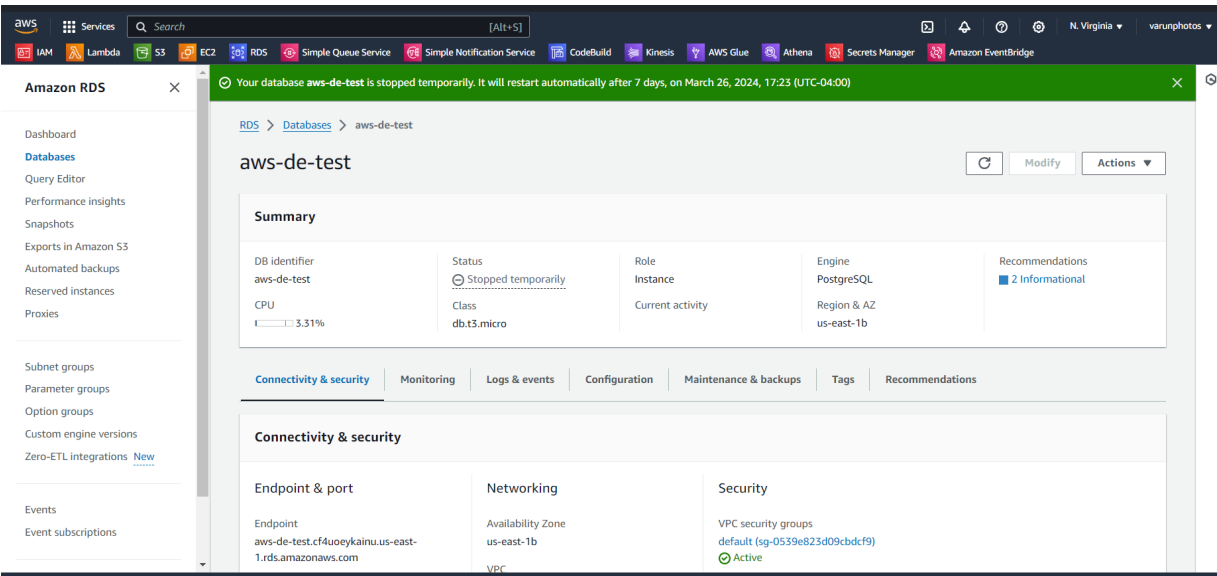


Step 3:

- 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

The screenshot shows the AWS Glue console for a crawler named 'customer-s3-daily-purchases'. The crawler properties are as follows:

Property	Value
Name	customer-s3-daily-purchases
Description	-
Maximum table threshold	-
IAM role	AWSGlueServiceRole-jobs
Security configuration	-
Database	customer-debitcard-purchases
Lake Formation configuration	-
State	READY
Table prefix	s3_input_

The crawler runs section shows 4 completed runs. The table below summarizes the data:

Start time (UTC)	End time (UTC)	Current/last duration	Status	DPU hours	Table changes
March 19, 2024 at 19:08:10	March 19, 2024 at 19:09:15	01 min 04 s	Completed	0.039	1 table change, 5 partition changes
March 19, 2024 at 18:41:15	March 19, 2024 at 18:43:15	02 min	Completed	0.035	1 table change, 0 partition changes
March 19, 2024 at 18:23:30	March 19, 2024 at 18:25:02	01 min 32 s	Completed	0.035	1 table change, 5 partition changes
March 19, 2024 at 16:28:00	March 19, 2024 at 16:29:35	01 min 34 s	Completed	0.035	1 table change, 3 partition changes

Crawler for Postgres Database

The screenshot shows the AWS Glue console for a crawler named 'postgresql-table-crawler'. The crawler properties are as follows:

Property	Value
Name	postgresql-table-crawler
Description	-
IAM role	AWSGlueServiceRole-jobs
Security configuration	-
Database	customer-debitcard-purchases
Lake Formation configuration	-
State	READY
Table prefix	postgres_output

The crawler runs section shows 1 completed run. The table below summarizes the data:

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

The screenshot shows the AWS Glue console interface. On the left, there's a navigation menu with options like 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Data Catalog', 'Databases', 'Tables', 'Stream schema registries', 'Schemas', 'Connections', 'Crawlers', 'Classifiers', 'Catalog settings', 'Data Integration and ETL', and 'Legacy pages'. The main content area displays the 's3_input_raw_data' table details. It includes a 'Table overview' tab and a 'Data quality' tab. The 'Table details' section shows the table's name, location, input/output formats, and database. The 'Schema' section shows a list of columns with their names, data types, and partition keys.

#	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

The screenshot shows the AWS Glue console interface. On the left, there's a navigation menu with options like 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Data Catalog', 'Databases', 'Tables', 'Stream schema registries', 'Schemas', 'Connections', 'Crawlers', 'Classifiers', 'Catalog settings', 'Data Integration and ETL', and 'Legacy pages'. The main content area displays the 'Postgresql connection' details. It includes a 'Connection details' section with fields for connector type, driver class name, username, subnet, description, last modified, connection URL, driver path, require SSL connection, security groups, created on, and class name. There's also a 'Tags' section and a 'Your jobs' section.

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

The screenshot shows the AWS VPC console interface. On the left, there's a navigation menu with options like 'VPC dashboard', 'EC2 Global View', 'Filter by VPC', 'Select a VPC', 'Virtual private cloud', 'Your VPCs', 'Subnets', 'Route tables', 'Internet gateways', 'Egress-only internet gateways', 'Carrier gateways', 'DHCP option sets', 'Elastic IPs', 'Managed prefix lists', 'Endpoints', 'Endpoint services', 'NAT gateways', 'Peering connections', 'Security', and 'Network ACLs'. The main content area displays the 'Endpoints' list. It includes a table with columns for Name, VPC endpoint ID, VPC ID, Service name, and Endpoint type. The table shows one endpoint named 's3-endpoint' with VPC endpoint ID 'vpce-033e8a403d72640cb' and VPC ID 'vpc-0b5b785befe509760'. Below the table, there's a 'Select an endpoint' section.

Name	VPC endpoint ID	VPC ID	Service name	Endpoint type
s3-endpoint	vpce-033e8a403d72640cb	vpc-0b5b785befe509760	com.amazonaws.us-east-1.s3	Gateway

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

The screenshot shows the AWS Glue console interface. The left sidebar contains navigation options like 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Data Catalog', 'Databases', 'Tables', 'Stream schema registries', 'Schemas', 'Connections', 'Crawlers', 'Classifiers', 'Catalog settings', 'Data Integration and ETL', and 'Legacy pages'. The main content area displays the 'Table details' for the table 'postgres_outputcustomers_public_customer_debitcard_purchases'. The table details include: Name (postgres_outputcustomers_public_customer_debitcard_purchases), Description (.), Database (customer-debitcard-purchases), Classification (postgres), Location (customers.public.customer_debitcard_purchases), Connection (Postgresql connection), Deprecated (.), Last updated (March 19, 2024 at 17:08:53), Input format (.), Output format (.), and Serde serialization lib (.). Below the table details, there is a 'Schema' section showing the table schema with 4 columns: debt_card_number (string), total_amount_spend (double), bank_name (string), and customer_id (int).

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

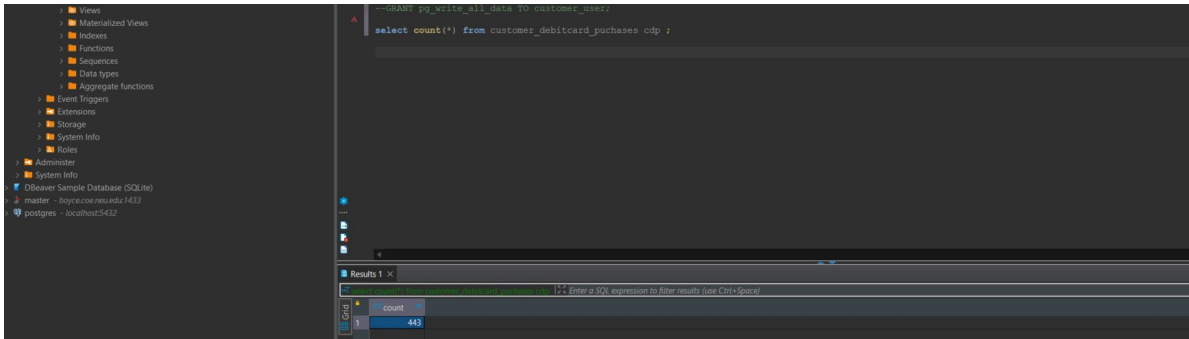
1. Add the Data source
2. Add aggregation transform
3. Change Schema based on your destination
4. Connect with the target database

The screenshot shows the AWS Glue console interface for the ETL job 'customer_debit_purces_job'. The left sidebar is the same as the previous screenshot. The main content area displays the 'Visual' tab of the job. The visual graph shows the following workflow: 'Data source - Data Catalog S3_data_source' -> 'Transform - Aggregate AggregateSum' -> 'Transform - Change Sch... Change Schema' -> 'Data target - PostgreSQL PostgresQLTgt'. The 'Data preview' section at the bottom shows a message: 'No node selection. You have a running data preview session but no node has been selected. To see the relevant data please select a node from the visual graph.'

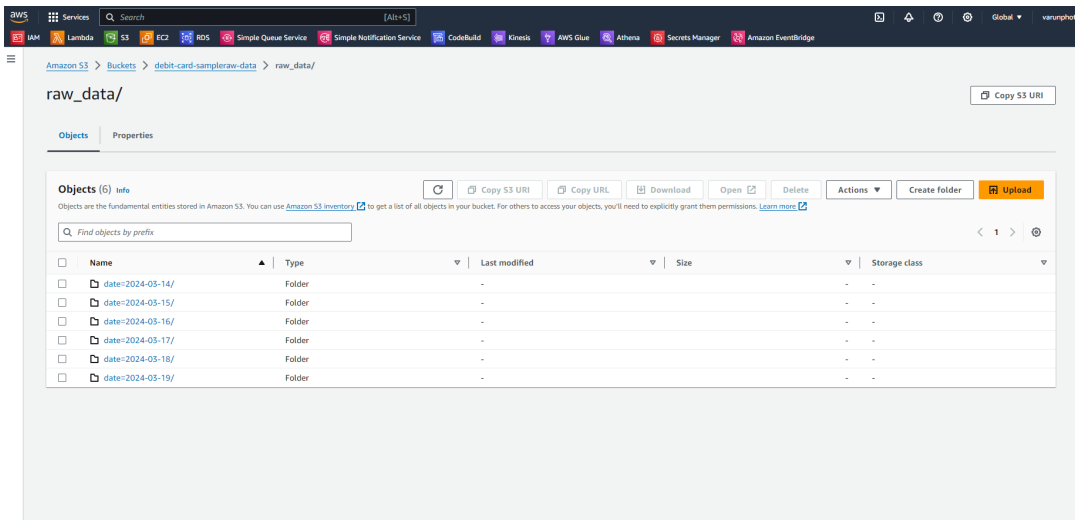
Go to jobdetails section in the GUI, then enable the Job bookmark part, which will help us for the incremental load of the daily data into postgres

The screenshot shows the 'Job details' section of the AWS Glue console for the job 'customer_debit_purces_job'. The 'Worker type' is set to 'G 1X (4vCPU and 16GB RAM)'. The 'Automatically scale the number of workers' checkbox is checked. The 'Requested number of workers' is set to 2. The 'Generate job insights' checkbox is checked. The 'Job bookmark' is set to 'Enable'. The 'Flex execution' checkbox is checked.

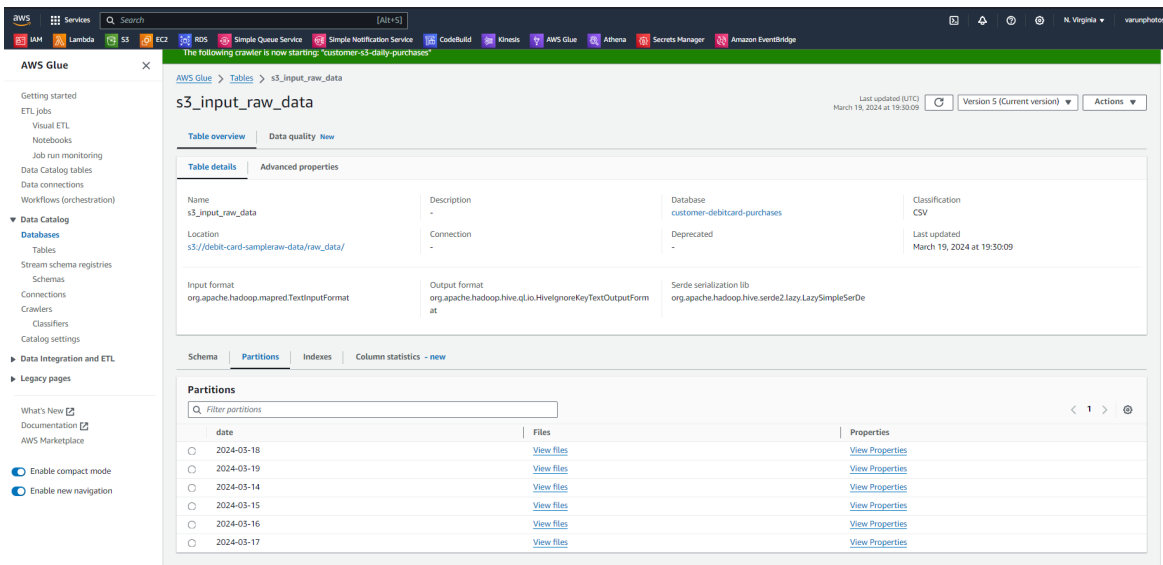
Before Incremental Load, it has only 443 records



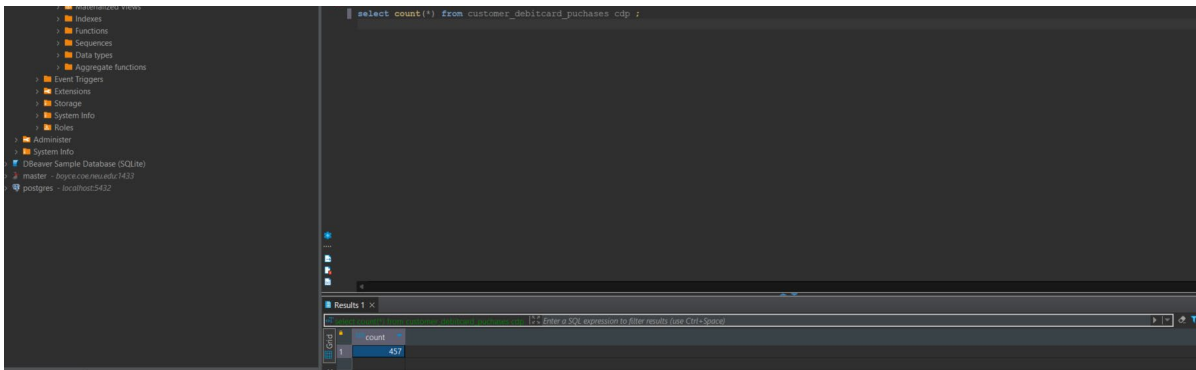
Below is the current Hive structure after adding today's data



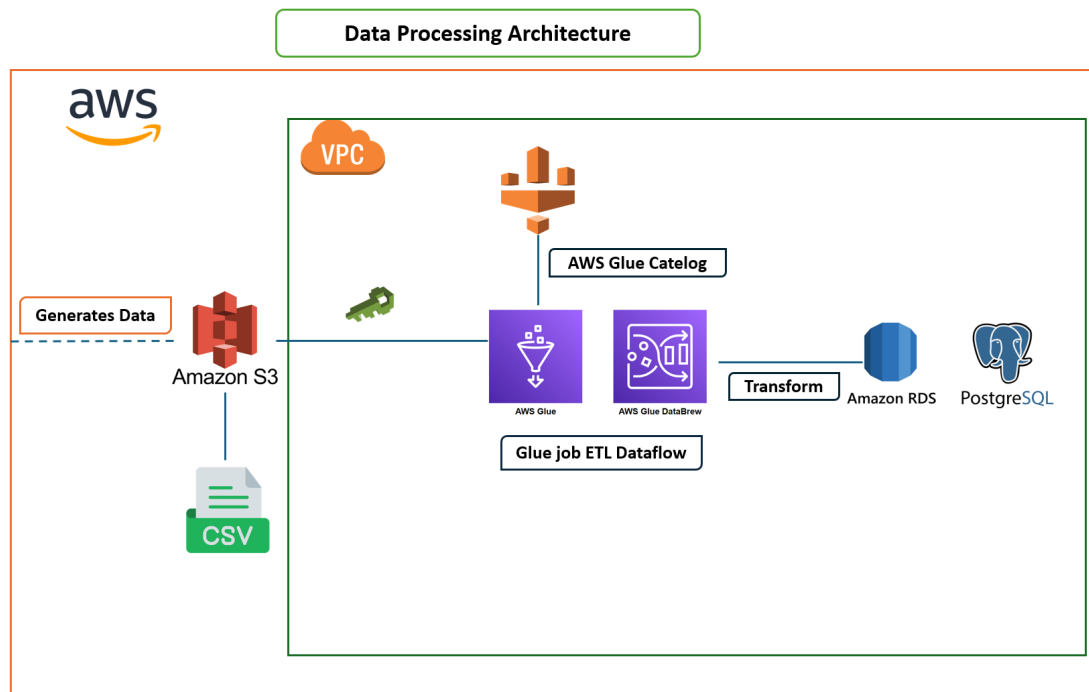
Again we need to run the crawler to identify if we have any changes



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.