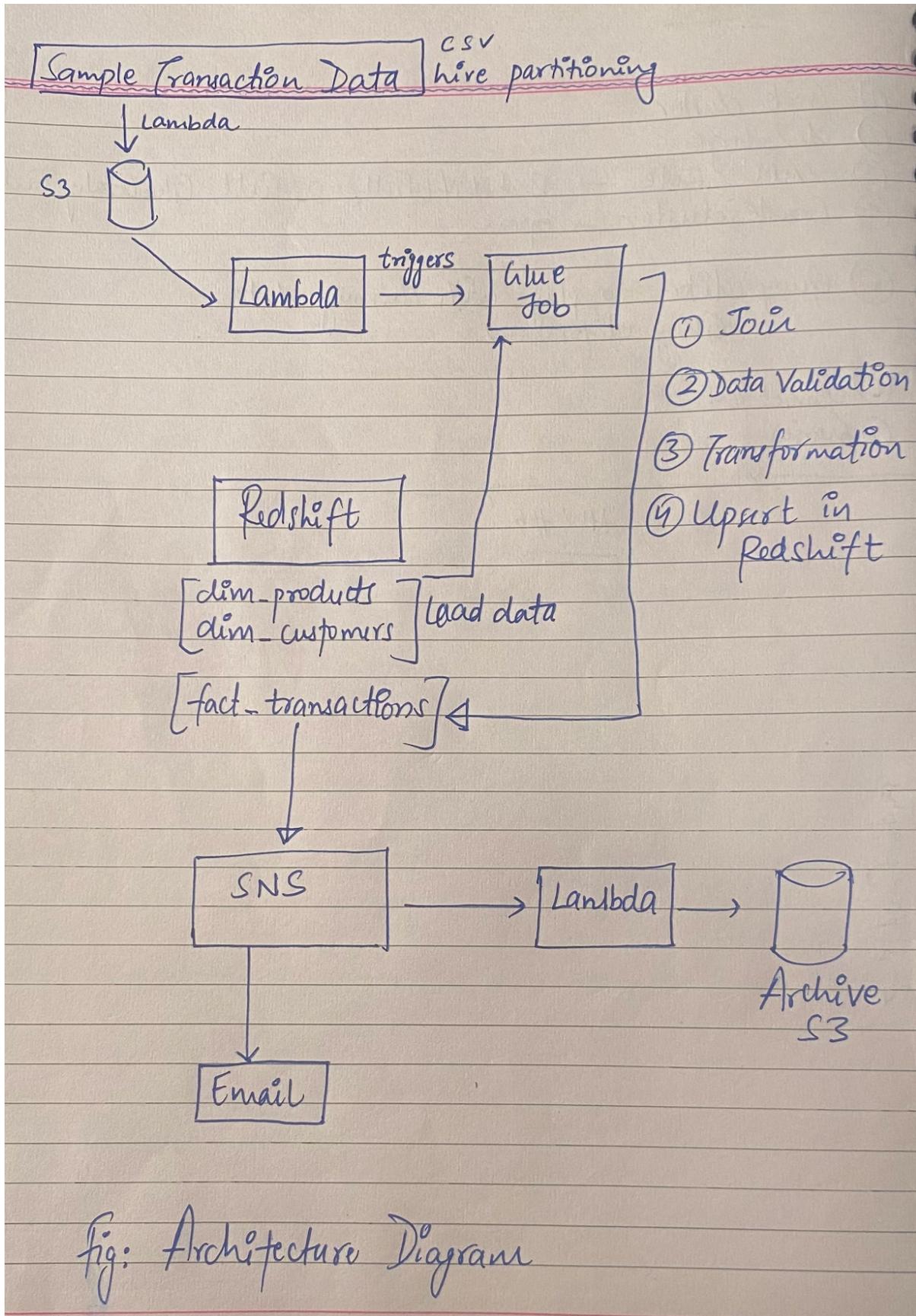


Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline



Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

Create Tables in AWS Redshift and Load Data

The screenshot shows two terminal sessions in a dark-themed PostgreSQL client.

Session #1: A single command to create a schema:CREATE SCHEMA ecommerce_data;

Session #2: A more complex command to create a dimension table named `dim_customers`. It includes columns for `customer_id`, `first_name`, `last_name`, `email`, and `membership_level`. The table uses `DISTSTYLE KEY` and `DISTKEY (membership_level)`.CREATE TABLE ecommerce_data.dim_customers (
 customer_id VARCHAR(20) ENCODE lzo,
 first_name VARCHAR(255) ENCODE lzo,
 last_name VARCHAR(255) ENCODE lzo,
 email VARCHAR(100) ENCODE lzo,
 membership_level VARCHAR(2) ENCODE lzo
)
DISTSTYLE KEY
DISTKEY (membership_level)
SORTKEY (customer_id, first_name, last_name);

The screenshot shows a third terminal session continuing from the previous ones.

Session #3: An `INSERT INTO` statement with multiple rows of data for the `dim_customers` table. The data includes five customers with their names, emails, and membership levels.INSERT INTO ecommerce_data.dim_customers (customer_id, first_name, last_name, email, membership_level)
VALUES ('CUST00001', 'John', 'Doe', 'john.doe@example.com', 'BR'),
 ('CUST00002', 'Jane', 'Smith', 'jane.smith@example.com', 'SL'),
 ('CUST00003', 'Michael', 'Lee', 'michael.lee@example.com', 'GL'),
 ('CUST00004', 'Alice', 'Brown', 'alice.brown@example.com', 'BR'),
 ('CUST00005', 'David', 'Miller', 'david.miller@example.com', 'SL');

Session #4: A simple query to select all columns from the `dim_customers` table.select * from ecommerce_data.dim_customers;

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

The screenshot shows a database query editor interface with the following details:

- Query:** A CREATE TABLE statement for "ecommerce_data.dim_products". The table has columns: product_id (VARCHAR(20)), product_name (VARCHAR(255)), category (VARCHAR(100)), price (DECIMAL(10,2)), and supplier_id (VARCHAR(20)). It uses DISTSTYLE KEY, DISTKEY for category, and SORTKEY for (product_id, product_name). The code is color-coded with syntax highlighting.
- Result 1:** Summary section showing 0 rows returned, 105ms elapsed time, and the result set query.
- Output:** The generated SQL code for creating the table.

The screenshot shows a database query editor interface with the following details:

- Query:** An INSERT INTO statement for "ecommerce_data.dim_products". It inserts 10 rows of data with columns: product_id, product_name, category, price, and supplier_id. The data includes various products like Laptop, Smartphone, Headphones, T-Shirt, Jeans, Coffee Maker, Microwave, Vacuum Cleaner, Book, and Movie.
- Result 1:** Summary section showing 10 affected rows, 23.6s elapsed time, and the result set query.
- Output:** The generated SQL code for inserting data into the table.

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

The screenshot shows a database query editor interface. At the top, there are buttons for 'Run' and 'Limit 100'. The main area contains the following SQL code:

```
1 CREATE TABLE ecommerce_data.fact_transactions (
2     transaction_id VARCHAR(20) ENCODE lzo,
3     customer_id VARCHAR(20) ENCODE lzo,
4     customer_email VARCHAR(100) ENCODE lzo,
5     product_id VARCHAR(20) ENCODE lzo,
6     product_name VARCHAR(255) ENCODE lzo,
7     quantity INT ENCODE lzo,
8     price DECIMAL(10,2) ENCODE delta,
9     supplier_id VARCHAR(20) ENCODE lzo,
10    transaction_date DATE ENCODE bytedict,
11    transaction_type VARCHAR(20) ENCODE lzo,
12    payment_type VARCHAR(20) ENCODE lzo,
13    status VARCHAR(20) ENCODE lzo
14 )
15 DISTSTYLE KEY
16 DISTKEY (transaction_type)
17 SORTKEY (transaction_id);
```

Below the code, there is a 'Result 1' section with a 'Summary' table:

Summary
Returned rows: 0 Elapsed time: 120ms Result set query: <pre>/* RQEY2-hMC0RHKRJX */ CREATE TABLE ecommerce_data.fact_transactions (transaction_id VARCHAR(20) ENCODE lzo, customer_id VARCHAR(20) ENCODE lzo, customer_email VARCHAR(100) ENCODE lzo, product_id VARCHAR(20) ENCODE lzo, product_name VARCHAR(255) ENCODE lzo,</pre>

On the right side of the interface, there are 'Export' and 'Chart' buttons.

Generate Mock Data for the customers based on products. Use Code build for lambda. This lambda function will generate data and upload csv files into S3 in Hive Partitioning manner.

Python Code Link: https://github.com/panchalashish4/ecommerce-data-pipeline/tree/main/generate_data

Ashish Panchal

Assignment – 6

E-Commerce Data Pipeline

CodeBuild:

The screenshot shows the AWS CodeBuild console. On the left, there's a navigation sidebar with sections like Source, Artifacts, Build, Deploy, Pipeline, Settings, Go to resource, and Feedback. The main area is titled "ecommerce-transactions-data". It shows the configuration for the project, including the source provider (GitHub) and primary repository (panchalashish4/ecommerce-data-pipeline). The artifacts upload location and service role are also specified. Below this, the "Build history" tab is selected, showing two completed build runs. The first run, "ecommerce-transactions-data:9f2047d7-c38c-48a7-8f97-c024129eb532", was successful (Succeeded) with build number 2, submitted by GitHub-Hookshot/c74581f, and took 21 seconds, completed 2 minutes ago. The second run, "ecommerce-transactions-data:a7ae256c-1d62-4738-98d2-", was also successful (Succeeded) with build number 1, submitted by GitHub-Hookshot/c74581f, and took 22 seconds, completed 5 minutes ago.

Test Lambda to Generate Data and Upload into S3

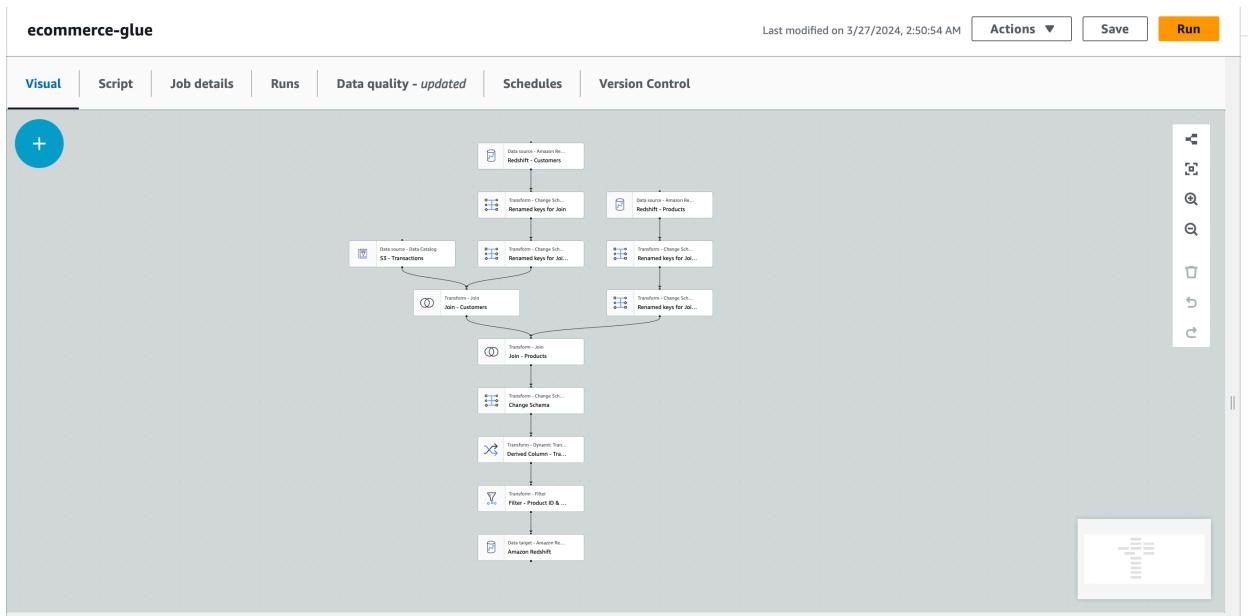
The screenshot shows the AWS S3 console. The path is "Amazon S3 > Buckets > ecommerce-transaction-data > transactions > year=2024/ > month=3/". The "Objects" tab is selected, showing five objects named "day=23/", "day=24/", "day=25/", "day=26/", and "day=27/". Each object is a folder. There are buttons for Actions, Create folder, and Upload at the top of the list. A search bar at the bottom allows finding objects by prefix.

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

Create crawlers in AWS Glue for new S3 bucket and AWS Redshift Tables

The screenshot shows the AWS Glue Crawler list page. It displays four crawlers: `customer-debit-card-purchase-rds-crawler`, `customer-debit-card-purchase-s3-crawler`, `ecommerce-s3-input-crawler`, and `redshift-fact-transaction`. All crawlers are in a `Ready` state with a green checkmark. Their last runs were successful, with timestamps ranging from March 25, 2024 at 05:00 to March 27, 2024 at 08:00. There are buttons for `Action`, `Run`, and `Create crawler`.

Create Glue ETL Flow as per our usecase



AWS Glue Script

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue import DynamicFrame
import gs_derived
```

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

```
import re

args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Script generated for node S3 - Transactions
S3Transactions_node1711523386952 =
glueContext.create_dynamic_frame.from_catalog(database="ecommerce-db",
table_name="s3_inputtransactions",
transformation_ctx="S3Transactions_node1711523386952")

# Script generated for node Redshift - Products
RedshiftProducts_node1711523340997 =
glueContext.create_dynamic_frame.from_catalog(database="ecommerce-db",
table_name="redshift_dev_ecommerce_data_dim_products",
redshift_tmp_dir="s3://aws-glue-assets-590184043718-us-east-1/temporary/",
transformation_ctx="RedshiftProducts_node1711523340997")

# Script generated for node Redshift - Customers
RedshiftCustomers_node1711523211825 =
glueContext.create_dynamic_frame.from_catalog(database="ecommerce-db",
table_name="redshift_dev_ecommerce_data_dim_customers",
redshift_tmp_dir="s3://aws-glue-assets-590184043718-us-east-1/temporary/",
transformation_ctx="RedshiftCustomers_node1711523211825")

# Script generated for node Renamed keys for Join - Products
RenamedkeysforJoinProducts_node1711523548475 =
ApplyMapping.apply(frame=RedshiftProducts_node1711523340997,
mappings=[("price", "decimal(10,2)", "price", "decimal"), ("product_id",
"string", "product_id", "string"), ("category", "string", "category",
"string"), ("product_name", "string", "product_name", "string"),
("supplier_id", "string", "supplier_id", "string")],
transformation_ctx="RenamedkeysforJoinProducts_node1711523548475")

# Script generated for node Renamed keys for Join
RenamedkeysforJoin_node1711523465265 =
ApplyMapping.apply(frame=RedshiftCustomers_node1711523211825,
mappings=[("customer_id", "string", "customer_id", "string"), ("first_name",
"string", "first_name", "string"), ("last_name", "string", "last_name",
"string"), ("email", "string", "email", "string"), ("membership_level",
"string", "membership_level", "string")],
transformation_ctx="RenamedkeysforJoin_node1711523465265")

# Script generated for node Renamed keys for Join - Products
RenamedkeysforJoinProducts_node1711531733910 =
ApplyMapping.apply(frame=RenamedkeysforJoinProducts_node1711523548475,
mappings=[("product_id", "string", "rs_product_id", "string"),
("product_name", "string", "rs_product_name", "string"), ("category",
"string", "rs_category", "string"), ("price", "decimal", "rs_price",
"decimal"), ("supplier_id", "string", "rs_supplier_id", "string")],
transformation_ctx="RenamedkeysforJoinProducts_node1711531733910")
```

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

```
# Script generated for node Renamed keys for Join - Customers
RenamedkeysforJoinCustomers_node1711531701689 =
ApplyMapping.apply(frame=RenamedkeysforJoin_node1711523465265,
mappings=[("customer_id", "string", "rs_customer_id", "string"),
("first_name", "string", "rs_first_name", "string"), ("last_name", "string",
"rs_last_name", "string"), ("email", "string", "rs_email", "string"),
("membership_level", "string", "rs_membership_level", "string")],
transformation_ctx="RenamedkeysforJoinCustomers_node1711531701689")

# Script generated for node Join - Customers
JoinCustomers_node1711523428722 =
Join.apply(frame1=S3Transactions_node1711523386952,
frame2=RenamedkeysforJoinCustomers_node1711531701689, keys1=["customer_id"],
keys2=["rs_customer_id"],
transformation_ctx="JoinCustomers_node1711523428722")

# Script generated for node Join - Products
JoinProducts_node1711523515185 =
Join.apply(frame1=JoinCustomers_node1711523428722,
frame2=RenamedkeysforJoinProducts_node1711531733910, keys1=["product_id"],
keys2=["rs_product_id"], transformation_ctx="JoinProducts_node1711523515185")

# Script generated for node Change Schema
ChangeSchema_node1711523572022 =
ApplyMapping.apply(frame=JoinProducts_node1711523515185, mappings=[("price",
"double", "price", "decimal"), ("product_id", "string", "product_id",
"varchar"), ("customer_id", "string", "customer_id", "varchar"),
("transaction_date", "string", "transaction_date", "date"), ("payment_type",
"string", "payment_type", "varchar"), ("transaction_id", "string",
"transaction_id", "varchar"), ("status", "string", "status", "varchar"),
("rs_email", "string", "customer_email", "varchar"), ("rs_product_name",
"string", "product_name", "varchar"), ("rs_supplier_id", "string",
"supplier_id", "varchar")],
transformation_ctx="ChangeSchema_node1711523572022")

# Script generated for node Derived Column - Transaction_Type
DerivedColumnTransaction_Type_node1711523858206 =
ChangeSchema_node1711523572022.gs_derived(colName="transaction_type",
expr="case when price < 100 then 'small' when price > 100 and price < 500
then 'medium' when price > 500 then 'large' else 'not defined' end")

# Script generated for node Filter - Product ID & Customer ID
FilterProductIDCustomerID_node1711525042051 =
Filter.apply(frame=DerivedColumnTransaction_Type_node1711523858206, f=lambda
row: (bool(re.match("^PROD[0-9]{5}$", row["product_id"])) and
bool(re.match("^CUST[0-9]{5}$", row["customer_id"]))),
transformation_ctx="FilterProductIDCustomerID_node1711525042051")

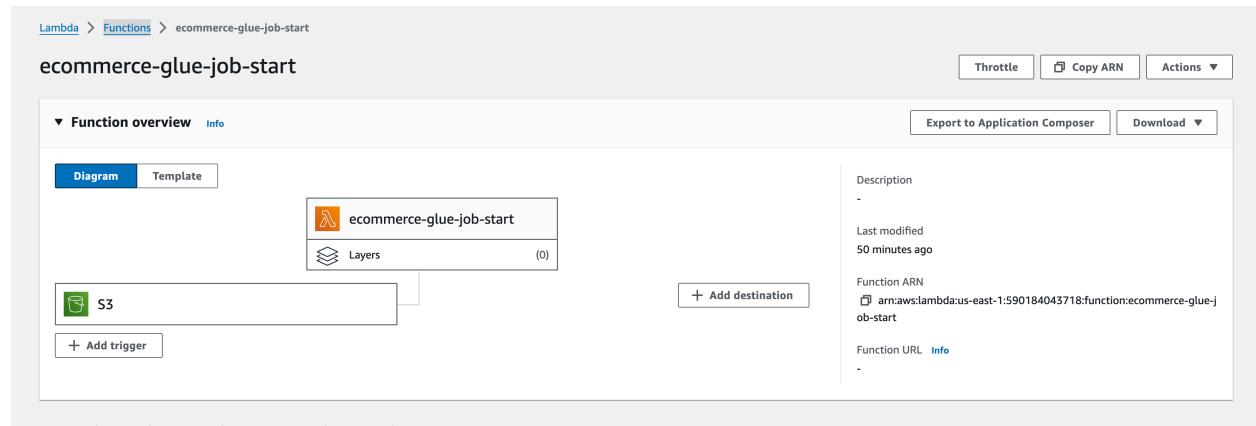
# Script generated for node Amazon Redshift
AmazonRedshift_node1711532392573 =
glueContext.write_dynamic_frame.from_options(frame=FilterProductIDCustomerID_
node1711525042051, connection_type="redshift",
connection_options={"postactions": "BEGIN; DELETE FROM
ecommerce_data.fact_transactions USING
```

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

```
ecommerce_data.fact_transactions_temp_09nd7y WHERE
ecommerce_data.fact_transactions_temp_09nd7y.transaction_id =
ecommerce_data.fact_transactions.transaction_id; INSERT INTO
ecommerce_data.fact_transactions SELECT * FROM
ecommerce_data.fact_transactions_temp_09nd7y; DROP TABLE
ecommerce_data.fact_transactions_temp_09nd7y; END;", "redshiftTmpDir":
"s3://aws-glue-assets-590184043718-us-east-1/temporary/",
"useConnectionProperties": "true", "dbtable":
"ecommerce_data.fact_transactions_temp_09nd7y", "connectionName": "Redshift
connection", "preactions": "CREATE TABLE IF NOT EXISTS
ecommerce_data.fact_transactions (price DECIMAL, product_id VARCHAR,
customer_id VARCHAR, transaction_date DATE, payment_type VARCHAR,
transaction_id VARCHAR, status VARCHAR, customer_email VARCHAR, product_name
VARCHAR, supplier_id VARCHAR, transaction_type VARCHAR); DROP TABLE IF EXISTS
ecommerce_data.fact_transactions_temp_09nd7y; CREATE TABLE
ecommerce_data.fact_transactions_temp_09nd7y AS SELECT * FROM
ecommerce_data.fact_transactions WHERE 1=2;"},
transformation_ctx="AmazonRedshift_node1711532392573")
```

job.commit()

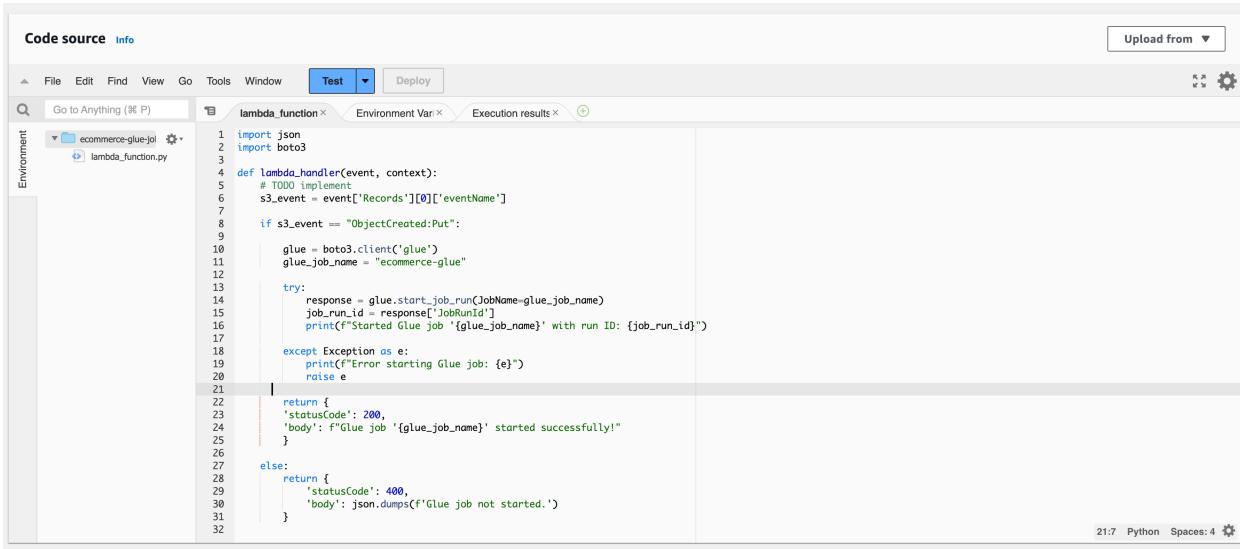
Create lambda to start glue job when data is generated in S3:



Ashish Panchal

Assignment – 6

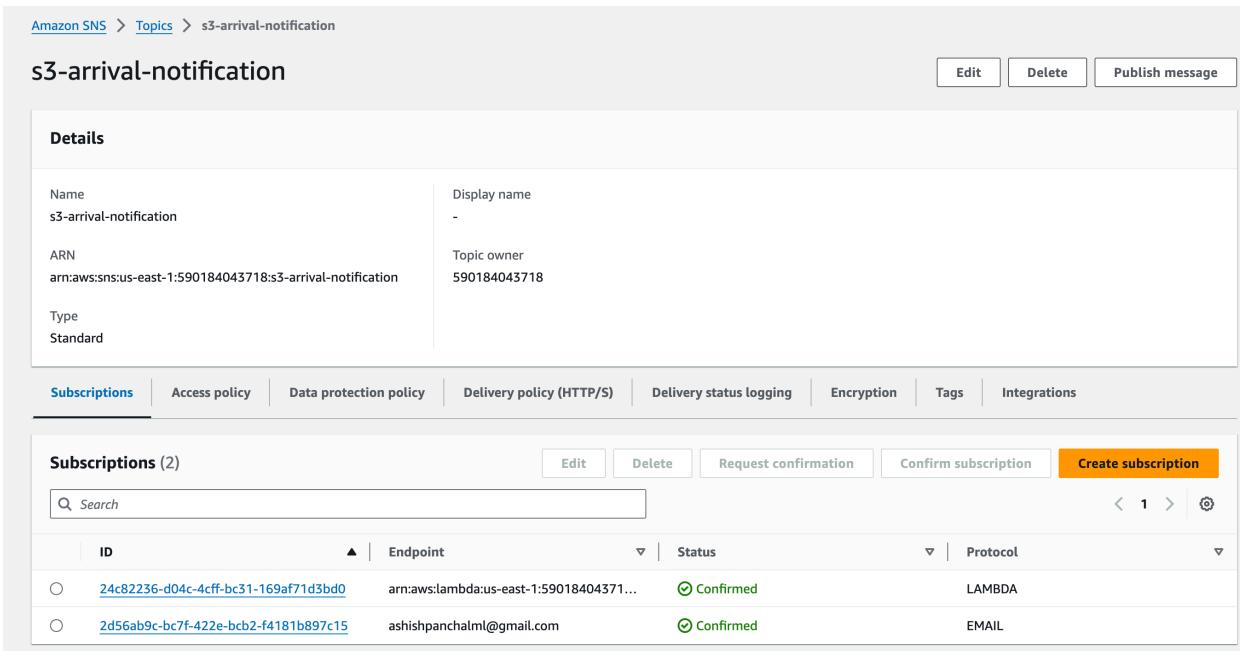
E-Commerce Data Pipeline



The screenshot shows the AWS Lambda function editor. The function name is 'lambda_function' and it is part of the 'ecommerce-glue-job' project. The code is written in Python and defines a lambda_handler function that starts a Glue job when an S3 object is created. The code uses the boto3 library to interact with Glue.

```
1 import json
2 import boto3
3
4 def lambda_handler(event, context):
5     # TODO implement
6     s3_event = event['Records'][0]['eventName']
7
8     if s3_event == "ObjectCreated:Put":
9         glue = boto3.client('glue')
10        glue_job_name = "ecommerce-glue"
11
12        try:
13            response = glue.start_job_run(JobName=glue_job_name)
14            job_run_id = response['JobRunId']
15            print(f"Started Glue job '{glue_job_name}' with run ID: {job_run_id}")
16
17        except Exception as e:
18            print(f"Error starting Glue job: {e}")
19            raise e
20
21
22    return {
23        'statusCode': 200,
24        'body': f"Glue job '{glue_job_name}' started successfully!"
25    }
26
27 else:
28    return {
29        'statusCode': 400,
30        'body': json.dumps(f"Glue job not started.")
31    }
32
```

Create SNS Topic to send email when Glue ETL job is done, and it can be used in final lambda to archive the data



The screenshot shows the AWS SNS Topics page. A new topic named 's3-arrival-notification' has been created. It has two subscriptions: one to a Lambda function and one to an email address (ashishpanchalml@gmail.com). The topic owner is listed as 590184043718.

Details

Name	s3-arrival-notification	Display name	-
ARN	arn:aws:sns:us-east-1:590184043718:s3-arrival-notification	Topic owner	590184043718
Type	Standard		

Subscriptions (2)

ID	Endpoint	Status	Protocol
24c82236-d04c-4cff-bc31-169af71d3bd0	arn:aws:lambda:us-east-1:590184043718:...	Confirmed	LAMBDA
2d56ab9c-bc7f-422e-bcb2-f4181b897c15	ashishpanchalml@gmail.com	Confirmed	EMAIL

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

Create EventBridge Rule to trigger SNS when Glue Job Changes Status:

The screenshot shows the AWS EventBridge Rules console. The rule is named "ecommerce-glue-job-status". It is enabled, using the "default" event bus, and is of type "Standard". The event pattern is defined as follows:

```
1 {
2   "source": ["aws.glue"],
3   "detail-type": ["Glue Job State Change"]
4 }
```

There is a "Copy" button available for the event pattern.

Create final Lambda which will archive data after SNS notification:

The screenshot shows the AWS Lambda Functions console. The function is named "ecommerce-data-archive". It has an "SNS" trigger and no other triggers or destinations. The function ARN is listed as:

```
arn:aws:lambda:us-east-1:590184043718:function:ecommerce-data-archive
```

Ashish Panchal

Assignment – 6

E-Commerce Data Pipeline

The screenshot shows the AWS Lambda function editor interface. The top navigation bar includes File, Edit, Find, View, Go, Tools, Window, Test, Deploy, and a status message "Changes not deployed". Below the navigation is a search bar "Go to Anything (% P)" and tabs for "lambda_function", "Environment Var", and "Execution results". On the left, there's a sidebar titled "Environment" showing an "ecommerce-data-arch" environment with a "lambda_function.py" file selected. The main content area displays the Python code for the lambda function:

```

1 import json
2 import boto3
3
4 source_bucket_name = 'ecommerce-transaction-data'
5 destination_bucket_name = 'ecommerce-transactions-archive'
6
7 def lambda_handler(event, context):
8
9     print(f"Event: {event}")
10
11    # Extract the message from the SNS notification
12    message = json.loads(event['Records'][0]['Sns']['Message'])
13
14    # Access the "detail" dictionary within the message
15    detail = message['detail']
16
17    # Extract the value of the "state" key
18    glue_job_status = detail['state']
19
20    print(f"Glue Job Status: {glue_job_status}")
21
22    if glue_job_status == "SUCCEEDED":
23
24        if not source_bucket_name:
25            return {
26                'statusCode': 400,
27                'body': json.dumps('Missing source bucket name in event')
28            }
29
30        # Create S3 client
31        s3 = boto3.client('s3')
32
33        try:
34            # Call transfer_files function (defined below)
35            transfer_files(s3, source_bucket_name, destination_bucket_name)
36            return {
37                'statusCode': 200,
38                'body': json.dumps('Files transferred successfully!')
39            }
40        except Exception as e:
41            return {
42                'statusCode': 500,
43                'body': json.dumps(f'Error transferring files: {e}')
44            }
45
46    else:
47        return {
48            'statusCode': 500,
49            'body': json.dumps(f'Glue job {glue_job_status}: {event}')
50        }
51
52    def transfer_files(s3, source_bucket, destination_bucket):
53        for obj in s3.list_objects_v2(Bucket=source_bucket)['Contents']:
54            source_key = obj['Key']
55            destination_key = source_key # Copy the entire source key
56
57            # Optional: Modify destination_key if needed (e.g., rename file)
58
59            s3.copy_object(CopySource={'Bucket': source_bucket, 'Key': source_key},
60                           Bucket=destination_bucket, Key=destination_key)
61            print(f"Copied {source_key} to {destination_bucket}/{destination_key}")
62
63

```

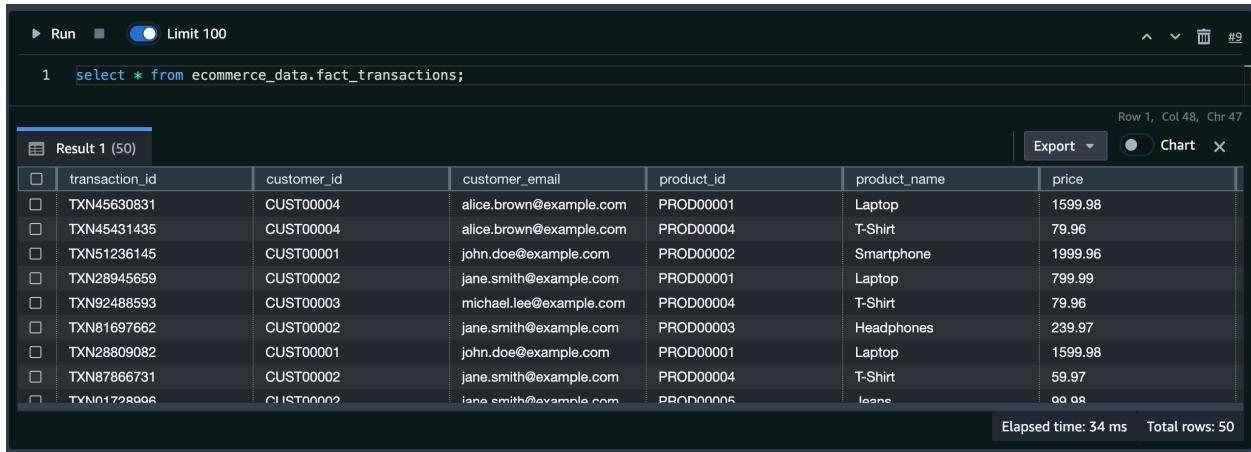
Create final S3 bucket where data is archived:

The screenshot shows the AWS S3 console. The left sidebar includes "Amazon S3", "Access Grants", "Access Points", "Object Lambda Access Points", "Multi-Region Access Points", "Batch Operations", "IAM Access Analyzer for S3", "Block Public Access settings for this account", "Storage Lens" (with "Dashboards" and "Storage Lens groups" listed), and "AWS Organizations controls". The main content area shows the "Buckets" section with a breadcrumb path: Amazon S3 > Buckets > ecommerce-transactions-archive > transactions_ > year=2024_ > month=3_ > day=23/. Below this, it says "day=23/" and "Objects (1) Info". A "Copy S3 URI" button is visible. The "Objects" tab is selected, showing a table with one item:

Name	Type	Last modified	Size	Storage class
transactions_2024-03-23.csv	csv	March 27, 2024, 21:55:07 (UTC-07:00)	859.0 B	Standard

Ashish Panchal
Assignment – 6
E-Commerce Data Pipeline

Check if data is populated in RedShift fact_transactions table:



The screenshot shows a Redshift query results interface. At the top, there is a toolbar with a 'Run' button, a 'Limit 100' toggle, and other standard database navigation icons. Below the toolbar is a code editor window containing the SQL query: 'select * from ecommerce_data.fact_transactions;'. To the right of the code editor, it says 'Row 1, Col 48, Chr 47'. Below the code editor is a table titled 'Result 1 (50)'. The table has columns: transaction_id, customer_id, customer_email, product_id, product_name, and price. The data in the table consists of 50 rows, each representing a transaction. The last row is partially visible. At the bottom right of the results area, it says 'Elapsed time: 34 ms Total rows: 50'.

	transaction_id	customer_id	customer_email	product_id	product_name	price
1	TXN45630831	CUST00004	alice.brown@example.com	PROD00001	Laptop	1599.98
2	TXN45431435	CUST00004	alice.brown@example.com	PROD00004	T-Shirt	79.96
3	TXN51236145	CUST00001	john.doe@example.com	PROD00002	Smartphone	1999.96
4	TXN28945659	CUST00002	jane.smith@example.com	PROD00001	Laptop	799.99
5	TXN92488593	CUST00003	michael.lee@example.com	PROD00004	T-Shirt	79.96
6	TXN81697662	CUST00002	jane.smith@example.com	PROD00003	Headphones	239.97
7	TXN28809082	CUST00001	john.doe@example.com	PROD00001	Laptop	1599.98
8	TXN87866731	CUST00002	jane.smith@example.com	PROD00004	T-Shirt	59.97
9	TXN01728096	CUST00002	jane.smith@example.com	PROD00005	Leans	aa aa

Once we execute the generate_mock_data Lambda, this whole pipeline flows as seen in the screenshots and executes/passes through different AWS services and finally data is archived in S3 bucket.

This completes the assignment.