Project Documentation

Step 1: EC2 Instance and MySQL Setup

1. **EC2 Instance Creation:** An EC2 instance was initially created in the Tokyo region, requiring a new keypair and region-specific security groups.
2. **Security Group Configuration:** A new security group was set up with permissions for SSH, HTTP, HTTPS, and MySQL access to facilitate project operations.
3. **MySQL 5.6 Installation:**

   Update yum packages:

   sudo yum update -y

   Install MySQL 5.7 community release:

   sudo yum install -y https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm

   Import MySQL GPG key:

   sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022

   Install MySQL community server:

   sudo yum install -y mysql-community-server

   Start MySQL service:

   sudo systemctl start mysqld

   Enable MySQL service to start on boot:

   sudo systemctl enable mysqld
4. **MySQL Password Configuration:**

   Retrieve temporary password:

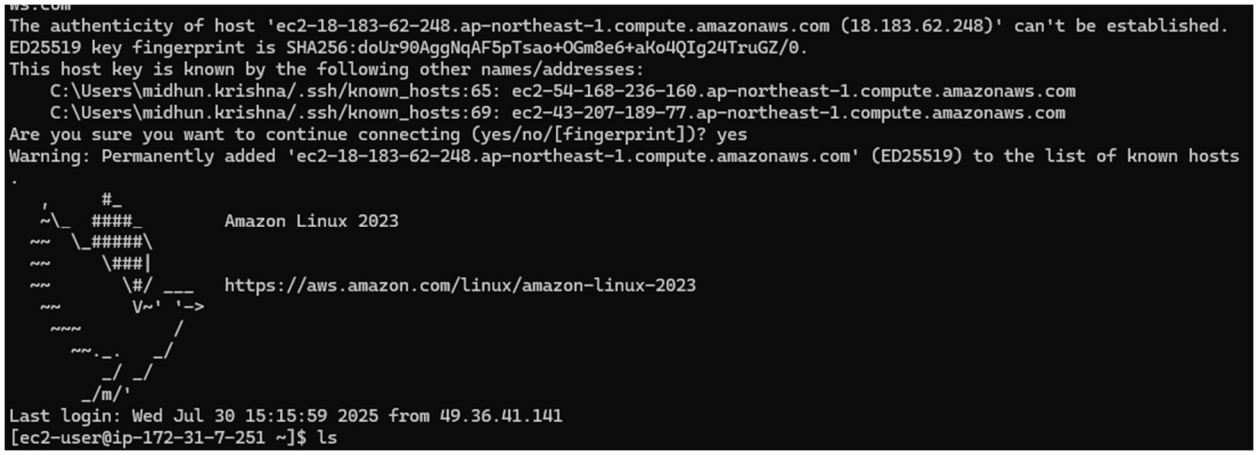
   sudo grep 'temporary password' /var/log/mysqld.log

   Connect to MySQL:

   mysql -u root -p

Change temporary password to Midhunproj@1:

ALTER USER 'root'@'localhost' IDENTIFIED BY 'Midhunproj@1';



Database and Table Creation in EC2 Instance

1. **Create Database:** create database midhun_proj;
2. **Select Database:** use midhun_proj;
3. **Create Dimension Tables:**
    - futurecart_calendar_details

SQL

CREATE TABLE futurecart_calendar_details (

calendar_date DATE,

date_desc VARCHAR(50),

week_day_nbr SMALLINT,

week_number SMALLINT,

week_name VARCHAR(50),

year_week_number INT,

month_number SMALLINT,

month_name VARCHAR(50),

quarter_number SMALLINT,

quarter_name VARCHAR(50),

half_year_number SMALLINT,

half_year_name VARCHAR(50),

geo_region_cd CHAR(2)

);

- futurecart_call_center_details

SQL

CREATE TABLE futurecart_call_center_details (

call_center_id VARCHAR(10),

call_center_vendor VARCHAR(50),

location VARCHAR(50),

country VARCHAR(50)

);

- futurecart_case_category_details

SQL

CREATE TABLE futurecart_case_category_details (

category_key VARCHAR(10),

sub_category_key VARCHAR(10),

category_description VARCHAR(50),

sub_category_description VARCHAR(50),

priority VARCHAR(10)

);

- futurecart_case_country_details

SQL

CREATE TABLE futurecart_case_country_details (

id INT,

Name VARCHAR(75),

Alpha_2 VARCHAR(2),

Alpha_3 VARCHAR(3)

);

- futurecart_case_priority_details

SQL

```
CREATE TABLE futurecart_case_priority_details (

Priority_key VARCHAR(5),

priority VARCHAR(20),

severity VARCHAR(100),

SLA VARCHAR(100)

);
```

- futurecart_employee_details

SQL

```
CREATE TABLE futurecart_employee_details (

emp_key INT,

first_name VARCHAR(100),

last_name VARCHAR(100),

email VARCHAR(100),

gender VARCHAR(10),

ldap VARCHAR(50),

hire_date DATE,

manager VARCHAR(50)

);
```

- futurecart_product_details

SQL

```
CREATE TABLE futurecart_product_details (
product_id VARCHAR(50),
department VARCHAR(100),
brand VARCHAR(100),
commodity_desc VARCHAR(100),
sub_commodity_desc VARCHAR(100)
);
```

- futurecart_survey_question_details

SQL

```
CREATE TABLE futurecart_survey_question_details (
question_id VARCHAR(50),
question_desc VARCHAR(755),
response_type VARCHAR(50),
`range` VARCHAR(50),
negative_response_range VARCHAR(50),
neutral_response_range VARCHAR(50),
positive_response_range VARCHAR(50)
);
```

(Note:range is a reserved keyword in MySQL, hence the backticks).

3.1 Loading Data into MySQL from S3

1. **Create MySQL files directory:** sudo mkdir -p /var/lib/mysql-files/
2. **Copy data files from S3 to EC2 instance:**
    - aws s3 cp s3://midhun-tokyo/futurecart_calendar_details.txt /var/lib/mysql-files/
    - aws s3 cp s3://midhun-tokyo/futurecart_call_center_details.txt /var/lib/mysql-files/

- aws s3 cp s3://midhun-tokyo/futurecart_case_category_details.txt /var/lib/mysql-files/
- aws s3 cp s3://midhun-tokyo/futurecart_case_country_details.txt /var/lib/mysql-files/
- aws s3 cp s3://midhun-tokyo/futurecart_case_priority_details.txt /var/lib/mysql-files/
- aws s3 cp s3://midhun-tokyo/futurecart_employee_details.txt /var/lib/mysql-files/
- aws s3 cp s3://midhun-tokyo/futurecart_product_details.txt /var/lib/mysql-files/
- aws s3 cp s3://midhun-tokyo/futurecart_survey_question_details.txt /var/lib/mysql-files/

3. **Load data into MySQL tables:**
   - futurecart_calendar_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_calendar_details.txt' INTO TABLE futurecart_calendar_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
Database changed
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_calendar_details.txt'
    -> INTO TABLE futurecart_calendar_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 21553 rows affected (0.21 sec)
Records: 21553  Deleted: 0  Skipped: 0  Warnings: 0
```

   - futurecart_call_center_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_call_center_details.txt' INTO TABLE futurecart_call_center_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_call_center_details.txt'
    -> INTO TABLE futurecart_call_center_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 25 rows affected (0.00 sec)
```

   - futurecart_case_category_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_case_category_details.txt' INTO TABLE

futurecart_case_category_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_case_category_details.txt'
    -> INTO TABLE futurecart_case_category_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 27 rows affected (0.00 sec)
Records: 27  Deleted: 0  Skipped: 0  Warnings: 0
```

- futurecart_case_country_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_case_country_details.txt' INTO TABLE futurecart_case_country_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_case_country_details.txt'
    -> INTO TABLE futurecart_case_country_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 193 rows affected (0.01 sec)
Records: 193  Deleted: 0  Skipped: 0  Warnings: 0
```

- futurecart_case_priority_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_case_priority_details.txt' INTO TABLE futurecart_case_priority_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_case_priority_details.txt'
    -> INTO TABLE futurecart_case_priority_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 25 rows affected (0.00 sec)
Records: 25  Deleted: 0  Skipped: 0  Warnings: 0
```

- futurecart_employee_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_employee_details.txt' INTO TABLE futurecart_employee_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_employee_details.txt'
    -> INTO TABLE futurecart_employee_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 300024 rows affected (2.18 sec)
Records: 300024  Deleted: 0  Skipped: 0  Warnings: 0
```

- futurecart_product_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_product_details.txt' INTO TABLE futurecart_product_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_product_details.txt'
    -> INTO TABLE futurecart_product_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 92353 rows affected (0.45 sec)
Records: 92353  Deleted: 0  Skipped: 0  Warnings: 0
```

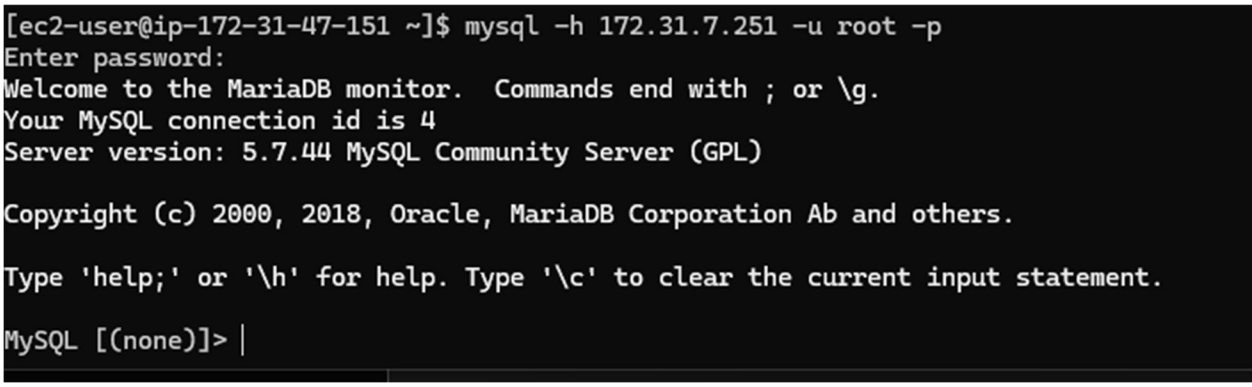
- futurecart_survey_question_details: LOAD DATA INFILE '/var/lib/mysql-files/futurecart_survey_question_details.txt' INTO TABLE futurecart_survey_question_details FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/futurecart_product_details.txt'
    -> INTO TABLE futurecart_product_details
    -> FIELDS TERMINATED BY '\t'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
Query OK, 92353 rows affected (0.45 sec)
Records: 92353  Deleted: 0  Skipped: 0  Warnings: 0
```

3.2 EMR Cluster Setup and Sqoop Import to Hive

1. **Create EMR Cluster:** An EMR cluster was created with Hadoop, Sqoop, Spark, and Hive.
2. **Grant MySQL Privileges to EMR Cluster:**
    - GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'Midhunproj@1';
    - Modified MySQL config file to set bind-address to 0.0.0.0 to allow external connections.

- Restart MySQL service: sudo systemctl restart mysql
- Verify MySQL is listening on 0.0.0.0:3306: sudo netstat -tulnp | grep mysqld
- Connect to MySQL from EMR cluster: mysql -h 172.31.7.251 -u root -p



3. **Install JDBC Driver for Sqoop:**

- Download JDBC driver: wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.49.tar.gz
- Extract archive: tar -xvzf mysql-connector-java-5.1.49.tar.gz
- Copy JAR to Sqoop lib directory: sudo cp mysql-connector-java-5.1.49/mysql-connector-java-5.1.49-bin.jar /usr/lib/sqoop/lib/

4. **HDFS Permissions for EC2 user:**
    - Create user directory: sudo -u hdfs hdfs dfs -mkdir /user/ec2-user
    - Change ownership: sudo -u hdfs hdfs dfs -chown ec2-user:ec2-user /user/ec2-user
5. **Sqoop Import Data to Hive Tables:**
    - futurecart_calendar_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_calendar_details --hive-import --hive-table futurecart_calendar_details --create-hive-table --m 1
    - futurecart_call_center_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_call_center_details --hive-import --hive-table futurecart_call_center_details --create-hive-table --m 1
    - futurecart_case_category_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_case_category_details --hive-import --hive-table futurecart_case_category_details --create-hive-table --m 1
    - futurecart_case_country_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_case_country_details --hive-import --hive-table futurecart_case_country_details --create-hive-table --m 1
    - futurecart_case_priority_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_case_priority_details --hive-import --hive-table futurecart_case_priority_details --create-hive-table --m 1

- futurecart_employee_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_employee_details --hive-import --hive-table futurecart_employee_details --create-hive-table --m 1
- futurecart_product_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_product_details --hive-import --hive-table futurecart_product_details --create-hive-table --m 1
- futurecart_survey_question_details: sqoop import --connect jdbc:mysql://172.31.7.251:3306/midhun_proj --username root --password Midhunproj@1 --table futurecart_survey_question_details --hive-import --hive-table futurecart_survey_question_details --create-hive-table --m 1

```
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: true
hive> show tables;
OK
futurecart_calendar_details
futurecart_call_center_details
futurecart_case_category_details
futurecart_case_country_details
futurecart_case_priority_details
futurecart_employee_details
futurecart_product_details
futurecart_survey_question_details
Time taken: 0.67 seconds, Fetched: 8 row(s)
hive>
```

Uploading Dimension Data to S3 using Spark

1. **Create Python script (dimension_export.py):**

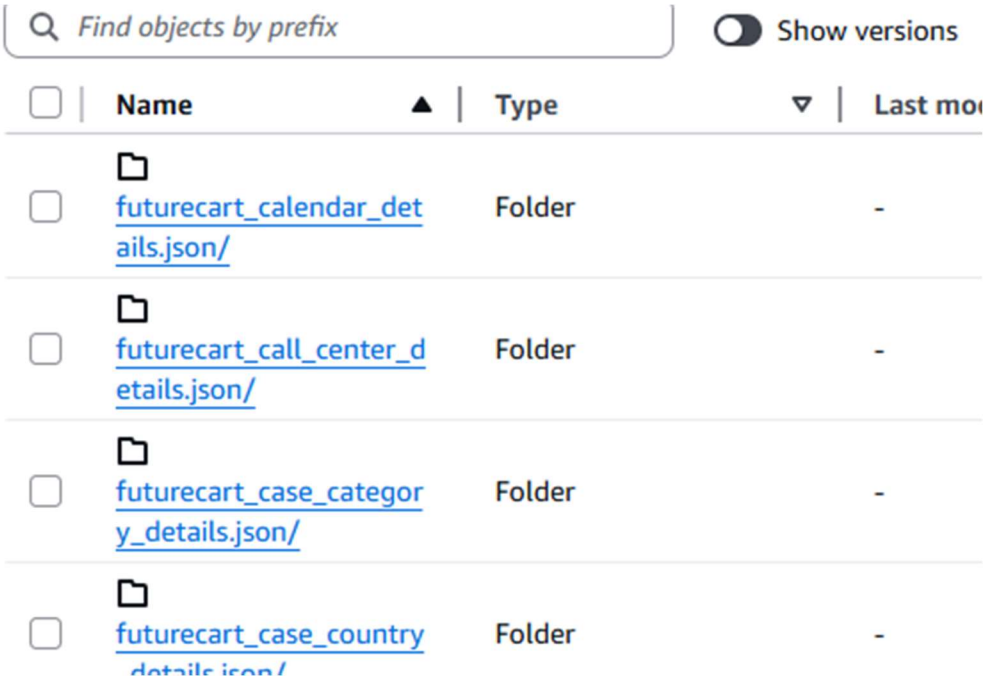
Python

```
import sys
from pyspark.sql import SparkSession
from pyspark.sql.utils import AnalysisException

dimension_tables = [
    "futurecart_calendar_details",
    "futurecart_call_center_details",
    "futurecart_case_category_details",
```

```
    "futurecart_case_country_details",
    "futurecart_case_priority_details",
    "futurecart_employee_details",
    "futurecart_product_details",
    "futurecart_survey_question_details"
]

s3_output_base_path = "s3a://midhun-tokyo/dimension_data_json/"

def process_table(table_name, spark, s3_path):
    try:
        print(f"Processing table: {table_name}")
        df = spark.sql(f"SELECT * FROM {table_name}")
        output_path = f"{s3_path}{table_name}.json"
        df.write.mode("overwrite").json(output_path)
        print(f"Successfully wrote data from {table_name} to {output_path}")
    except AnalysisException as e:
        print(f"Error processing table {table_name}: {e}")
        print("Please check if the Hive table exists and if the Spark session has access to it.")
    except Exception as e:
        print(f"An unexpected error occurred while processing table {table_name}: {e}")

if __name__ == "__main__":
    spark = SparkSession.builder \
        .appName("Hive to S3 Dimension Data Export") \
        .enableHiveSupport() \
```

```
    .getOrCreate()

for table in dimension_tables:
    process_table(table, spark, s3_output_base_path)

spark.stop()
print("All tables have been processed. Spark session stopped.")
```

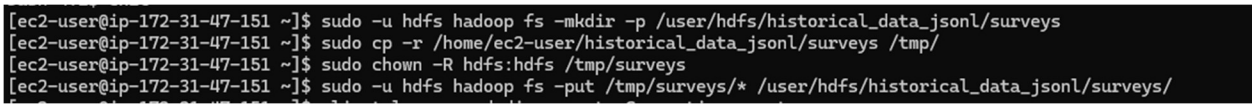
2. **Run Spark job:** spark-submit dimension_export.py



3.4 Generating and Uploading Historical Data to HDFS

1. **Fetch historical data generation script:** aws s3 cp s3://midhun-tokyo/generate_historical_data.py ~/generate_historical_data.py
2. **Run script:** python3 generate_historical_data.py
3. **Upload historical data (cases) to HDFS:**
    - Create HDFS directory: sudo -u hdfs hadoop fs -mkdir -p /user/hdfs/historical_data_jsonl/cases

- Copy local files to /tmp: sudo cp -r /home/ec2-user/historical_data_jsonl/cases /tmp/
- Change ownership to hdfs: sudo chown -R hdfs:hdfs /tmp/cases
- Put files into HDFS: sudo -u hdfs hadoop fs -put /tmp/cases/* /user/hdfs/historical_data_jsonl/cases/



4. **Upload historical data (surveys) to HDFS:**
    - Create HDFS directory: sudo -u hdfs hadoop fs -mkdir -p /user/hdfs/historical_data_jsonl/surveys
    - Copy local files to /tmp: sudo cp -r /home/ec2-user/historical_data_jsonl/surveys /tmp/
    - Change ownership to hdfs: sudo chown -R hdfs:hdfs /tmp/surveys
    - Put files into HDFS: sudo -u hdfs hadoop fs -put /tmp/surveys/* /user/hdfs/historical_data_jsonl/surveys/

3.5 Creating External Hive Tables for Historical Data

1. **futurecart_case_details:**

SQL

```
CREATE EXTERNAL TABLE IF NOT EXISTS futurecart_case_details (
case_no varchar(700),
create_timestamp varchar(700),
last_modified_timestamp varchar(700),
created_employee_key varchar(700),
call_center_id varchar(700),
status varchar(700),
category varchar(700),
sub_category varchar(700),
communication_mode varchar(700),
```

country_cd varchar(700),

product_code varchar(700)

)

ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'

STORED AS TEXTFILE

LOCATION '/user/hdfs/historical_data_jsonl/cases/';

2. **futurecart_case_survey_details:**

SQL

CREATE EXTERNAL TABLE IF NOT EXISTS futurecart_case_survey_details (

survey_id varchar(700),

case_no varchar(700),

survey_timestamp varchar(700),

Q1 varchar(700),

Q2 varchar(700),

Q3 varchar(700),

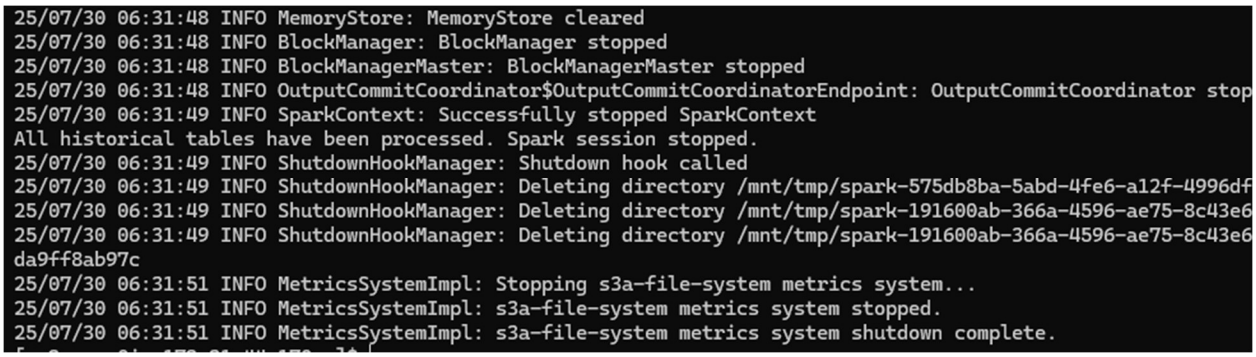
Q4 varchar(700),

Q5 varchar(700)

)

ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'

LOCATION '/user/hdfs/historical_data_jsonl/surveys/';



3.6 Uploading Historical Data from Hive to S3 using Spark

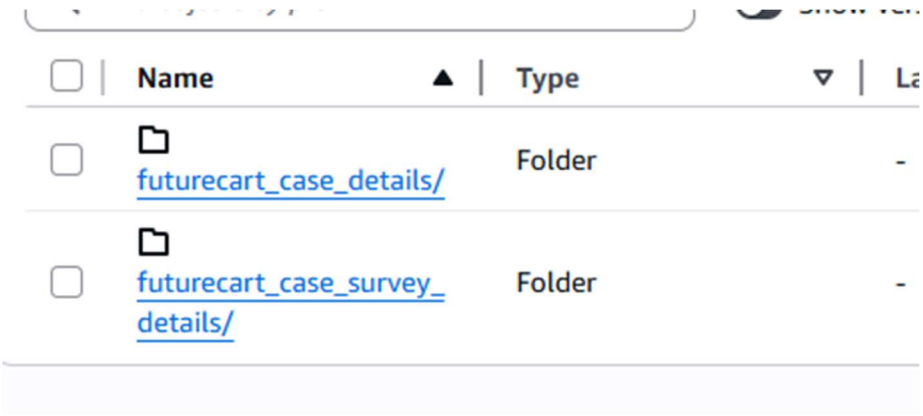
1. **Create Python script (bucket_export.py):**

Python

```
import sys
from pyspark.sql import SparkSession
from pyspark.sql.utils import AnalysisException

historical_tables = [
  "futurecart_case_details",
  "futurecart_case_survey_details"
]

s3_output_base_path = "s3a://midhun-tokyo/historical_data_json_export/"

def process_table(table_name, spark, s3_path):
  try:
    print(f"Processing historical table: {table_name}")
    df = spark.sql(f"SELECT * FROM {table_name}")
    output_path = f"{s3_path}{table_name}/"
    df.write.mode("overwrite").json(output_path)
    print(f"Successfully wrote data from {table_name} to {output_path}")
  except AnalysisException as e:
    print(f"Error processing table {table_name}: {e}")
    print("Please check if the Hive table exists and if the Spark session has access to it.")
  except Exception as e:
    print(f"An unexpected error occurred while processing table {table_name}: {e}")
```

```
if __name__ == "__main__":
    spark = SparkSession.builder \
        .appName("Historical Hive to S3 JSON Export") \
        .enableHiveSupport() \
        .getOrCreate()

    for table in historical_tables:
        process_table(table, spark, s3_output_base_path)

    spark.stop()
    print("All historical tables have been processed. Spark session stopped.")
```

2. **Download Hive HCatalog Core JAR:** wget https://repo1.maven.org/maven2/org/apache/hive/hcatalog/hive-hcatalog-core/3.1.3/hive-hcatalog-core-3.1.3.jar
3. **Run Spark job with JAR:** spark-submit --jars /home/ec2-user/hive-hcatalog-core-3.1.3.jar bucket_export.py



Step 4: Redshift Setup and Data Loading

1. **Create Tables in Redshift:** 10 tables were created in Redshift.

2. **Load Data from S3 to Redshift:**

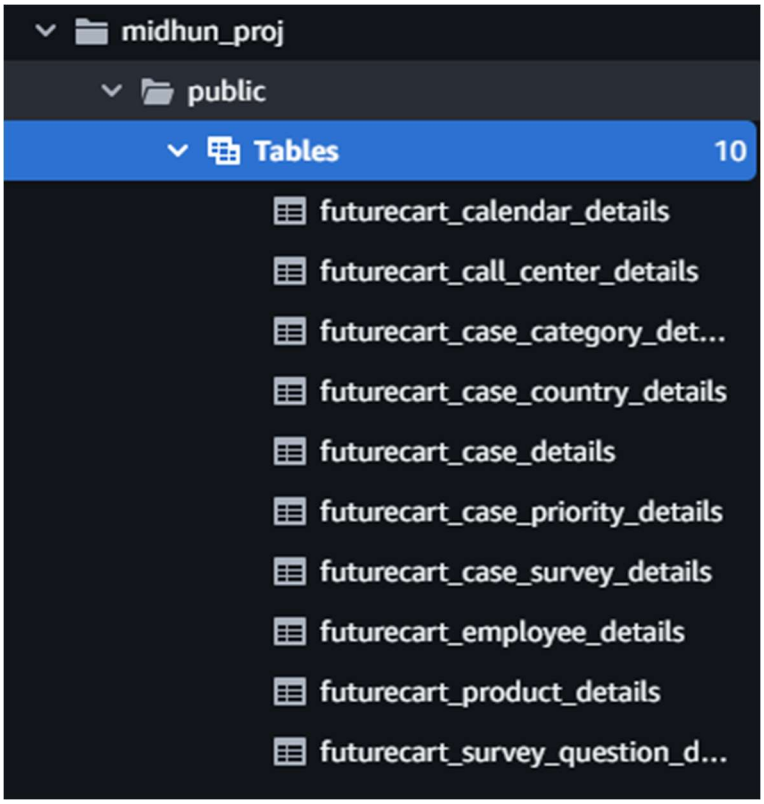
SQL

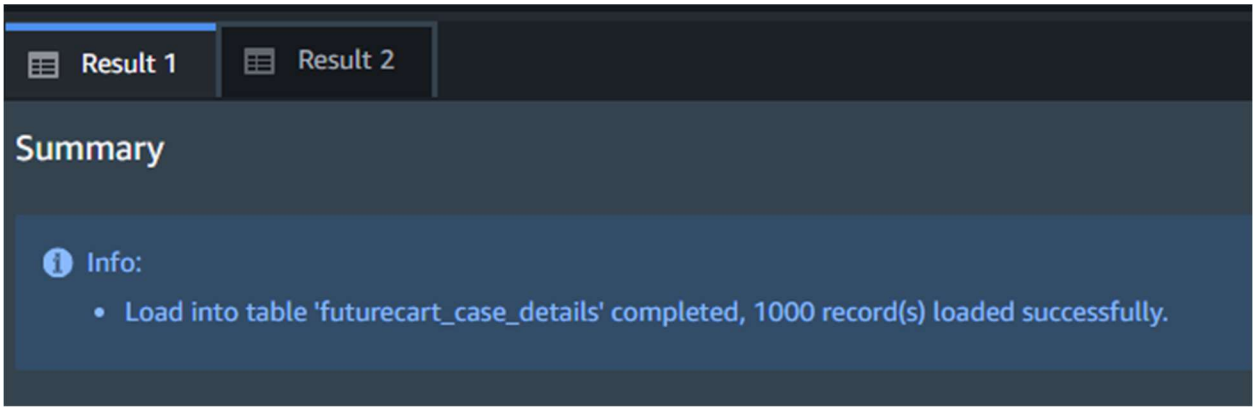
COPY midhun_proj.public.futurecart_product_details FROM 's3://midhun-tokyo/dimension_data_json/futurecart_product_details.json/'

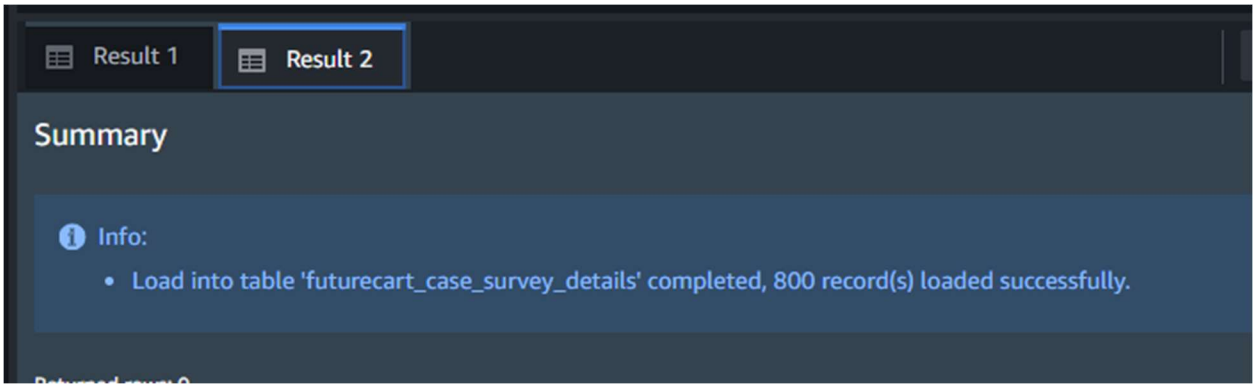
IAM_ROLE 'arn:aws:iam::008673239246:role/Midhun-Redshift-s3fullaccess'

JSON 'auto';

- Loading reported successful for futurecart_case_details (1000 records) and futurecart_case_survey_details (800 records).







Step 5: Real-Time Data Generation and Kinesis Integration

1. **Real-Time Data Generation to Kinesis:** A script was run on an EC2 instance to send real-time data to a Kinesis data stream. The region and Kinesis data stream name were configured in the Python file.

2. **Spark Script for Kinesis to Redshift:** A Spark script was used to capture data from Kinesis and load it into Redshift.

   Python code:

   from pyspark.sql import SparkSession

   from pyspark.sql.functions import col, from_json, to_timestamp, when, lit

   from pyspark.sql.types import StructType, StructField, StringType, IntegerType

```
import os
import shutil
import boto3
import json
import time
import threading
from queue import Queue
import traceback


def start_spark_session():
    """
    Starts and gives back a SparkSession set up for handling data streams.
    It makes Spark run faster for big data tasks.
    """
    return SparkSession.builder \
        .appName("LiveDataProcessor") \
        .config("spark.jars.packages", "com.amazon.redshift:redshift-jdbc42:2.1.0.9") \
        .config("spark.sql.adaptive.enabled", "true") \
        .config("spark.sql.adaptive.coalescePartitions.enabled", "true") \
        .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
        .getOrCreate()


def define_data_structures():
    """
    Sets up the exact layouts (blueprints) for two types of incoming data:
    customer calls and customer survey answers.
```

These blueprints help Spark understand the data correctly.

"""

```
customer_call_data_structure = StructType([
    StructField("call_id", StringType(), True),
    StructField("agent_id", StringType(), True),
    StructField("service_center_id", StringType(), True),
    StructField("status", StringType(), True),
    StructField("category", StringType(), True),
    StructField("sub_category", StringType(), True),
    StructField("talk_channel", StringType(), True),
    StructField("country_code", StringType(), True),
    StructField("product_code", StringType(), True),
    StructField("last_change_time", StringType(), True),
    StructField("create_time", StringType(), True)
])

customer_survey_data_structure = StructType([
    StructField("survey_id", StringType(), True),
    StructField("call_id", StringType(), True),
    StructField("survey_time", StringType(), True),
    StructField("score_q1", IntegerType(), True),
    StructField("score_q2", IntegerType(), True),
    StructField("score_q3", IntegerType(), True),
    StructField("comment_q4", StringType(), True),
    StructField("score_q5", IntegerType(), True)
```

```
    ])

    return customer_call_data_structure, customer_survey_data_structure

class KinesisDataFetcher:
    """
    Handles getting data from an AWS Kinesis stream safely, even with multiple tasks at once.
    It keeps pulling new data and puts it into a waiting list (queue) for processing.
    """

    def __init__(self, stream_name, aws_region):
        """
        Sets up the KinesisDataFetcher with stream details and connects to Kinesis.

        Args:
            stream_name (str): The name of the Kinesis stream to get data from.
            aws_region (str): The AWS region where the Kinesis stream is.
        """
        self.stream_name = stream_name
        self.aws_region = aws_region
        self.kinesis_client = boto3.client('kinesis', region_name=aws_region)
        self.data_storage_queue = Queue()
        self.is_running = False
        self.shard_cursors = {} # Keeps track of where we left off in each Kinesis shard
```

```
  def start_fetching(self):
    """
    Starts the background task that continuously gets data from Kinesis.
    """
    self.is_running = True
    fetch_thread = threading.Thread(target=self._data_fetch_loop)
    fetch_thread.daemon = True # Lets the main program close even if this task is still running
    fetch_thread.start()
    return fetch_thread

  def stop_fetching(self):
    """
    Tells the data fetching task to stop gently.
    """
    self.is_running = False

  def get_stored_data_batch(self, max_items=100):
    """
    Gets a group of records from the internal data waiting list.

    Args:
      max_items (int): The most records to get in one group.

    Returns:
      list: A list of data items, each being a Kinesis record.
```

```
        """
        batch_of_data = []
        while not self.data_storage_queue.empty() and len(batch_of_data) < max_items:
            batch_of_data.append(self.data_storage_queue.get())
        return batch_of_data

    def _data_fetch_loop(self):
        """
        The main loop for the background data fetching task. It keeps
        getting records from Kinesis sections (shards) and adds them to the queue.
        """
        print("🔄 Starting Kinesis data fetching task...")

        while self.is_running:
            try:
                # Get stream details to find active sections
                stream_info =
self.kinesis_client.describe_stream(StreamName=self.stream_name)
                shards = stream_info['StreamDescription']['Shards']

                # Go through each section to get records
                for shard in shards: # Checking all sections
                    shard_id = shard['ShardId']

                    # Get or refresh the section marker (cursor)
                    if shard_id not in self.shard_cursors:
```

```
        response = self.kinesis_client.get_shard_iterator(
            StreamName=self.stream_name,
            ShardId=shard_id,
            ShardIteratorType='LATEST' # Start from the newest records
        )
        self.shard_cursors[shard_id] = response['ShardIterator']

    # Get records using the current section marker
    try:
        response = self.kinesis_client.get_records(
            ShardIterator=self.shard_cursors[shard_id],
            Limit=50 # Max records to get per try
        )

        records = response.get('Records', [])
        if records:
            print(f" 📦 Got {len(records)} records from section {shard_id}")

        for record in records:
            try:
                raw_record_data = record['Data']
                if isinstance(raw_record_data, bytes):
                    raw_record_data = raw_record_data.decode('utf-8')

                # Save important info along with the main data
                processed_record = {
```

```
                    "json_data": str(raw_record_data),
                    "partitionKey": record.get('PartitionKey', ''),
                    "sequenceNumber": record.get('SequenceNumber', ''),
                    "approximateArrivalTimestamp":
str(record.get('ApproximateArrivalTimestamp', ''))
                }
                self.data_storage_queue.put(processed_record)

            except Exception as e:
                print(f" ⚠️  Problem handling one record: {str(e)}")

        # Update the section marker for the next fetch
        if response.get('NextShardIterator'):
            self.shard_cursors[shard_id] = response['NextShardIterator']
        else:
            # If the next marker is empty, the current one has expired.
            # Get a new one starting from the latest point.
            response = self.kinesis_client.get_shard_iterator(
                StreamName=self.stream_name,
                ShardId=shard_id,
                ShardIteratorType='LATEST'
            )
            self.shard_cursors[shard_id] = response['ShardIterator']

    except Exception as e:
        print(f" ⚠️  Problem reading records from section {shard_id}: {str(e)}")
```

```
            # Wait a bit before getting the next round of data
            time.sleep(5)

        except Exception as e:
            print(f" ⚠️  Problem during Kinesis data fetching loop: {str(e)}")
            time.sleep(10) # Wait longer if there's a big error

def sort_and_understand_data(raw_records, call_data_structure, survey_data_structure):
    """
    Takes raw Kinesis records, tries to turn their data into JSON,
    and puts them into "customer call" or "customer survey" groups
    based on what key words are inside.

    Args:
        raw_records (list): A list of raw Kinesis data items.
        call_data_structure (StructType): The blueprint for customer call data.
        survey_data_structure (StructType): The blueprint for customer survey data.

    Returns:
        tuple: Two lists: (call_events, survey_events).
               Each list holds the understood JSON data.
    """
    call_events = []
    survey_events = []
```

```
  for record in raw_records:
    json_text = record['json_data']

    # A simple way to guess the data type based on words it contains
    is_survey = any(field.name in json_text for field in survey_data_structure.fields if field.dataType == IntegerType()) or \
                any(field_name in json_text for field_name in ['survey_id', 'survey_time', '"Q1":', '"Q2":'])
    is_call = any(field.name in json_text for field in call_data_structure.fields if field.dataType == StringType()) or \
                any(field_name in json_text for field_name in ['call_center_id', 'talk_channel', 'created_employee_key', '"status":'])

    try:
      event_data = json.loads(json_text)
      if is_survey:
        survey_events.append(event_data)
      elif is_call:
        call_events.append(event_data)
      else:
        print(f" ❓ Data not put into a group: {json_text[:50]}...")
    except json.JSONDecodeError:
      print(f" ⚠️ Bad JSON format for record: {json_text[:50]}...")
    except Exception as e:
      print(f" ⚠️ Problem understanding record: {str(e)} - Data: {json_text[:50]}...")
```

```
    return call_events, survey_events


def send_data_to_warehouse_batch(spark_session, event_list, table_name, data_structure, warehouse_login_info):
    """
    Sends a group of processed data items to the chosen Redshift table.
    It also changes time formats to fit the table's needs.

    Args:
        spark_session (SparkSession): The active SparkSession.
        event_list (list): A list of data items, each being an event.
        table_name (str): The name of the Redshift table to write to.
        data_structure (StructType): The Spark blueprint matching the table.
        warehouse_login_info (dict): Details for connecting to Redshift.
    """
    if not event_list:
        return

    try:
        # Make a Spark DataFrame from the list of events and apply the blueprint
        df_to_save = spark_session.createDataFrame(event_list, data_structure)

        # Change time formats based on the table name
        if table_name == "customer_interactions_data":
            df_to_save = df_to_save.select(
                col("call_id"),
```

```
            col("agent_id"),
            col("service_center_id"),
            col("status"),
            col("category"),
            col("sub_category"),
            col("talk_channel"),
            col("country_code"),
            col("product_code"),
            to_timestamp(col("last_change_time"), "yyyy-MM-dd
HH:mm:ss").alias("last_change_time"),
            to_timestamp(col("create_time"), "yyyy-MM-dd
HH:mm:ss").alias("create_time")
        )
    elif table_name == "customer_feedback_data":
        df_to_save = df_to_save.select(
            col("survey_id"),
            col("call_id"),
            to_timestamp(col("survey_time"), "yyyy-MM-dd
HH:mm:ss").alias("survey_time"),
            col("score_q1"),
            col("score_q2"),
            col("score_q3"),
            col("comment_q4"),
            col("score_q5")
        )

    # Send the DataFrame to Redshift using JDBC (Java Database Connectivity)
```

```
        df_to_save.write \
            .format("jdbc") \
            .option("url", warehouse_login_info["url"]) \
            .option("dbtable", table_name) \
            .option("user", warehouse_login_info["user"]) \
            .option("password", warehouse_login_info["password"]) \
            .option("driver", warehouse_login_info["driver"]) \
            .mode("append") \
            .save()

        print(f"✅ Successfully sent {len(event_list)} records to {table_name}")

    except Exception as e:
        print(f"❌ Problem sending to {table_name}: {str(e)}")
        traceback.print_exc()

def check_warehouse_connection(spark_session, warehouse_login_info):
    """
    Tests if we can connect to the Redshift data warehouse by running a simple query.

    Args:
        spark_session (SparkSession): The active SparkSession.
        warehouse_login_info (dict): Details for connecting to Redshift.

    Returns:
        bool: True if the connection works, False otherwise.
```

```
    """
    print("🔍 Checking data warehouse connection...")

    try:
        # Try to read from a dummy table or run a simple select to test the connection
        test_df = spark_session.read \
            .format("jdbc") \
            .option("url", warehouse_login_info["url"]) \
            .option("dbtable", "(SELECT 1 as connection_test_col) as test_query") \
            .option("user", warehouse_login_info["user"]) \
            .option("password", warehouse_login_info["password"]) \
            .option("driver", warehouse_login_info["driver"]) \
            .load()

        # Make Spark actually try to connect
        result = test_df.collect()
        print(f"✅ Data warehouse connection successful!")
        return True

    except Exception as e:
        print(f"❌ Data warehouse connection failed: {str(e)}")
        traceback.print_exc()
        return False

def run_main_data_pipeline():
    """
```

This is the main part of the live data streaming system.

It manages getting data from Kinesis, processing it with Spark, and saving it to Redshift.

"""

print(" 🚀 Starting Live Data Processing Pipeline...")

print("=" * 60)

Start Spark Session

spark = start_spark_session()

spark.sparkContext.setLogLevel("WARN") # Set Spark logging to show fewer messages

Set up data blueprints

customer_call_data_structure, customer_survey_data_structure = define_data_structures()

Data Warehouse (Redshift) connection details

warehouse_login_info = {

"url": "jdbc:redshift://midhun-proj.008673239246.ap-northeast-1.redshift-serverless.amazonaws.com:5439/midhun_proj",

"user": "admin",

"password": "Midhunproj1",

"driver": "com.amazon.redshift.jdbc.Driver"

}

print(f" 🔗 Data Warehouse Address: {warehouse_login_info['url']}")

```
print(f" User: {warehouse_login_info['user']}")

# Check if we can connect to the data warehouse
if not check_warehouse_connection(spark, warehouse_login_info):
    print("❌ Data warehouse connection check failed. Stopping pipeline.")
    return

# Start getting data from Kinesis
data_fetcher = KinesisDataFetcher("Midhun-proj", "ap-northeast-1")
fetch_thread_link = data_fetcher.start_fetching()

print("=" * 60)
print(" 🎯 LIVE DATA SYSTEM STATUS")
print("=" * 60)
print(" ✅ Kinesis Source Stream: Midhun-proj")
print(" ✅ How it Works: Background fetching + small-group Spark processing")
print(" ✅ Customer Call Data → customer_interactions_data table")
print(" ✅ Customer Survey Data → customer_feedback_data table")
print(" ✅ Processing Every: 15 seconds")
print(" ✅ Data Safety: No special custom functions, direct Spark actions")
print("=" * 60)
print(" 🔄 System running... Press Ctrl+C to stop")
print("=" * 60)

try:
```

```
    processed_groups_count = 0
    while True:
      # Get a group of raw data from the Kinesis fetcher's storage
      current_raw_data_batch = data_fetcher.get_stored_data_batch()

      if current_raw_data_batch:
        print(f" 📦 Processing group {processed_groups_count} with {len(current_raw_data_batch)} records...")

        # Understand and sort the raw data into different event types
        call_events, survey_events = sort_and_understand_data(
          current_raw_data_batch, customer_call_data_structure, customer_survey_data_structure
        )

        # Send the sorted data to their correct Redshift tables
        if call_events:
          print(f" 📋 Found {len(call_events)} customer call events")
          send_data_to_warehouse_batch(
            spark, call_events, "customer_interactions_data",
            customer_call_data_structure, warehouse_login_info
          )

        if survey_events:
          print(f" 📊 Found {len(survey_events)} customer survey events")
          send_data_to_warehouse_batch(
```

```
            spark, survey_events, "customer_feedback_data",
            customer_survey_data_structure, warehouse_login_info
        )

        if not call_events and not survey_events:
            print(f" ⚠️ No good data found in this group.")

        processed_groups_count += 1
    else:
        print(" ⏳ No new data in storage, waiting for Kinesis to get more...")

    # Wait for a set time before processing the next group
    time.sleep(15)

except KeyboardInterrupt:
    print("\n 🛑 System stopped by user (Ctrl+C).")
except Exception as e:
    print(f" ❌ A major problem happened in the system: {str(e)}")
    traceback.print_exc() # Show full error details for fixing
finally:
    print(" 🔄 Stopping Kinesis data fetcher...")
    data_fetcher.stop_fetching()
    fetch_thread_link.join(timeout=5) # Wait for the fetching task to finish

    print(" 🔄 Closing Spark session...")
```

```
    spark.stop() # Stop Spark
    print("✅ System shutdown complete.")

if __name__ == "__main__":
    run_main_data_pipeline()
```

- **Spark-submit command:** spark-submit --packages "com.amazon.redshift:redshift-jdbc42:2.1.0.9" script.py

3. **Redshift Table Definition for Real-Time Cases:**

SQL

```
CREATE TABLE IF NOT EXISTS futurecart_case_details (
case_no varchar(700),
create_timestamp varchar(700),
last_modified_timestamp varchar(700),
created_employee_key varchar(700),
call_center_id varchar(700),
status varchar(700),
category varchar(700),
sub_category varchar(700),
communication_mode varchar(700),
country_cd varchar(700),
product_code varchar(700)
)
```

Initial data:

```
select * from futurecart_case_survey_details;
```

Row 50, Col 1, Chr 2900

Result 1 (1600) — Export — Chart

survey_id	case_no	survey_timestamp	q1	q2
S-514032	696729	2025-07-21 05:10:51	1	4
S-514264	649950	2025-07-21 05:10:51	9	8
S-571629	617917	2025-07-21 05:10:51	5	2
S-560236	682240	2025-07-21 05:10:51	10	4
S-554023	698346	2025-07-21 05:10:51	9	8

1 to 100 of 1600 rows — Query ID 5346 — Elapsed time: 144 ms — Total rows: 1600

Row 51, Col 39, Chr 2893

Result 1 (2014) — Export — Chart

case_no	create_timestamp	last_modified_timest...	created_employee_key	call_center_id
610253	2025-07-21 05:10:51	2025-07-21 05:10:51	223210	C-114
605965	2025-07-21 05:10:51	2025-07-21 05:10:51	277102	C-115
613774	2025-07-21 05:10:51	2025-07-21 05:10:51	262430	C-108
620044	2025-07-21 05:10:51	2025-07-21 05:10:51	233478	C-102
647629	2025-07-21 05:10:51	2025-07-21 05:10:51	270911	C-106

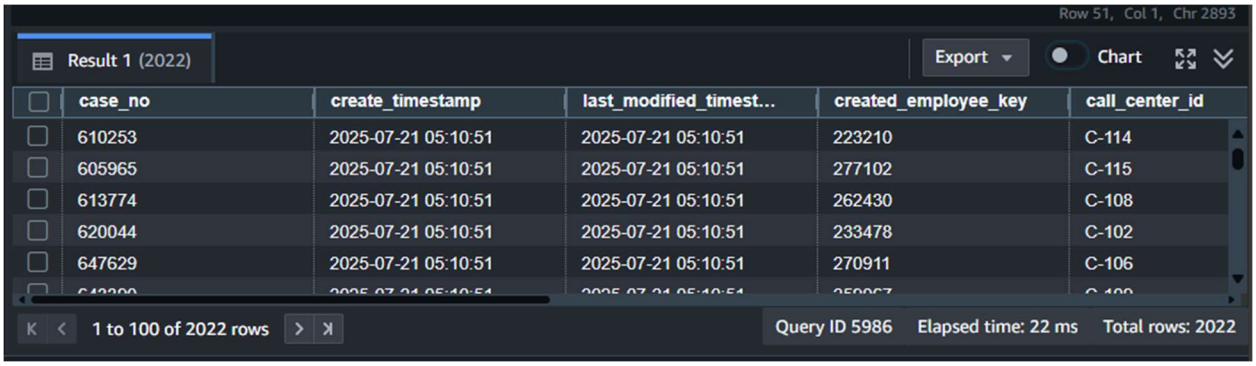
1 to 100 of 2014 rows — Query ID 5672 — Elapsed time: 25 ms — Total rows: 2014

Data after the spark streaming:

Result 1 (1608) — Export — Chart

survey_id	case_no	survey_timestamp	q1	q2
S-514032	696729	2025-07-21 05:10:51	1	4
S-514264	649950	2025-07-21 05:10:51	9	8
S-571629	617917	2025-07-21 05:10:51	5	2
S-560236	682240	2025-07-21 05:10:51	10	4
S-554023	698346	2025-07-21 05:10:51	9	8

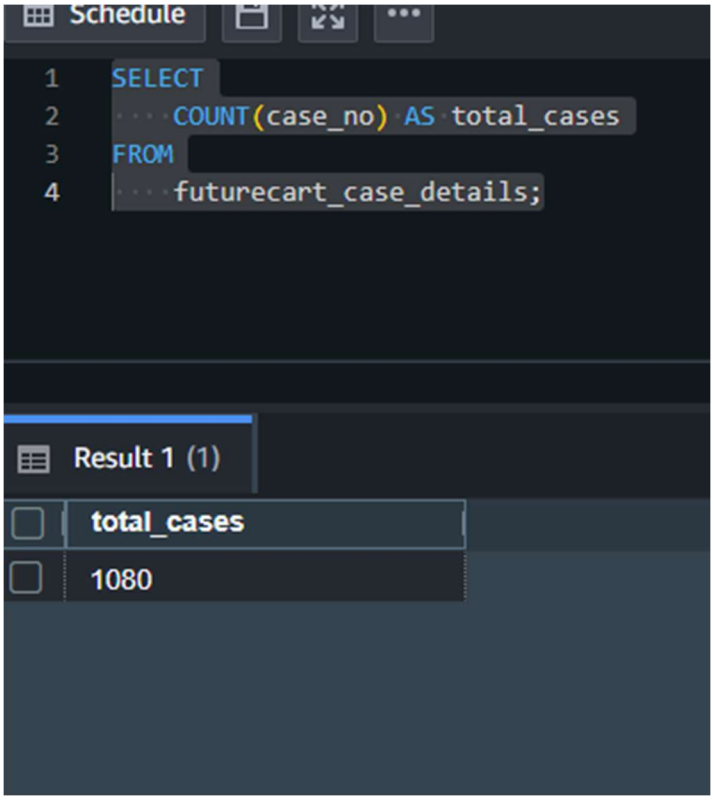
1 to 100 of 1608 rows — Query ID 6041 — Elapsed time: 25 ms — Total rows: 1608



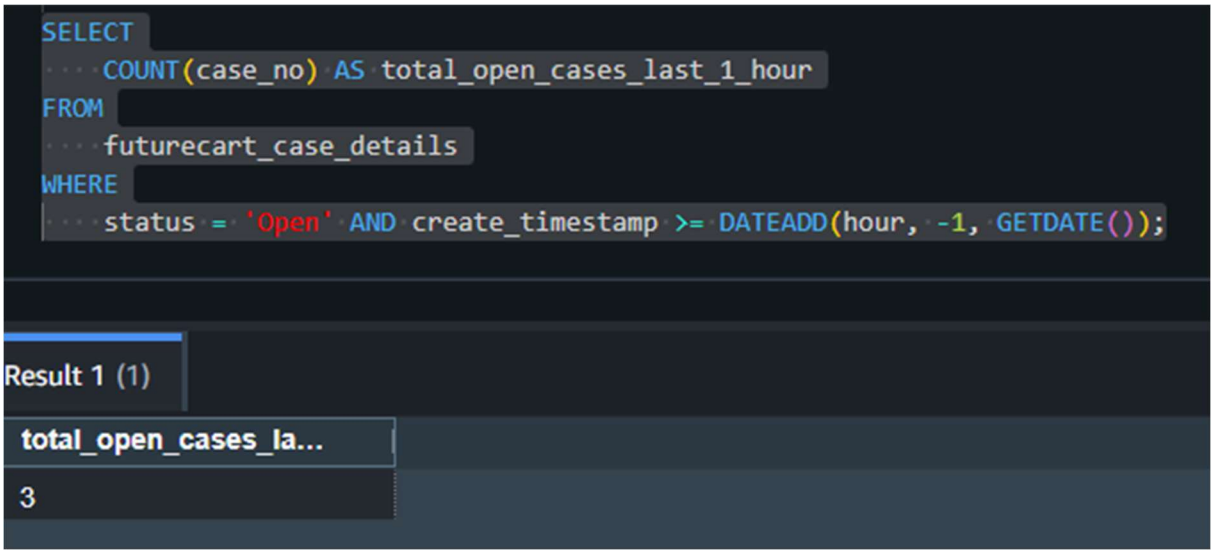
Step 6: Data Analysis and Querying in Redshift

The following queries were executed in Redshift to analyze the data:

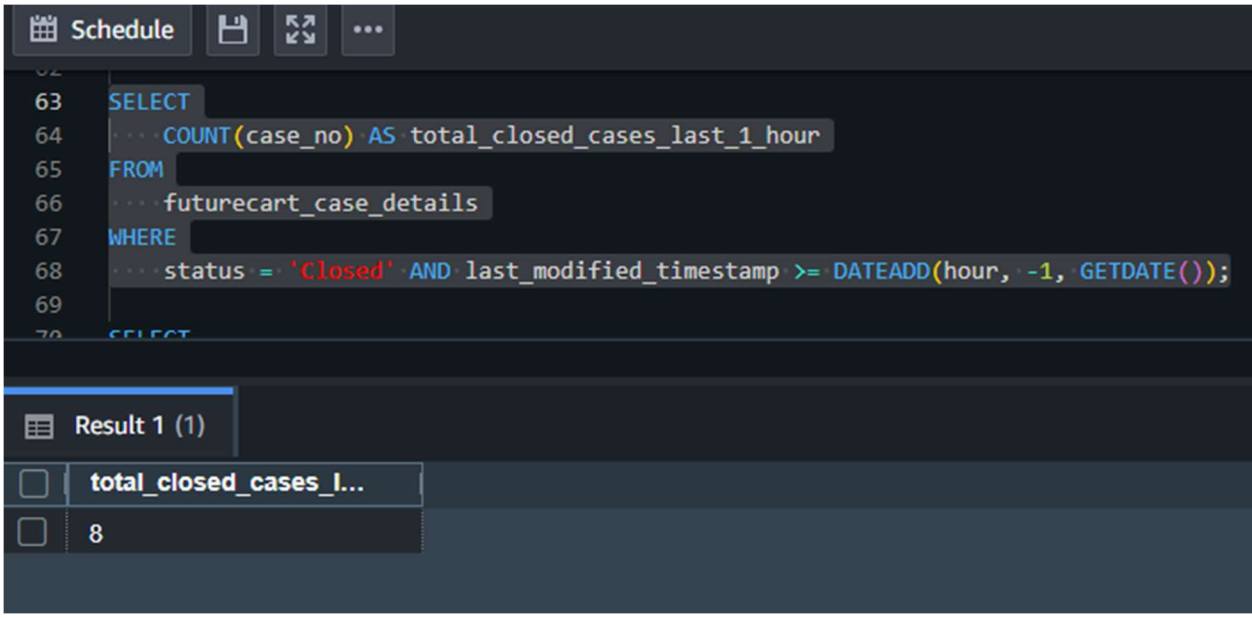
- **Total Cases:**
    - Query: SELECT COUNT(case_no) AS total_cases FROM futurecart_case_details;
    - Result: 1080 cases.



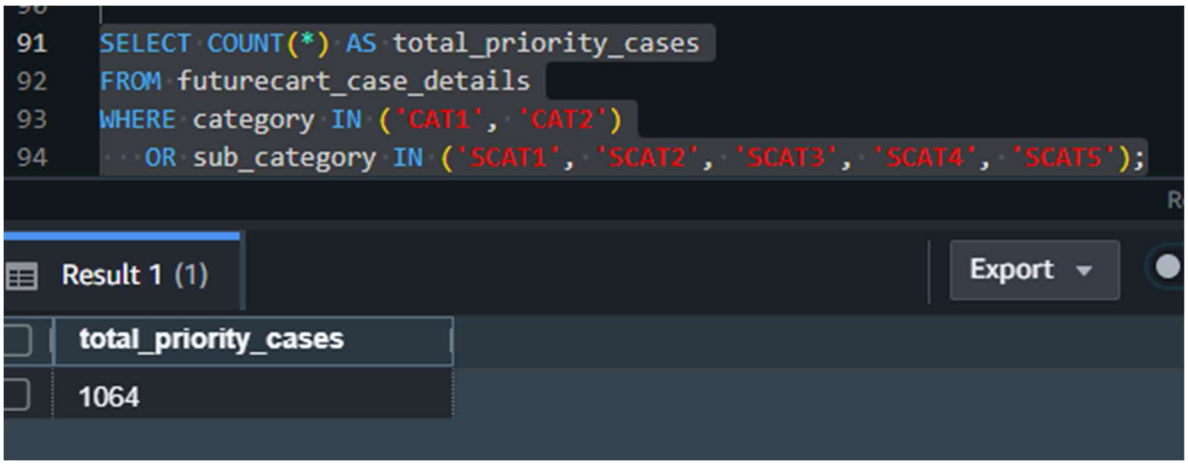
- **Total Open Cases Last 1 Hour:**
    - Query: SELECT COUNT(case_no) AS total_open_cases_last_1_hour FROM futurecart_case_details WHERE status = 'Open' AND create_timestamp >= DATEADD(hour, -1, GETDATE());
    - Result: 3



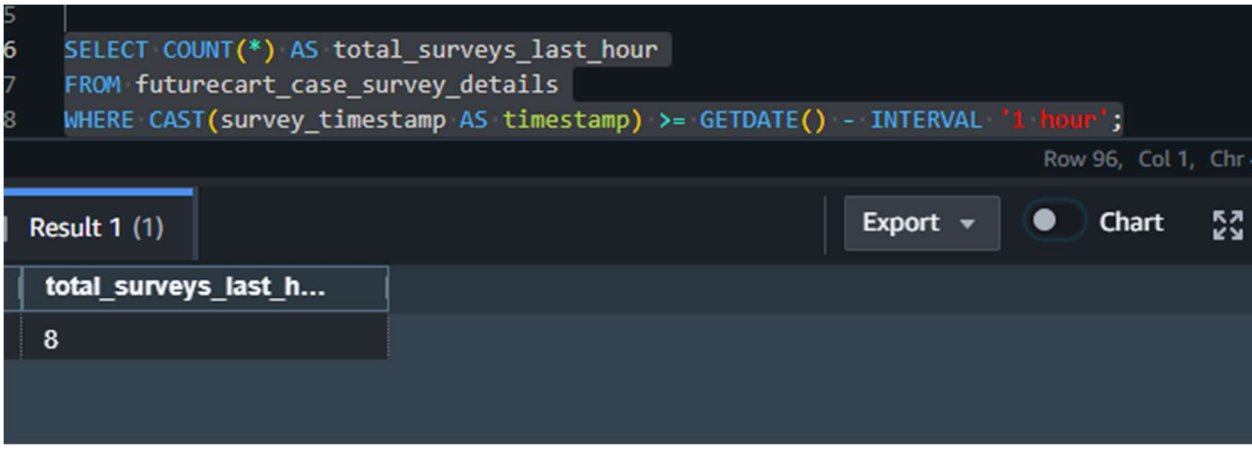
- **Total Closed Cases Last 1 Hour:**
    - Query: SELECT COUNT(case_no) AS total_closed_cases_last_1_hour FROM futurecart_case_details WHERE status = 'Closed' AND last_modified_timestamp >= DATEADD(hour, -1, GETDATE());
    - Result: 8



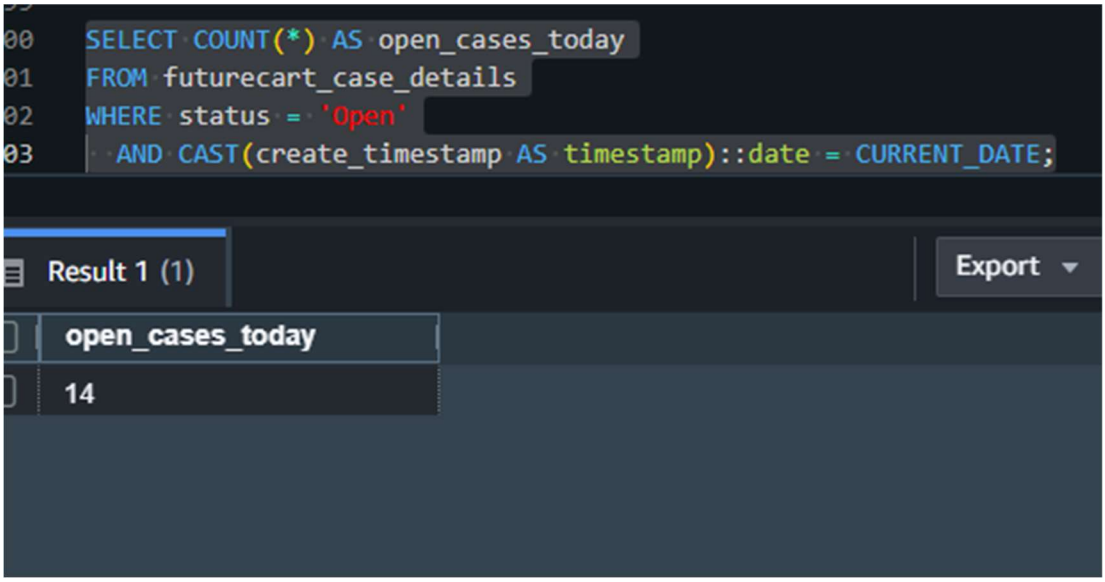
- **Total Priority Cases (CAT1 or CAT2, or specific sub-categories):**
    - Query: SELECT COUNT(*) AS total_priority_cases FROM futurecart_case_details WHERE category IN ('CAT1', 'CAT2') OR sub_category IN ('SCAT1', 'SCAT2', 'SCAT3', 'SCAT4', 'SCAT5');
    - Result: 1064



- **Total Surveys Last Hour:**
    - Query: SELECT COUNT(*) AS total_surveys_last_hour FROM futurecart_case_survey_details WHERE CAST(survey_timestamp AS timestamp) >= GETDATE() - INTERVAL '1 hour';
    - Result: 8

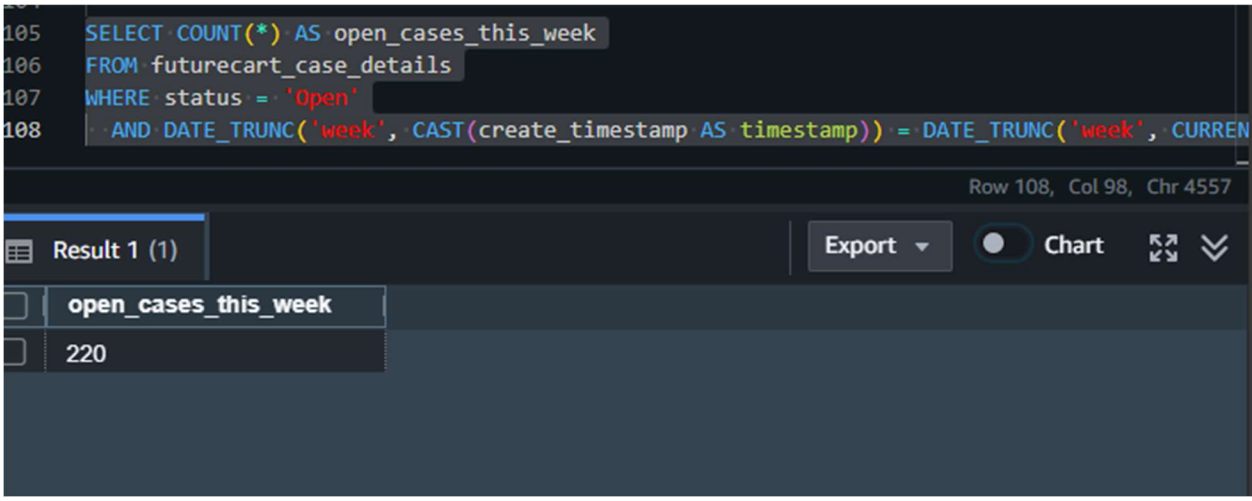


- **Open Cases Today:**
    - Query: SELECT COUNT(*) AS open_cases_today FROM futurecart_case_details WHERE status = 'Open' AND CAST(create_timestamp AS timestamp)::date = CURRENT_DATE;
    - Result: 14

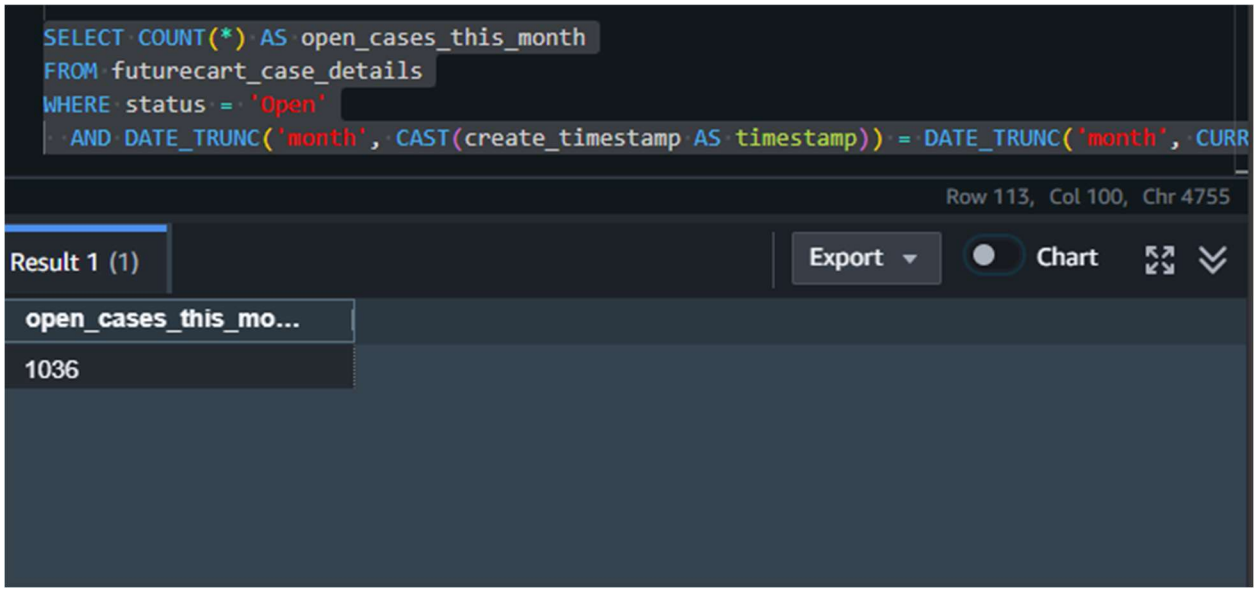


- **Open Cases This Week:**

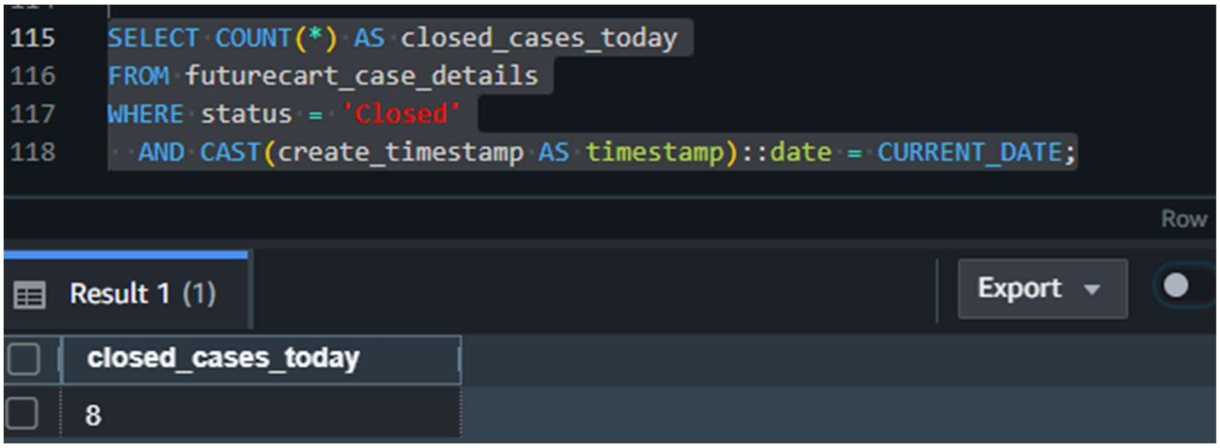
- Query: SELECT COUNT(*) AS open_cases_this_week FROM futurecart_case_details WHERE status = 'Open' AND DATE_TRUNC('week', CAST(create_timestamp AS timestamp)) = DATE_TRUNC('week', CURRENT_DATE);
- Result: 220



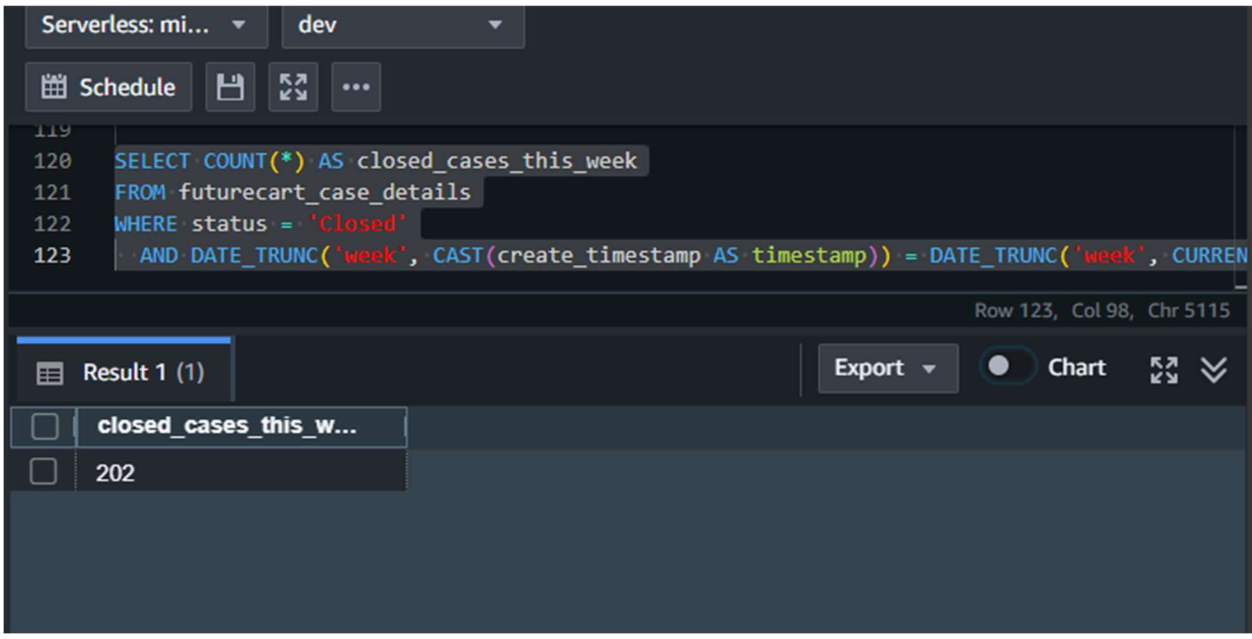
- **Open Cases This Month:**
  - Query: SELECT COUNT(*) AS open_cases_this_month FROM futurecart_case_details WHERE status = 'Open' AND DATE_TRUNC('month', CAST(create_timestamp AS timestamp)) = DATE_TRUNC('month', CURRENT_DATE);
  - Result: 1036



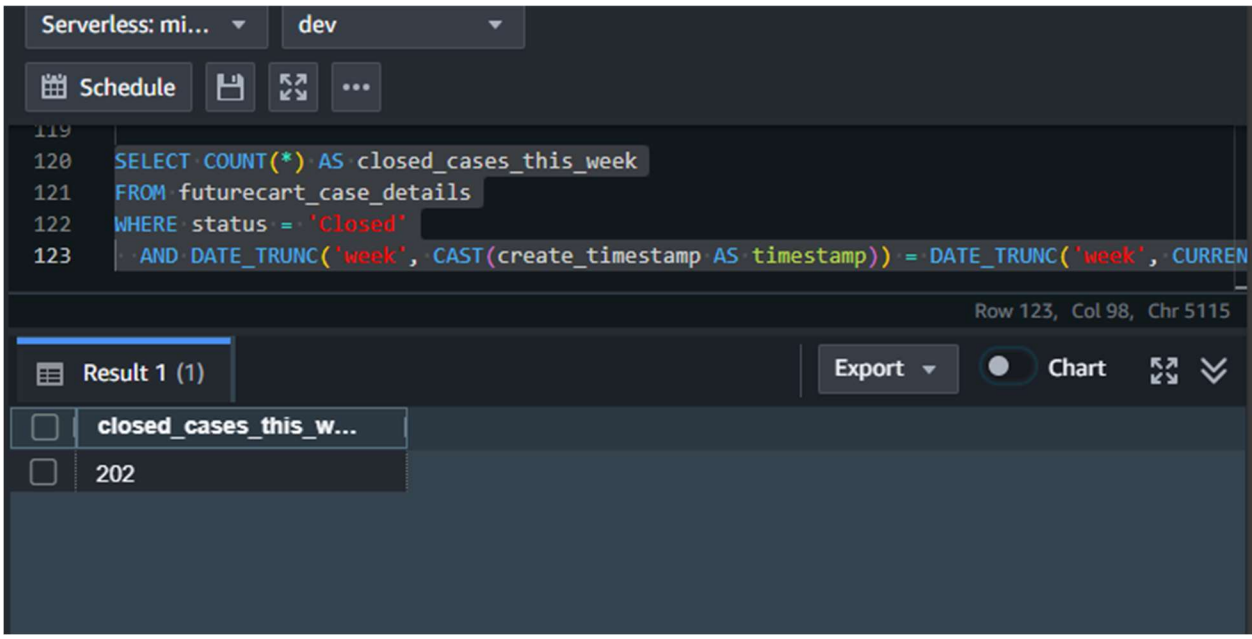
- **Closed Cases Today:**
    - Query: SELECT COUNT(*) AS closed_cases_today FROM futurecart_case_details WHERE status = 'Closed' AND CAST(create_timestamp AS timestamp)::date = CURRENT_DATE;
    - Result: 8



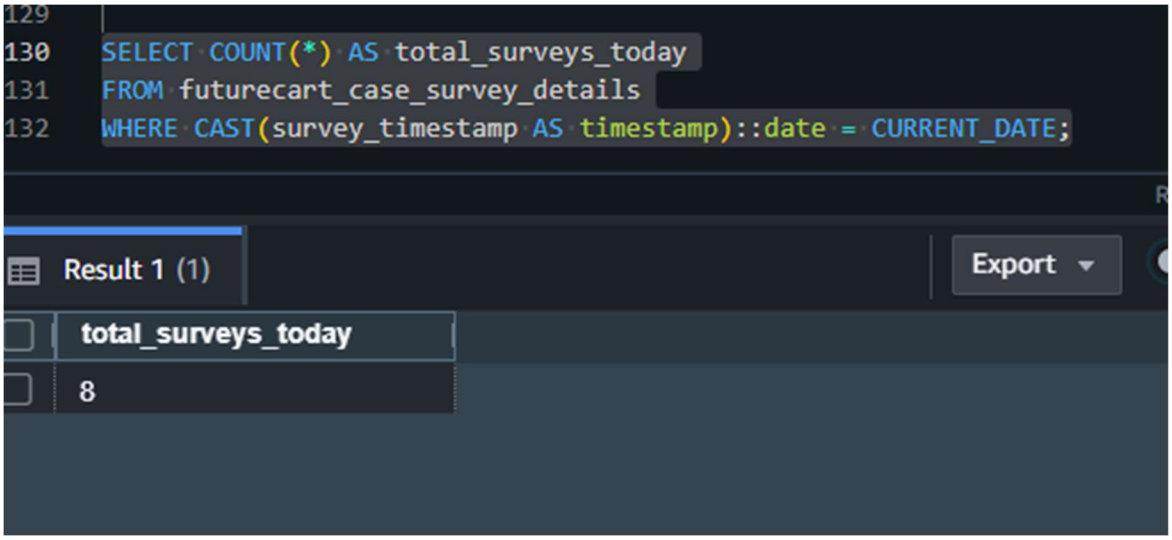
- **Closed Cases This Week:**
    - Query: SELECT COUNT(*) AS closed_cases_this_week FROM futurecart_case_details WHERE status = 'Closed' AND DATE_TRUNC('week', CAST(create_timestamp AS timestamp)) = DATE_TRUNC('week', CURRENT_DATE);
    - Result: 202



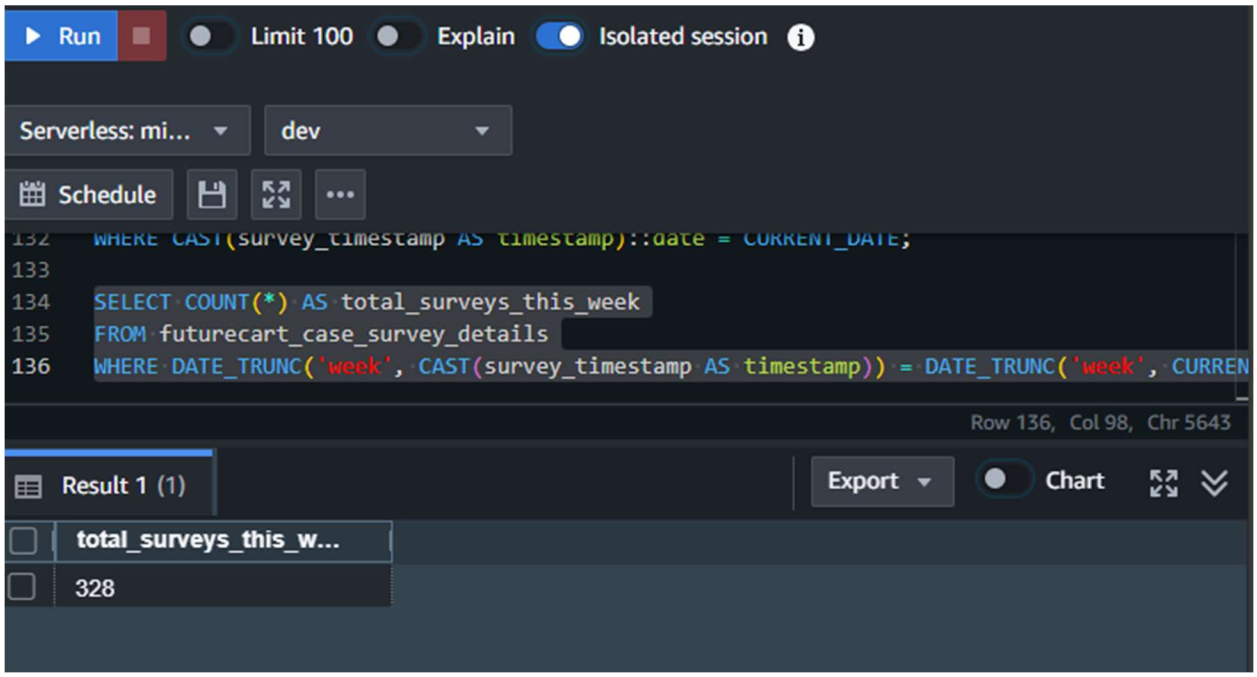
- **Closed Cases This Month:**
    - Query: SELECT COUNT(*) AS closed_cases_this_month FROM futurecart_case_details WHERE status = 'Closed' AND DATE_TRUNC('month', CAST(create_timestamp AS timestamp)) = DATE_TRUNC('month', CURRENT_DATE);
    - Result: 986



- **Total Surveys Today:**
    - Query: SELECT COUNT(*) AS total_surveys_today FROM futurecart_case_survey_details WHERE CAST(survey_timestamp AS timestamp)::date = CURRENT_DATE;
    - Result: 8

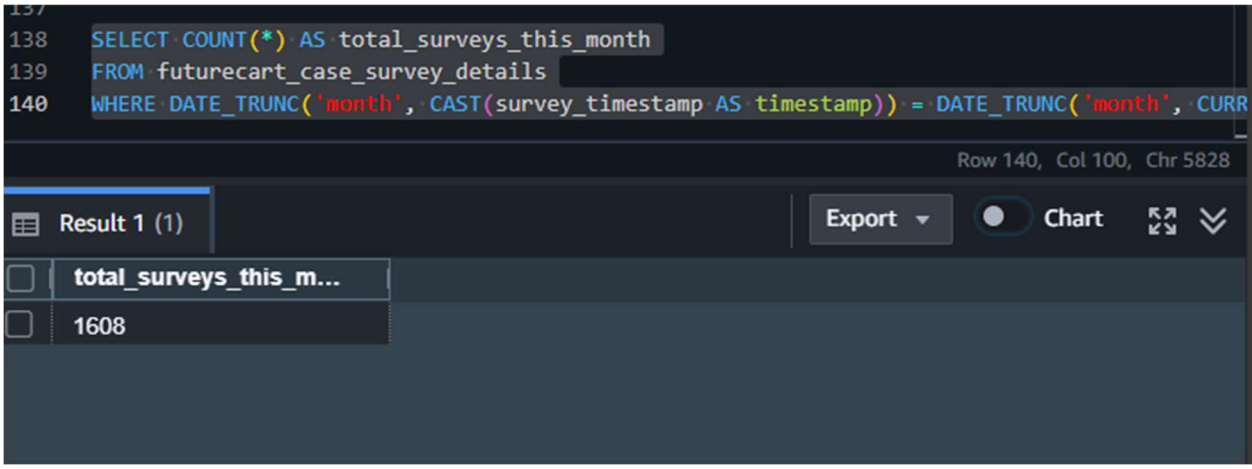


- **Total Surveys This Week:**
    - Query: SELECT COUNT(*) AS total_surveys_this_week FROM futurecart_case_survey_details WHERE DATE_TRUNC('week', CAST(survey_timestamp AS timestamp)) = DATE_TRUNC('week', CURRENT_DATE);
    - Result: 328

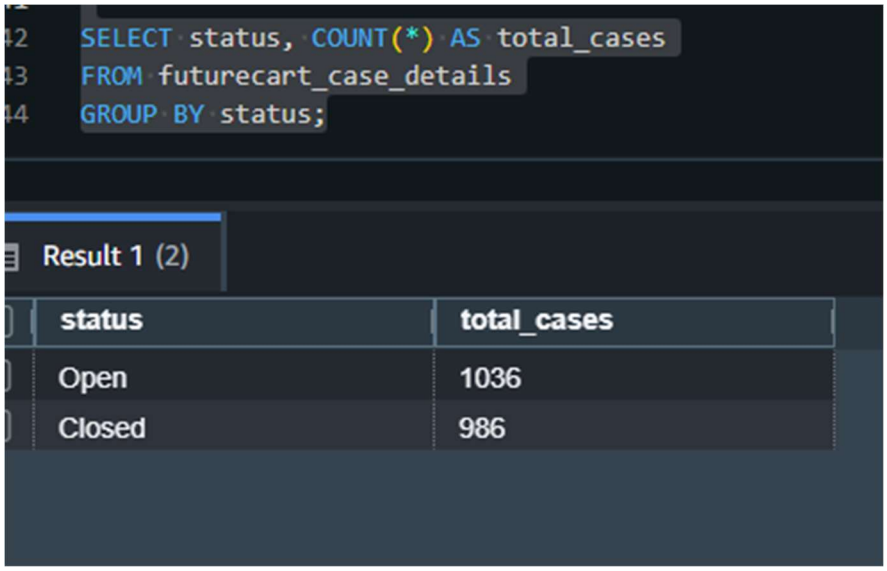


- **Total Surveys This Month:**

- Query: SELECT COUNT(*) AS total_surveys_this_month FROM futurecart_case_survey_details WHERE DATE_TRUNC('month', CAST(survey_timestamp AS timestamp)) = DATE_TRUNC('month', CURRENT_DATE);
- Result: 1608



- **Total Cases by Status:**
  - Query: SELECT status, COUNT(*) AS total_cases FROM futurecart_case_details GROUP BY status;
  - Result: Open: 1036, Closed: 986



- **Cases by Category and Sub-category (Priority and Severity):**
  - Query:

SELECT category AS priority, sub_category AS severity, COUNT(*) AS total_cases FROM futurecart_case_details GROUP BY category, sub_category ORDER BY category, sub_category;