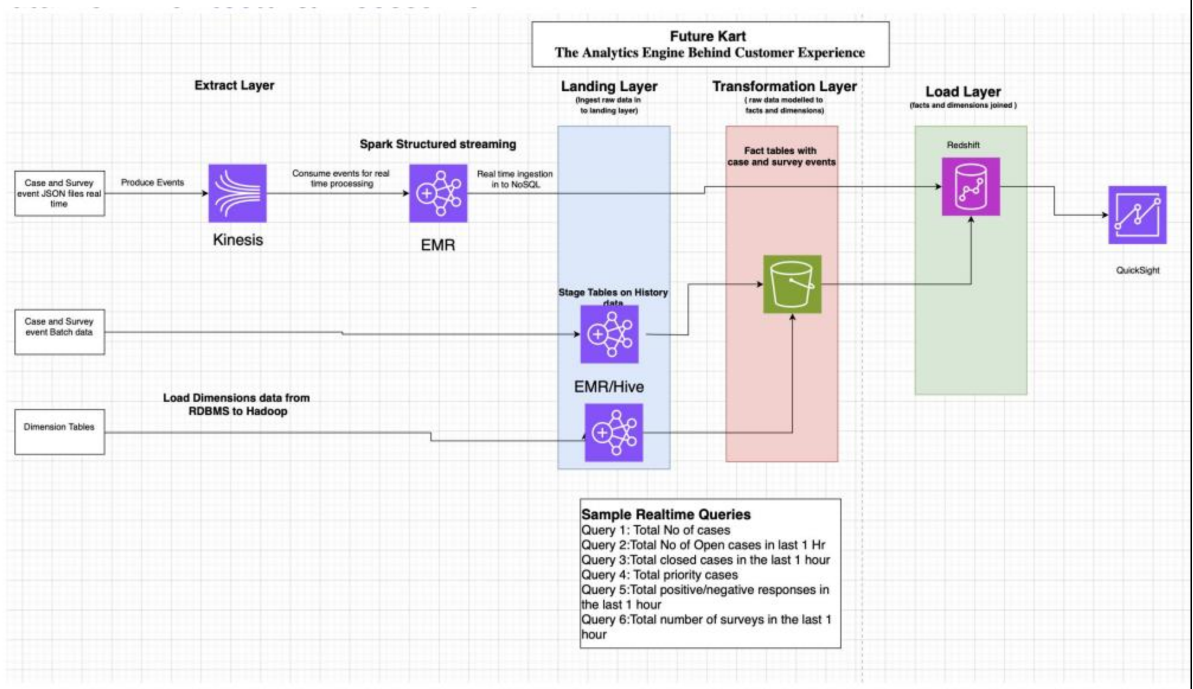
Project Report

Customer Retention Strategy

Priti Ranjan Samal

Contents

Customer Retention Strategy Project – Overview


Future Kart
The Analytics Engine Behind Customer Experience
Extract Layer
Landing Layer
(Ingest raw data in to landing layer)
Transformation Layer
(raw data modelled to facts and dimensions)
Load Layer
(facts and dimensions joined)
Spark Structured streaming
Case and Survey event JSON files real time
Produce Events
Kinesis
Consume events for real time processing
EMR
Real time ingestion in to NoSQL
Fact tables with case and survey events
Redshift
QuickSight
Stage Tables on History data
Case and Survey event Batch data
EMR/Hive
Load Dimensions data from RDBMS to Hadoop
Dimension Tables

Sample Realtime Queries
Query 1: Total No of cases
Query 2:Total No of Open cases in last 1 Hr
Query 3:Total closed cases in the last 1 hour
Query 4: Total priority cases
Query 5:Total positive/negative responses in the last 1 hour
Query 6:Total number of surveys in the last 1 hour

1. Business Challenge

FutureCart Inc., a leading retail chain with 5000+ stores and a strong e-commerce presence in India, aims to strengthen customer retention and satisfaction through improved after-sales service. To achieve this, the company has implemented a **CRM strategy** that leverages **data warehousing** and **real-time analytics**.

A dedicated support team handles customer complaints across multiple **contact centers**, allowing customers to raise issues via **calls, chats, or emails**. Each complaint is registered as a **case**, assigned a **priority**, and tracked with a defined **SLA**. After resolution, customers may complete a **feedback survey**, rating their experience on a scale of 1–10.

The collected data powers both **real-time** and **batch analytics** to optimize the CRM process and improve service quality.

KPIs Tracked

- Total cases (open/closed) by hour/day/week/month
- Priority case volumes
- SLA compliance
- Positive/negative survey responses
- Real-time case and sentiment trends

2. Project Goal

Business Goals

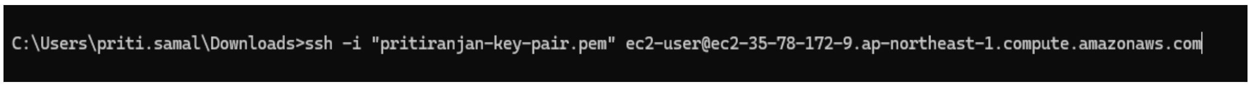
- Improve SLA compliance and customer satisfaction
- Use CRM insights to boost customer loyalty and retention

Technical Goals

- Implement data pipelines for real-time and batch ingestion using **Kinesis**, **Spark**, and **Glue**
- Clean and transform data using **Spark** and **Hive**
- Store processed data in **Hive/DynamoDB/HDFS**
- Build dashboards to monitor key metrics in real time
- Adopt a **Lambda architecture** for hybrid processing

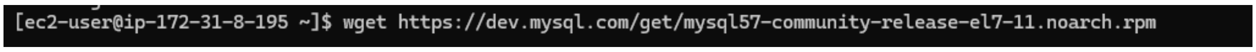
EC2-Instance Setup

- Login to the EC2 Instance from command line using credentials



```
C:\Users\priti.samal\Downloads>ssh -i "pritiranjan-key-pair.pem" ec2-user@ec2-35-78-172-9.ap-northeast-1.compute.amazonaws.com
```

- Once logged in to the instance download SQL into the instance

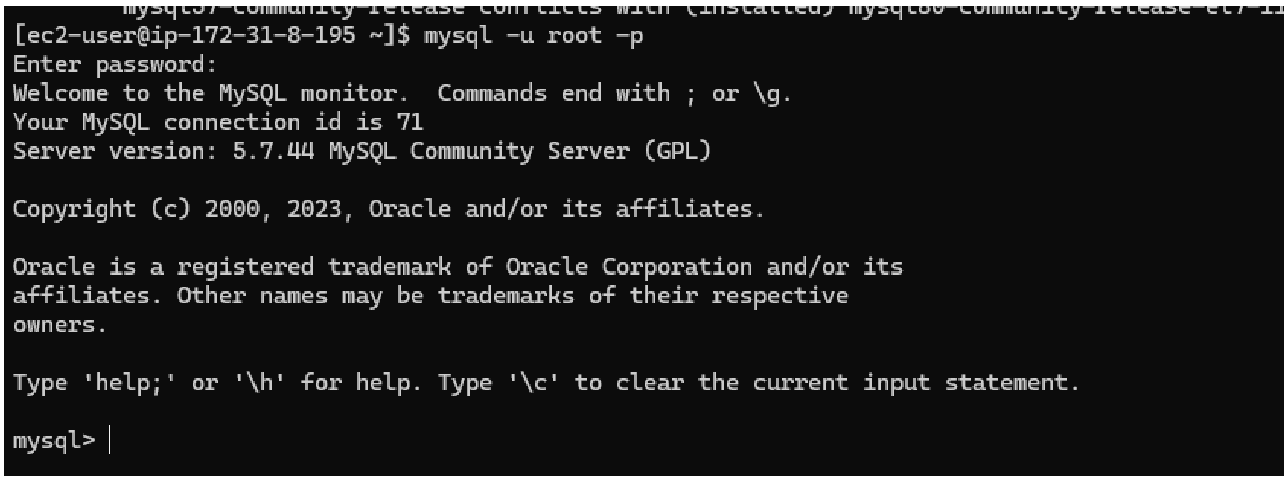


```
[ec2-user@ip-172-31-8-195 ~]$ wget https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm
```

- Once installation is done and mysql service is enabled using the command

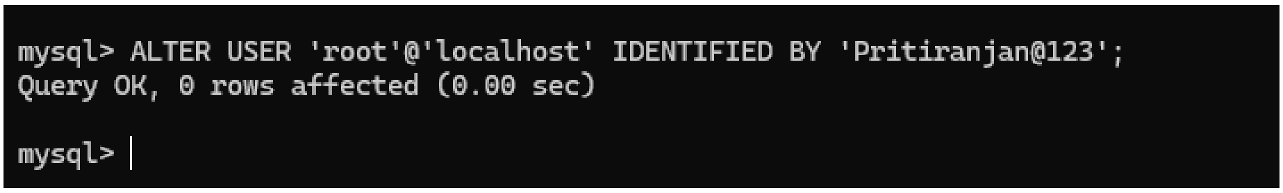
```
Sudo systemctl start mysql
```

- Login to mysql server using the initial temporary password set during installation



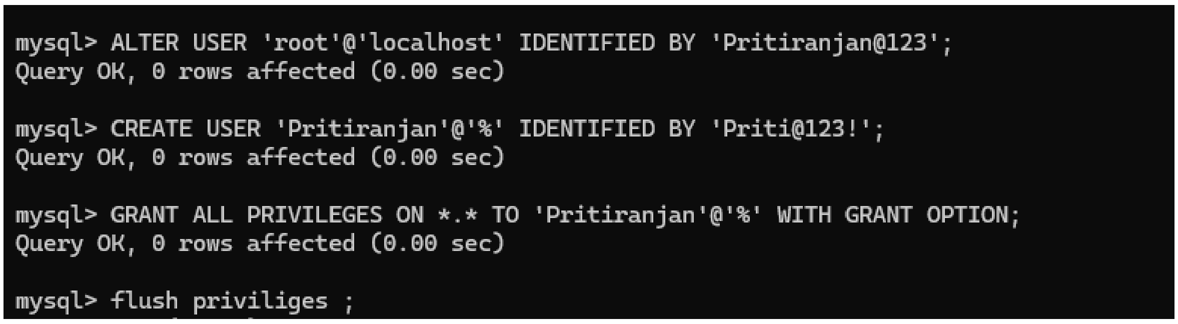
```
   mysql57-community-release conflicts with (installed) mysql80-community-release-el7-1
[ec2-user@ip-172-31-8-195 ~]$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 71
Server version: 5.7.44 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> |
```

- Once logged in change the password for root and set a permanent password



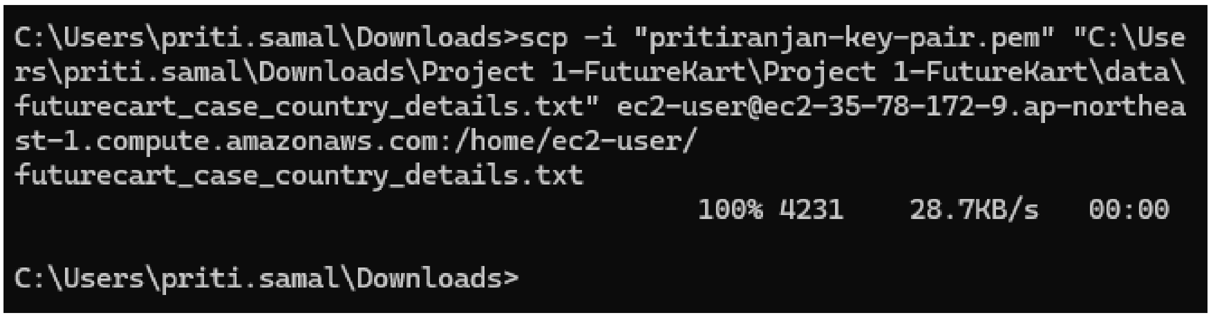
```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'Pritiranjan@123';
Query OK, 0 rows affected (0.00 sec)

mysql> |
```

- Now create a new user with all the privileges to access the database from any desired IP address



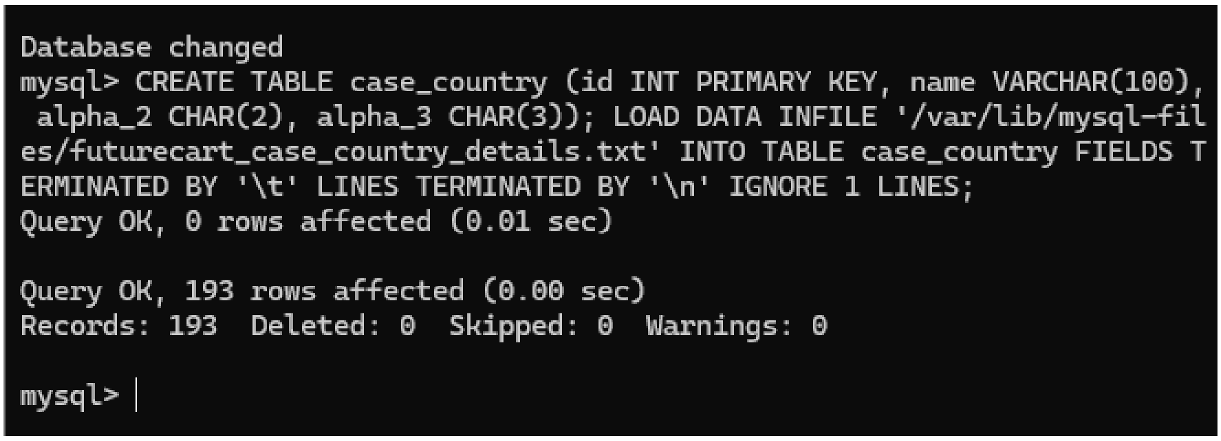
```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'Pritiranjan@123';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE USER 'Pritiranjan'@'%' IDENTIFIED BY 'Priti@123!';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON *.* TO 'Pritiranjan'@'%' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)

mysql> flush priviliges ;
```

- Importing the data from local machine to EC2 Instance
- Now move all the datasets provided , into the EC2 instance with the help of which we will be creating the tables



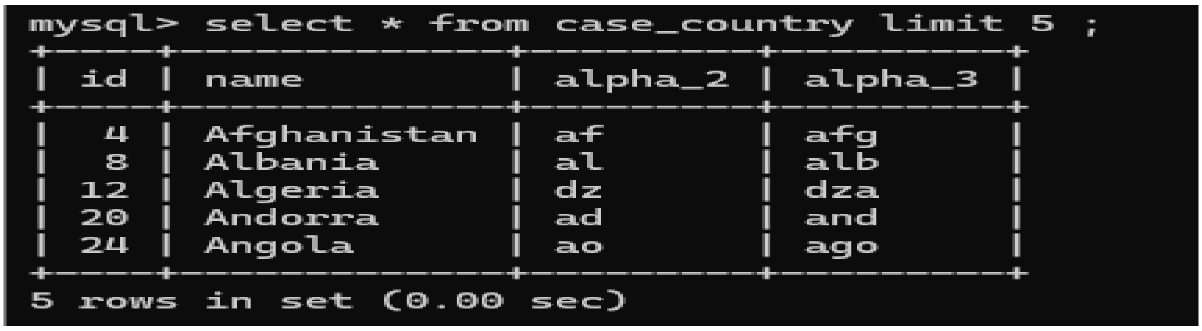
```
C:\Users\priti.samal\Downloads>scp -i "pritiranjan-key-pair.pem" "C:\Use
rs\priti.samal\Downloads\Project 1-FutureKart\Project 1-FutureKart\data\
futurecart_case_country_details.txt" ec2-user@ec2-35-78-172-9.ap-northea
st-1.compute.amazonaws.com:/home/ec2-user/
futurecart_case_country_details.txt
                                            100% 4231     28.7KB/s   00:00

C:\Users\priti.samal\Downloads>
```

- Now once the datasets are moved inside the EC2 Instance we have to move it to **'/var/lib/mysql-files/'** because mysql can access files from that directory
- Once moved start creating table and load the data into the table from the dataset in **'/var/lib/mysql-files/'**



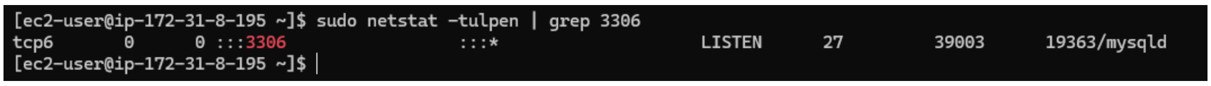
```
Database changed
mysql> CREATE TABLE case_country (id INT PRIMARY KEY, name VARCHAR(100),
 alpha_2 CHAR(2), alpha_3 CHAR(3)); LOAD DATA INFILE '/var/lib/mysql-fil
es/futurecart_case_country_details.txt' INTO TABLE case_country FIELDS T
ERMINATED BY '\t' LINES TERMINATED BY '\n' IGNORE 1 LINES;
Query OK, 0 rows affected (0.01 sec)

Query OK, 193 rows affected (0.00 sec)
Records: 193  Deleted: 0  Skipped: 0  Warnings: 0

mysql> 
```

- Check if table has been created and data has been inserted



```
mysql> select * from case_country limit 5 ;
+----+-------------+---------+---------+
| id | name        | alpha_2 | alpha_3 |
+----+-------------+---------+---------+
|  4 | Afghanistan | af      | afg     |
|  8 | Albania     | al      | alb     |
| 12 | Algeria     | dz      | dza     |
| 20 | Andorra     | ad      | and     |
| 24 | Angola      | ao      | ago     |
+----+-------------+---------+---------+
5 rows in set (0.00 sec)
```

- Check if the database and the tables can be accessed from anywhere from any IP address



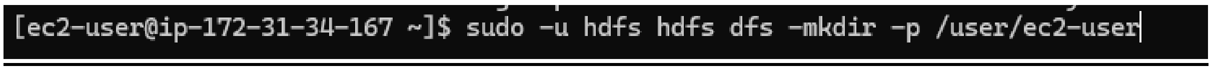
```
[ec2-user@ip-172-31-8-195 ~]$ sudo netstat -tulpen | grep 3306
tcp6       0      0 :::3306                 :::*                    LISTEN      27         39003      19363/mysqld
[ec2-user@ip-172-31-8-195 ~]$ |
```

- As it can be assessed from any point so now we can set up the EMR Cluster

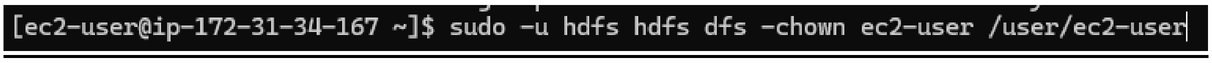
EMR Cluster Setup

- As all the privileges to the user has been granted from the EC2 instance to access the data from the database so we will be using Sqoop in order to fetch data from the database and put in HDFS
- First to setup HDFS run the commands :

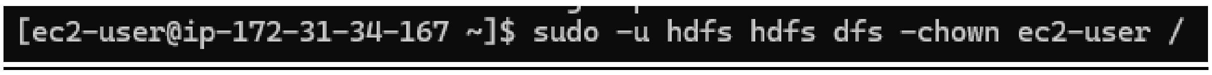
Sudo -u hdfs hdfs dfs -mkdir -p /user/ec2-user
Sudo -u hdfs hdfs dfs -chown ec2-user /user/ec2-user
Sudo -u hdfs hdfs dfs -chown ec2-user /



```
[ec2-user@ip-172-31-34-167 ~]$ sudo -u hdfs hdfs dfs -mkdir -p /user/ec2-user
```

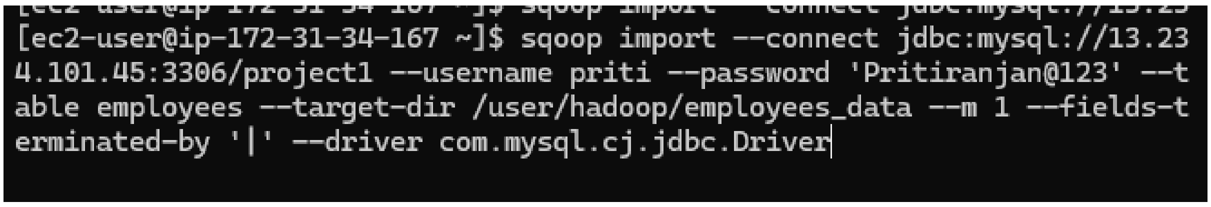


```
[ec2-user@ip-172-31-34-167 ~]$ sudo -u hdfs hdfs dfs -chown ec2-user /user/ec2-user
```



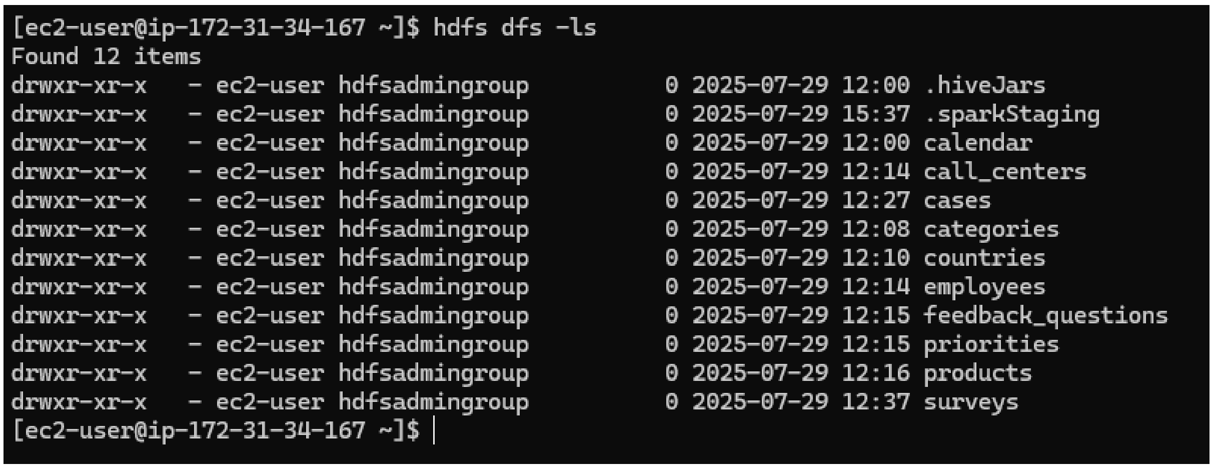
```
[ec2-user@ip-172-31-34-167 ~]$ sudo -u hdfs hdfs dfs -chown ec2-user /
```

- Once the **'/user/ec2-user '** directory is created and the ec2-user has accessibility
- Now start moving the data from SQL in EC2-Instance to HDFS system in EMR Cluster

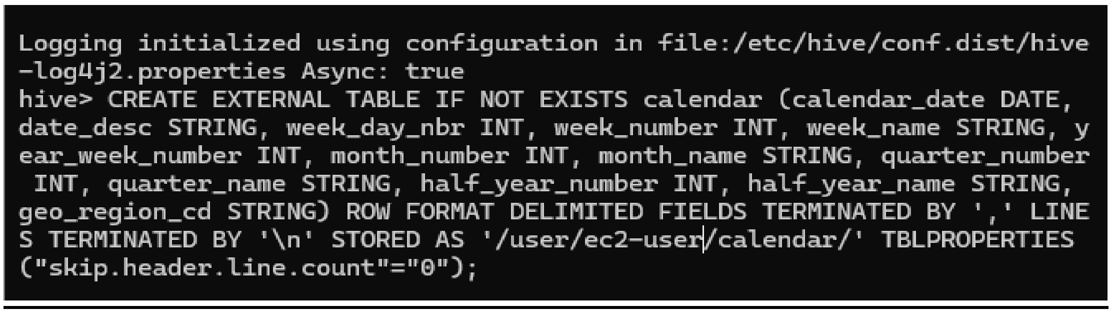


```
[ec2-user@ip-172-31-34-167 ~]$ sqoop import --connect jdbc:mysql://13.23
4.101.45:3306/project1 --username priti --password 'Pritiranjan@123' --t
able employees --target-dir /user/hadoop/employees_data --m 1 --fields-t
erminated-by '|' --driver com.mysql.cj.jdbc.Driver
```

- Once all the data is moved to HDFS start creating table in Hive and loading data from HDFS

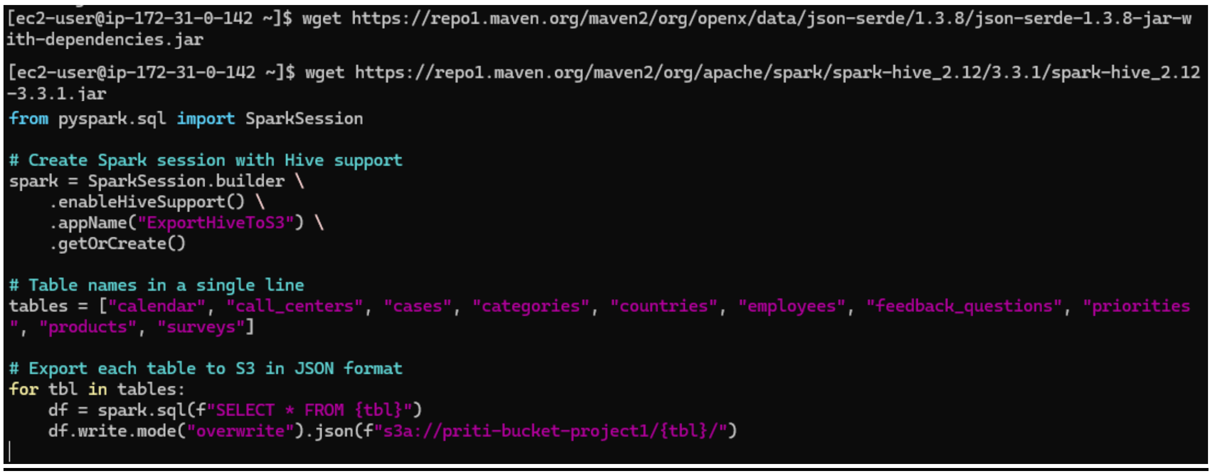


```
[ec2-user@ip-172-31-34-167 ~]$ hdfs dfs -ls
Found 12 items
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:00 .hiveJars
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 15:37 .sparkStaging
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:00 calendar
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:14 call_centers
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:27 cases
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:08 categories
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:10 countries
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:14 employees
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:15 feedback_questions
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:15 priorities
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:16 products
drwxr-xr-x   - ec2-user hdfsadmingroup          0 2025-07-29 12:37 surveys
[ec2-user@ip-172-31-34-167 ~]$
```



```
Logging initialized using configuration in file:/etc/hive/conf.dist/hive
-log4j2.properties Async: true
hive> CREATE EXTERNAL TABLE IF NOT EXISTS calendar (calendar_date DATE,
date_desc STRING, week_day_nbr INT, week_number INT, week_name STRING, y
ear_week_number INT, month_number INT, month_name STRING, quarter_number
 INT, quarter_name STRING, half_year_number INT, half_year_name STRING,
geo_region_cd STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINE
S TERMINATED BY '\n' STORED AS '/user/ec2-user/calendar/' TBLPROPERTIES
("skip.header.line.count"="0");
```

- Once all the tables are created then upload it to S3-Bucket in JSON Format using spark job .
- While uploading data to S3 in JSON format, I got an error saying the data wasn't properly formatted. So, I downloaded a newer version of the SerDe library, updated my Spark job to use it, and then uploaded the data again successfully.

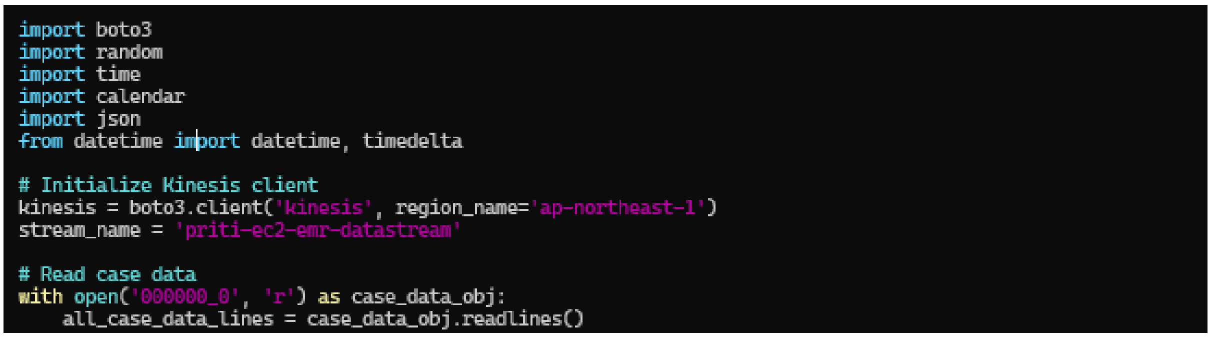


```
[ec2-user@ip-172-31-0-142 ~]$ wget https://repo1.maven.org/maven2/org/openx/data/json-serde/1.3.8/json-serde-1.3.8-jar-w
ith-dependencies.jar

[ec2-user@ip-172-31-0-142 ~]$ wget https://repo1.maven.org/maven2/org/apache/spark/spark-hive_2.12/3.3.1/spark-hive_2.12
-3.3.1.jar
from pyspark.sql import SparkSession

# Create Spark session with Hive support
spark = SparkSession.builder \
    .enableHiveSupport() \
    .appName("ExportHiveToS3") \
    .getOrCreate()

# Table names in a single line
tables = ["calendar", "call_centers", "cases", "categories", "countries", "employees", "feedback_questions", "priorities
", "products", "surveys"]

# Export each table to S3 in JSON format
for tbl in tables:
    df = spark.sql(f"SELECT * FROM {tbl}")
    df.write.mode("overwrite").json(f"s3a://priti-bucket-project1/{tbl}/")
```

- Now create a Python script to generate historical data of cases and surveys

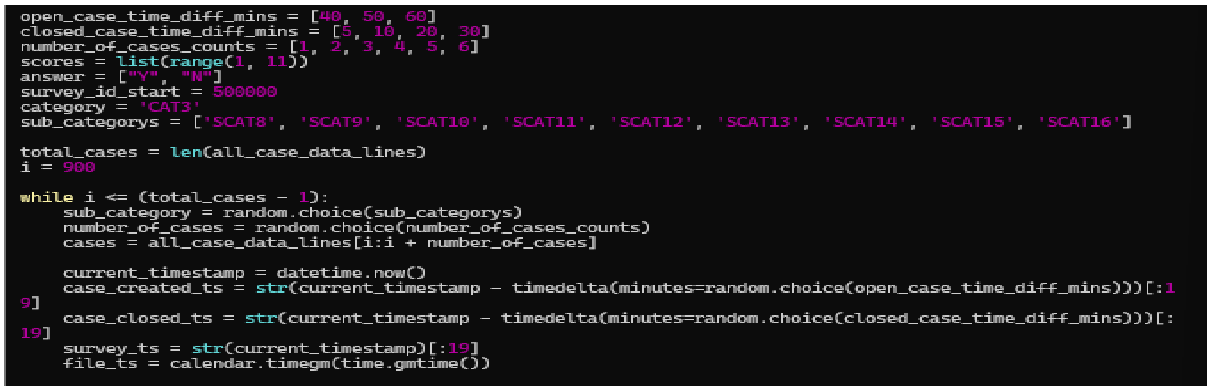
Live Data Streaming From EC2-EMR Cluster

- Now to work with live data or streaming data we need to create a python script in some other EC2-Instance which would generate some sample continuous streaming data
- We would be sending that data with the help of Kinesis DataStream and be receiving it on the EMR cluster using a spark job
- So in order to create the live streaming data we would create and run the script



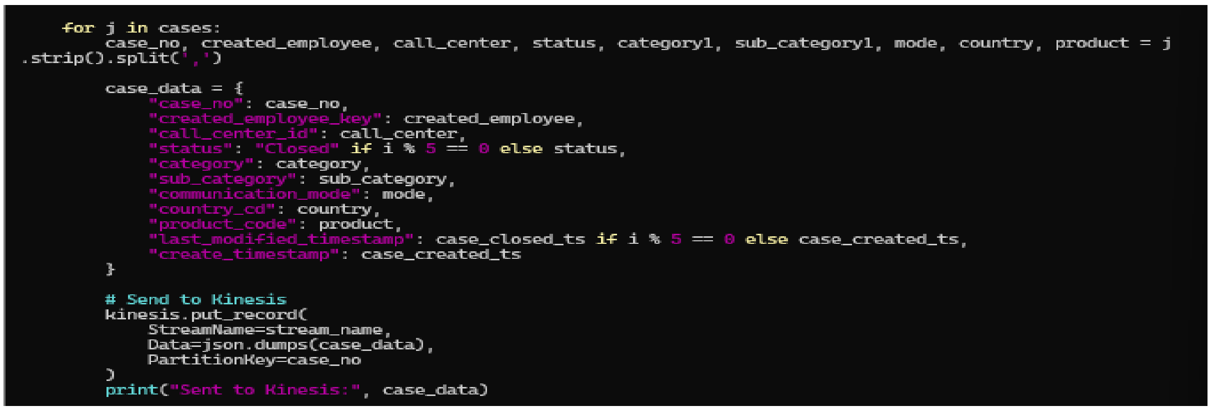
```
import boto3
import random
import time
import calendar
import json
from datetime import datetime, timedelta

# Initialize Kinesis client
kinesis = boto3.client('kinesis', region_name='ap-northeast-1')
stream_name = 'priti-ec2-emr-datastream'

# Read case data
with open('000000_0', 'r') as case_data_obj:
    all_case_data_lines = case_data_obj.readlines()
```

- Opens the file named 000000_0 . Reads all lines into a list called all_case_data_lines



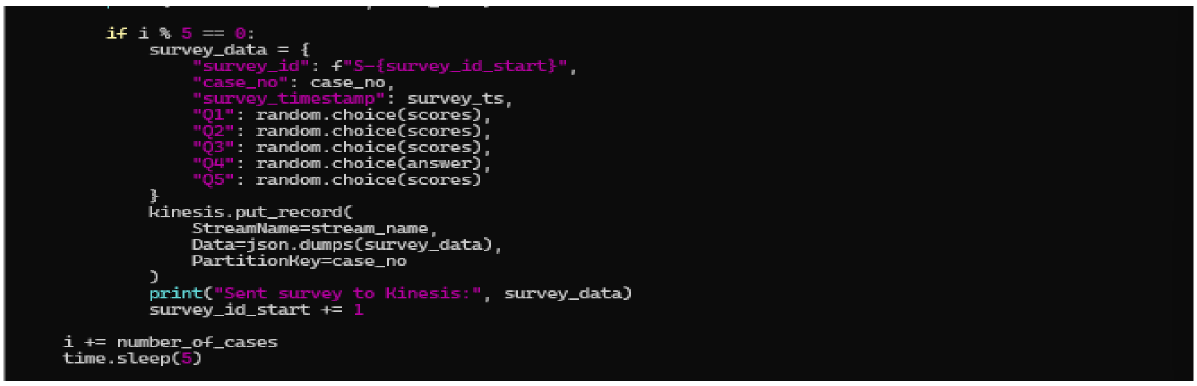
```
open_case_time_diff_mins = [40, 50, 60]
closed_case_time_diff_mins = [5, 10, 20, 30]
number_of_cases_counts = [1, 2, 3, 4, 5, 6]
scores = list(range(1, 11))
answer = ["Y", "N"]
survey_id_start = 500000
category = 'CAT3'
sub_categorys = ['SCAT8', 'SCAT9', 'SCAT10', 'SCAT11', 'SCAT12', 'SCAT13', 'SCAT14', 'SCAT15', 'SCAT16']

total_cases = len(all_case_data_lines)
i = 900

while i <= (total_cases - 1):
    sub_category = random.choice(sub_categorys)
    number_of_cases = random.choice(number_of_cases_counts)
    cases = all_case_data_lines[i:i + number_of_cases]

    current_timestamp = datetime.now()
    case_created_ts = str(current_timestamp - timedelta(minutes=random.choice(open_case_time_diff_mins)))[:1
9]
    case_closed_ts = str(current_timestamp - timedelta(minutes=random.choice(closed_case_time_diff_mins)))[:
19]
    survey_ts = str(current_timestamp)[:19]
    file_ts = calendar.timegm(time.gmtime())
```

- This code block simulates **survey case records** by adding random timestamps and details to existing case data



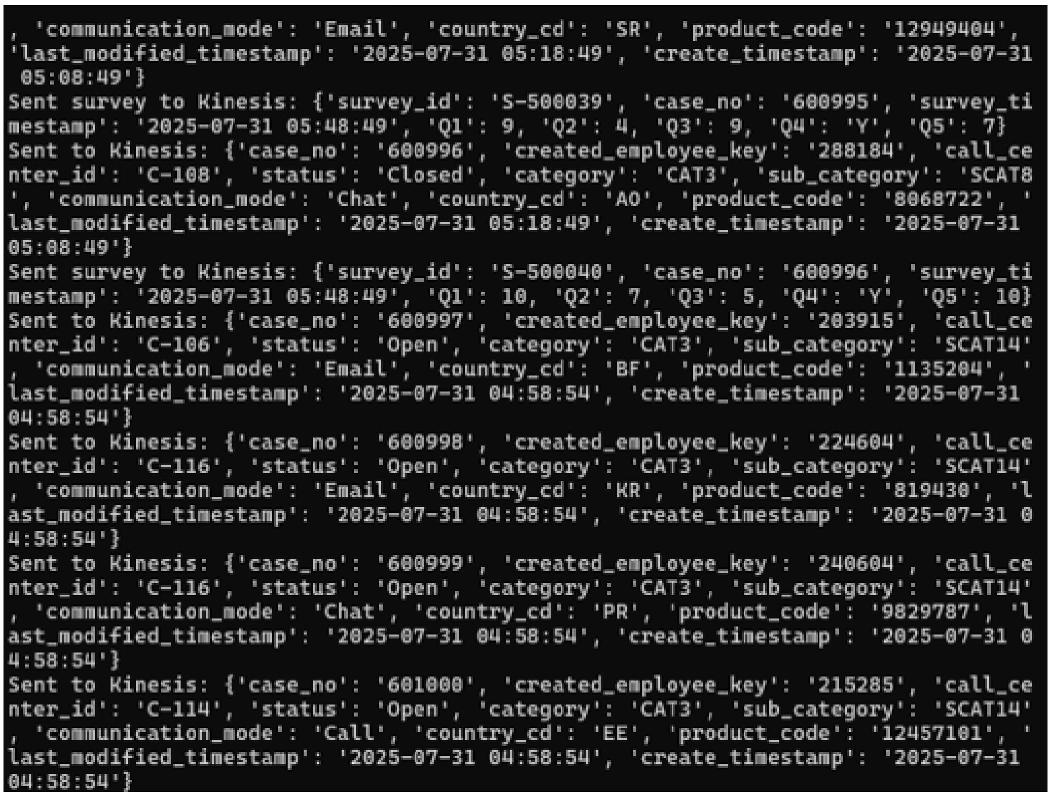
```
    for j in cases:
        case_no, created_employee, call_center, status, category1, sub_category1, mode, country, product = j
.strip().split(',')

        case_data = {
            "case_no": case_no,
            "created_employee_key": created_employee,
            "call_center_id": call_center,
            "status": "Closed" if i % 5 == 0 else status,
            "category": category,
            "sub_category": sub_category,
            "communication_mode": mode,
            "country_cd": country,
            "product_code": product,
            "last_modified_timestamp": case_closed_ts if i % 5 == 0 else case_created_ts,
            "create_timestamp": case_created_ts
        }

        # Send to Kinesis
        kinesis.put_record(
            StreamName=stream_name,
            Data=json.dumps(case_data),
            PartitionKey=case_no
        )
        print("Sent to Kinesis:", case_data)
```

- This block processes each individual case and **sends it to an AWS Kinesis data stream** in JSON format.



```
        if i % 5 == 0:
            survey_data = {
                "survey_id": f"S-{survey_id_start}",
                "case_no": case_no,
                "survey_timestamp": survey_ts,
                "Q1": random.choice(scores),
                "Q2": random.choice(scores),
                "Q3": random.choice(scores),
                "Q4": random.choice(answer),
                "Q5": random.choice(scores)
            }
            kinesis.put_record(
                StreamName=stream_name,
                Data=json.dumps(survey_data),
                PartitionKey=case_no
            )
            print("Sent survey to Kinesis:", survey_data)
            survey_id_start += 1

    i += number_of_cases
    time.sleep(5)
```

- This final part of code handles **sending survey responses** to Kinesis, but only for every 5th batch of cases.



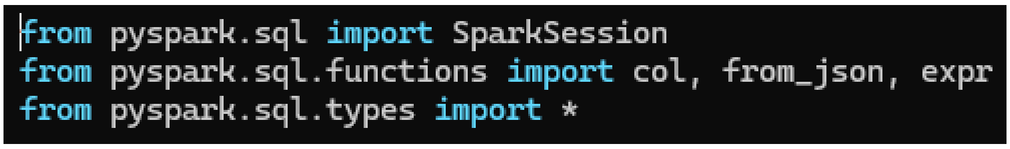
```
, 'communication_mode': 'Email', 'country_cd': 'SR', 'product_code': '12949404',
'last_modified_timestamp': '2025-07-31 05:18:49', 'create_timestamp': '2025-07-31
 05:08:49'}
Sent survey to Kinesis: {'survey_id': 'S-500039', 'case_no': '600995', 'survey_ti
mestamp': '2025-07-31 05:48:49', 'Q1': 9, 'Q2': 4, 'Q3': 9, 'Q4': 'Y', 'Q5': 7}
Sent to Kinesis: {'case_no': '600996', 'created_employee_key': '288184', 'call_ce
nter_id': 'C-108', 'status': 'Closed', 'category': 'CAT3', 'sub_category': 'SCAT8
', 'communication_mode': 'Chat', 'country_cd': 'AO', 'product_code': '8068722', '
last_modified_timestamp': '2025-07-31 05:18:49', 'create_timestamp': '2025-07-31
05:08:49'}
Sent survey to Kinesis: {'survey_id': 'S-500040', 'case_no': '600996', 'survey_ti
mestamp': '2025-07-31 05:48:49', 'Q1': 10, 'Q2': 7, 'Q3': 5, 'Q4': 'Y', 'Q5': 10}
Sent to Kinesis: {'case_no': '600997', 'created_employee_key': '203915', 'call_ce
nter_id': 'C-106', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT14'
, 'communication_mode': 'Email', 'country_cd': 'BF', 'product_code': '1135204', '
last_modified_timestamp': '2025-07-31 04:58:54', 'create_timestamp': '2025-07-31
04:58:54'}
Sent to Kinesis: {'case_no': '600998', 'created_employee_key': '224604', 'call_ce
nter_id': 'C-116', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT14'
, 'communication_mode': 'Email', 'country_cd': 'KR', 'product_code': '819430', 'l
ast_modified_timestamp': '2025-07-31 04:58:54', 'create_timestamp': '2025-07-31 0
4:58:54'}
Sent to Kinesis: {'case_no': '600999', 'created_employee_key': '240604', 'call_ce
nter_id': 'C-116', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT14'
, 'communication_mode': 'Chat', 'country_cd': 'PR', 'product_code': '9829787', 'l
ast_modified_timestamp': '2025-07-31 04:58:54', 'create_timestamp': '2025-07-31 0
4:58:54'}
Sent to Kinesis: {'case_no': '601000', 'created_employee_key': '215285', 'call_ce
nter_id': 'C-114', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT14'
, 'communication_mode': 'Call', 'country_cd': 'EE', 'product_code': '12457101', '
last_modified_timestamp': '2025-07-31 04:58:54', 'create_timestamp': '2025-07-31
04:58:54'}
```

Receiving Live Streaming Data in EMR :

- Once the data start live streaming we have to fetch it into the EMR using a spark job
- We will extract the data from the kinesis data using boto3 client and send it over to Redshift using JDBC connector
- So the data flow is something like :

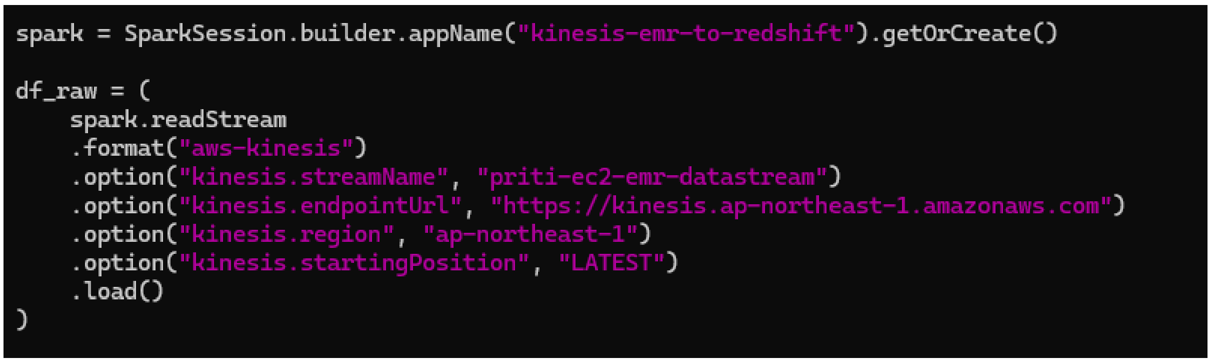
```
EC2 Instance --> Live data sent over data stream --> EMR --> Redshift
```

- We have already started generating the live data using the above generated script now we will be collecting it from the EMR Cluster
- In order to collect data from Kinesis datastream in EMR Cluster and send it to redshift we will be using a python script



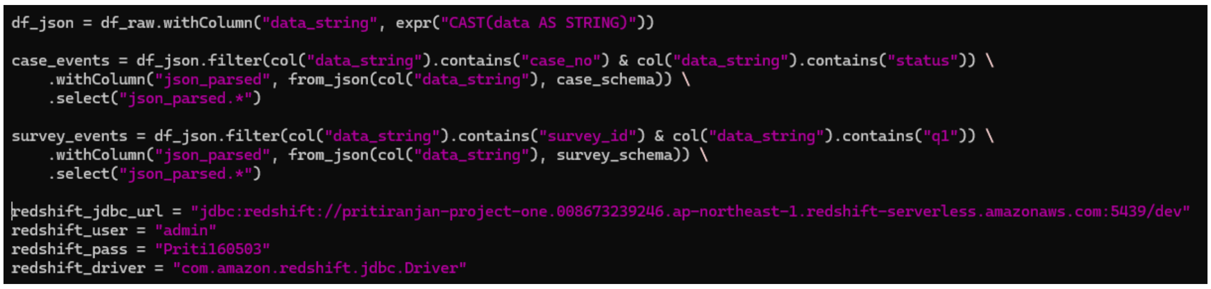
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, from_json, expr
from pyspark.sql.types import *
```

- SparkSession lets you create and manage a Spark application. col, from_json, and expr are functions used to reference columns, parse JSON strings, and run SQL-like expressions on data .
- Create the structure of cases and surveys because we are using json.dumps to extract the data from incoming json records in order to correctly map it respective columns we would be using it .
- It reads real-time data from an AWS **Kinesis Data Stream** (priti-ec2-emr-datastream) in the **ap-northeast-1** region, starting from the **latest** records.



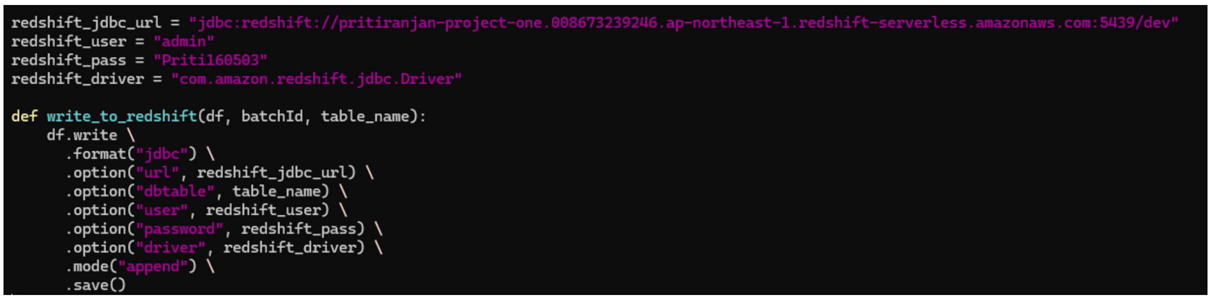
```python
spark = SparkSession.builder.appName("kinesis-emr-to-redshift").getOrCreate()

df_raw = (
    spark.readStream
    .format("aws-kinesis")
    .option("kinesis.streamName", "priti-ec2-emr-datastream")
    .option("kinesis.endpointUrl", "https://kinesis.ap-northeast-1.amazonaws.com")
    .option("kinesis.region", "ap-northeast-1")
    .option("kinesis.startingPosition", "LATEST")
    .load()
)
```

- The code first converts the binary data field from Kinesis into a readable string using CAST(data AS STRING) and stores it in a new column called data_string.
- It then filters out case-related events by checking for the presence of "case_no" and "status" in the string, parses them using a predefined case_schema, and extracts the structured fields.



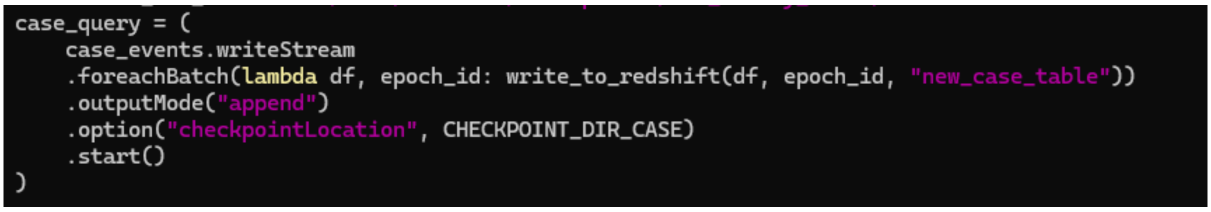
```
df_json = df_raw.withColumn("data_string", expr("CAST(data AS STRING)"))

case_events = df_json.filter(col("data_string").contains("case_no") & col("data_string").contains("status")) \
    .withColumn("json_parsed", from_json(col("data_string"), case_schema)) \
    .select("json_parsed.*")

survey_events = df_json.filter(col("data_string").contains("survey_id") & col("data_string").contains("q1")) \
    .withColumn("json_parsed", from_json(col("data_string"), survey_schema)) \
    .select("json_parsed.*")

redshift_jdbc_url = "jdbc:redshift://pritiranjan-project-one.008673239246.ap-northeast-1.redshift-serverless.amazonaws.com:5439/dev"
redshift_user = "admin"
redshift_pass = "Priti160503"
redshift_driver = "com.amazon.redshift.jdbc.Driver"
```

- This function write_to_redshift writes a Spark DataFrame (df) to a **Redshift table** using **JDBC** in **append** mode. It connects using the provided Redshift URL, username, password, and driver, and saves the data to the given table_name.



```
redshift_jdbc_url = "jdbc:redshift://pritiranjan-project-one.008673239246.ap-northeast-1.redshift-serverless.amazonaws.com:5439/dev"
redshift_user = "admin"
redshift_pass = "Priti160503"
redshift_driver = "com.amazon.redshift.jdbc.Driver"

def write_to_redshift(df, batchId, table_name):
    df.write \
      .format("jdbc") \
      .option("url", redshift_jdbc_url) \
      .option("dbtable", table_name) \
      .option("user", redshift_user) \
      .option("password", redshift_pass) \
      .option("driver", redshift_driver) \
      .mode("append") \
      .save()
```

- This starts a **streaming query** that sends micro batches and writes case_events to the **Redshift table cases** using the write_to_redshift function.
- It uses **append mode** for **fault tolerance** and **exactly-once processing**.



```
case_query = (
    case_events.writeStream
    .foreachBatch(lambda df, epoch_id: write_to_redshift(df, epoch_id, "new_case_table"))
    .outputMode("append")
    .option("checkpointLocation", CHECKPOINT_DIR_CASE)
    .start()
)
```

- This starts a **streaming query** that writes survey_events to the **Redshift table surveys** using write_to_redshift.
- It uses **append mode** to ensure **no duplicate writes**.

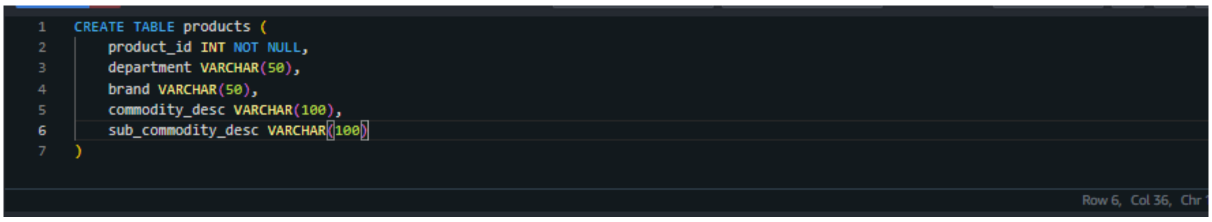
```
survey_query = (
    survey_events.writeStream
    .foreachBatch(lambda df, epoch_id: write_to_redshift(df, epoch_id, "new_survey_table"))
    .outputMode("append")
    .option("checkpointLocation", CHECKPOINT_DIR_SURVEY)
    .start()
)
```

- The lines case_query.awaitTermination() and survey_query.awaitTermination() ensure that both streaming queries keep running continuously.
- They block the main program from exiting, allowing Spark to keep processing incoming data from Kinesis and writing it to Redshift in real time.

```
case_query.awaitTermination()
survey_query.awaitTermination()
```

Loading Data to Redshift From S3

- Now we will load data from S3 to Redshift . The data I stored in S3 from earlier historical data and the data sent to S3 from the EMR Cluster using spark job.
- The data are stored in S3 in JSON Format . We have to create the Schema in the Redshift and then Load data from S3 into it .

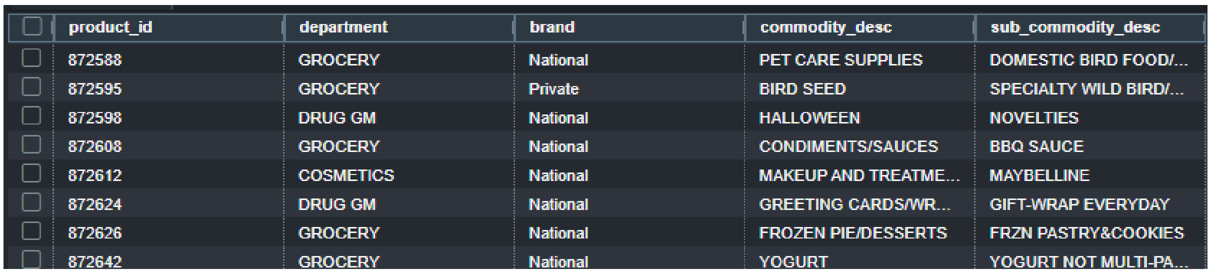


```
CREATE TABLE products (
    product_id INT NOT NULL,
    department VARCHAR(50),
    brand VARCHAR(50),
    commodity_desc VARCHAR(100),
    sub_commodity_desc VARCHAR(100)
)
```

- Now we have to load the data in table from S3. We will be using the below query:

```
COPY products
FROM 's3://priti-bucket-project1/priorities/'
IAM_ROLE '  arn:aws:iam::008673239246:role/redshift-s3fullacess-priti'
FORMAT AS JSON ‘auto’
REGION 'ap-northeast-1'
```

- Regarding the Data from the live streaming that is sent from the EMR Cluster is being loaded into the case and survey table automatically with the help of that script .
- We just have to keep in note that the schema we are providing in the code should be same as that to the schema of the table present in the redshift database .
- We can see the data has been added to the table , from the execution of the above code .



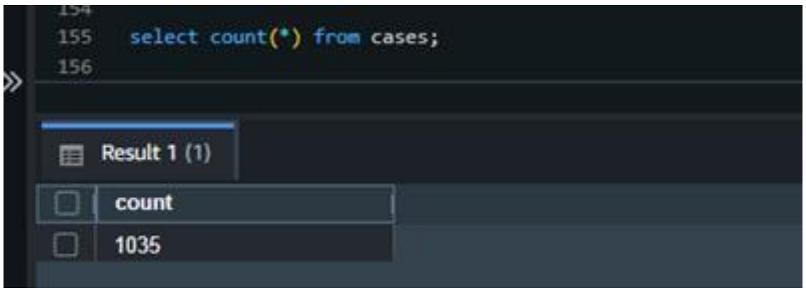
product_id	department	brand	commodity_desc	sub_commodity_desc
872588	GROCERY	National	PET CARE SUPPLIES	DOMESTIC BIRD FOOD/...
872595	GROCERY	Private	BIRD SEED	SPECIALTY WILD BIRD/...
872598	DRUG GM	National	HALLOWEEN	NOVELTIES
872608	GROCERY	National	CONDIMENTS/SAUCES	BBQ SAUCE
872612	COSMETICS	National	MAKEUP AND TREATME...	MAYBELLINE
872624	DRUG GM	National	GREETING CARDS/WR...	GIFT-WRAP EVERYDAY
872626	GROCERY	National	FROZEN PIE/DESSERTS	FRZN PASTRY&COOKIES
872642	GROCERY	National	YOGURT	YOGURT NOT MULTI-PA...

- Same goes for all the tables . For each table we load the data from the S3 bucket to the respective table .
- Further based on this data we will be running the queries . Some of the queries do need multiple tables to be accessed i.e both fact and dimensional tables . we can access both with the help of matching columns .

Query Execution on Redshift

- Total numbers of cases

```
SELECT COUNT(*) FROM CASES
```



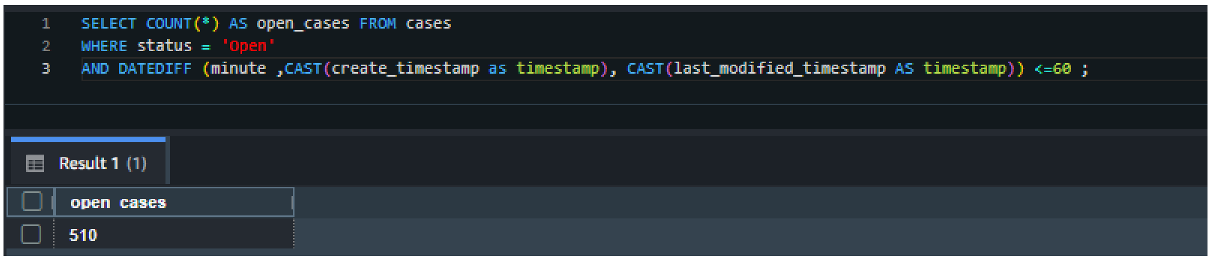
```
select count(*) from cases;
```

Result 1 (1)

count
1035

- Total open cases in the last 1 hour

```
SELECT COUNT(*) AS open_cases FROM cases WHERE status = ‘Open’
AND DATEDIFF (minute ,CAST(create_timestamp as timestamp),
CAST(last_modified_timestamp AS timestamp)) <=60 ;
```



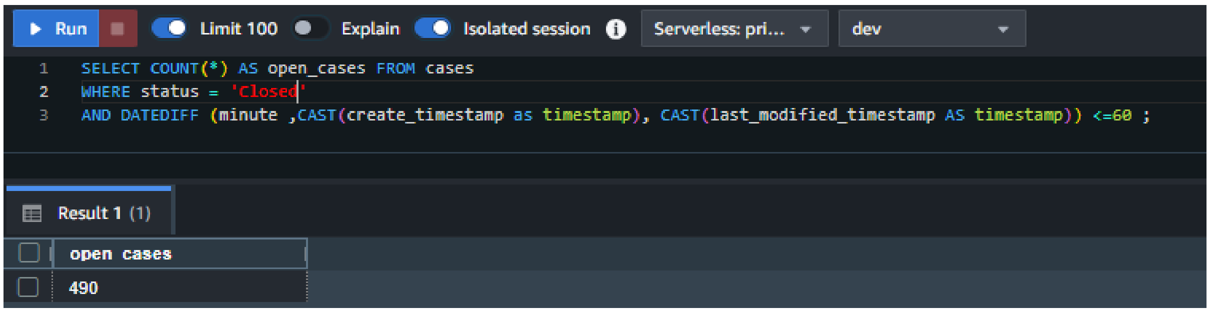
```
SELECT COUNT(*) AS open_cases FROM cases
WHERE status = 'Open'
AND DATEDIFF (minute ,CAST(create_timestamp as timestamp), CAST(last_modified_timestamp AS timestamp)) <=60 ;
```

Result 1 (1)

open_cases
510

- Total closed cases in the last 1 hour

```
SELECT COUNT(*) AS open_cases FROM cases WHERE status = ‘Closed’
AND DATEDIFF (minute ,CAST(create_timestamp as timestamp),
CAST(last_modified_timestamp AS timestamp)) <=60 ;
```



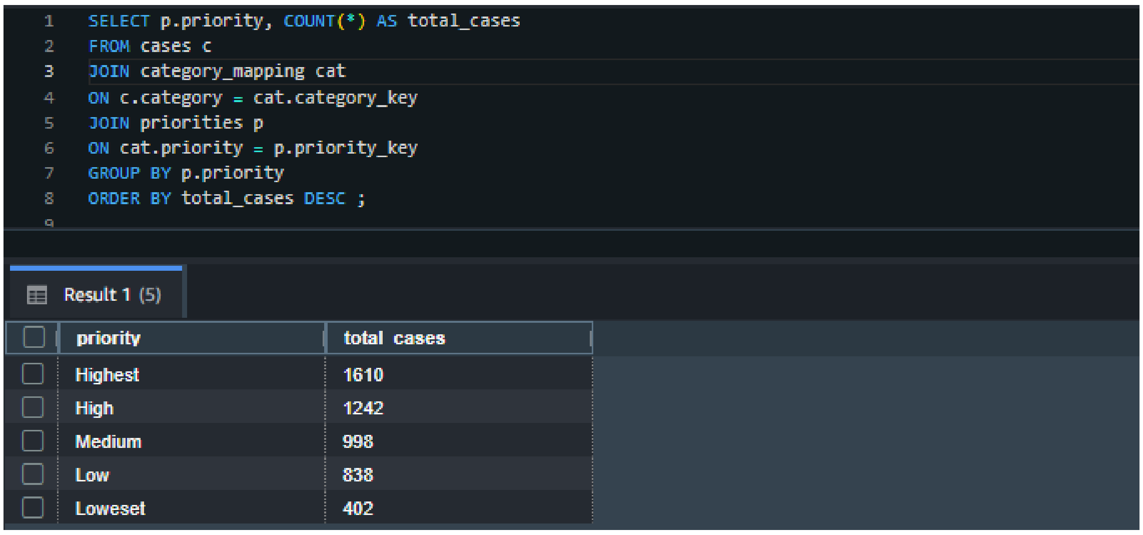
```
SELECT COUNT(*) AS open_cases FROM cases
WHERE status = 'Closed'
AND DATEDIFF (minute ,CAST(create_timestamp as timestamp), CAST(last_modified_timestamp AS timestamp)) <=60 ;
```

Result 1 (1)

open_cases
490

• Total priority cases

```
SELECT p.priority, COUNT(*) AS total_cases
FROM cases c
JOIN categories cat
ON c.category = cat.category_key
JOIN priorities p
ON cat.priority = p.priority_key
GROUP BY p.priority
ORDER BY total_cases DESC ;
```



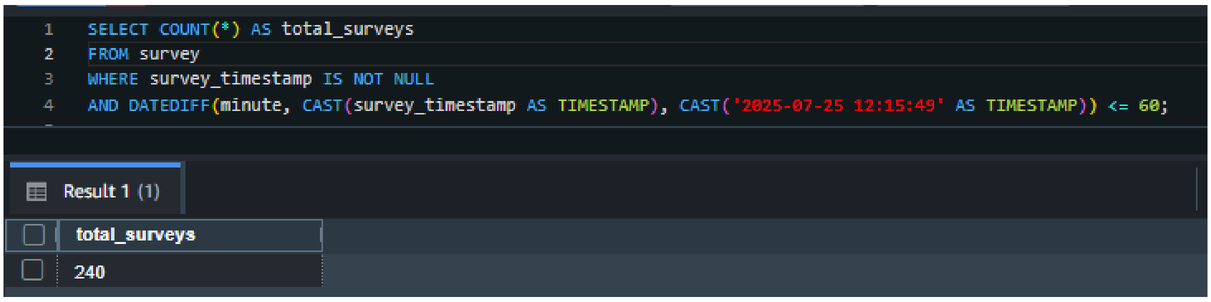
```
SELECT p.priority, COUNT(*) AS total_cases
FROM cases c
JOIN category_mapping cat
ON c.category = cat.category_key
JOIN priorities p
ON cat.priority = p.priority_key
GROUP BY p.priority
ORDER BY total_cases DESC ;
```

Result 1 (5)

priority	total_cases
Highest	1610
High	1242
Medium	998
Low	838
Loweset	402

• Total number of surveys in the last 1 hour

```
SELECT COUNT (*) AS total_surveys
FROM survey
WHERE survey_timestamp IS NOT NULL
AND DATEDIFF(minute,CAST(survey_timestamp as
TIMESTAMP),CAST('2025-07-25 12:15:49' AS TIMESTAMP)) <=60 ;
```



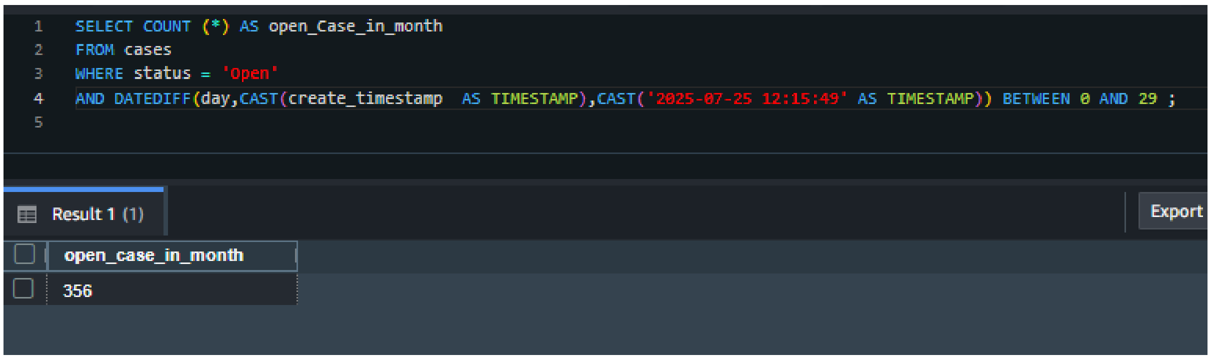
```
SELECT COUNT(*) AS total_surveys
FROM survey
WHERE survey_timestamp IS NOT NULL
AND DATEDIFF(minute, CAST(survey_timestamp AS TIMESTAMP), CAST('2025-07-25 12:15:49' AS TIMESTAMP)) <= 60;
```

Result 1 (1)

total_surveys
240

- Total open cases in a day/week/month

```
SELECT COUNT (*) AS open_Case_in_month
FROM cases
WHERE status = ‘Open’
AND DATEDIFF(day,CAST(create_timestamp  AS TIMESTAMP),CAST(‘2025-07-25 12:15:49’ AS TIMESTAMP)) BETWEEN 0 AND 29 ;
```

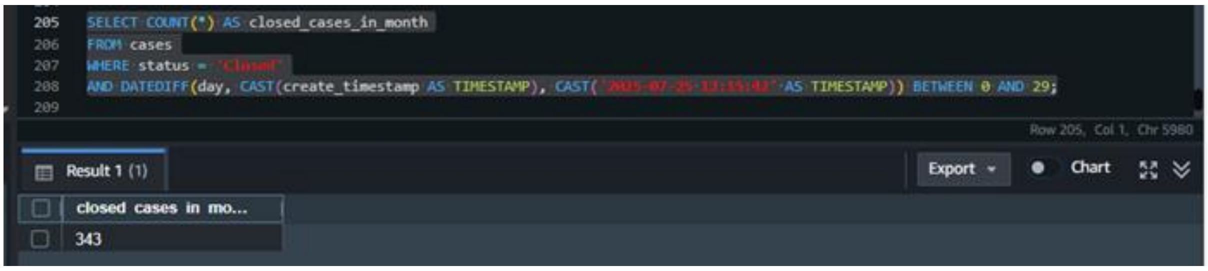


```
SELECT COUNT (*) AS open_Case_in_month
FROM cases
WHERE status = 'Open'
AND DATEDIFF(day,CAST(create_timestamp  AS TIMESTAMP),CAST('2025-07-25 12:15:49' AS TIMESTAMP)) BETWEEN 0 AND 29 ;
```

open_case_in_month
356

- Total closed cases in a day/week/month

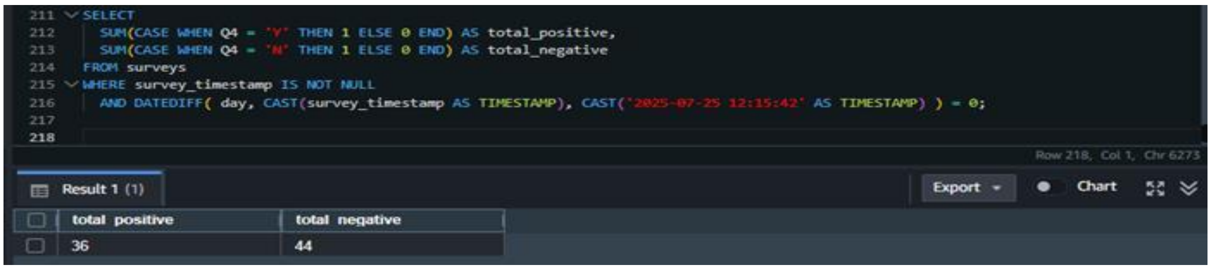
```
SELECT COUNT (*) AS closed_Case_in_month
FROM cases
WHERE status = ‘Closed’
AND DATEDIFF(day,CAST(create_timestamp  AS TIMESTAMP),CAST(‘2025-07-25 12:15:49’ AS TIMESTAMP)) BETWEEN 0 AND 29 ;
```



```
SELECT COUNT(*) AS closed_cases_in_month
FROM cases
WHERE status = 'Closed'
AND DATEDIFF(day, CAST(create_timestamp AS TIMESTAMP), CAST('2025-07-25 12:15:42' AS TIMESTAMP)) BETWEEN 0 AND 29;
```

closed cases in mo...
343

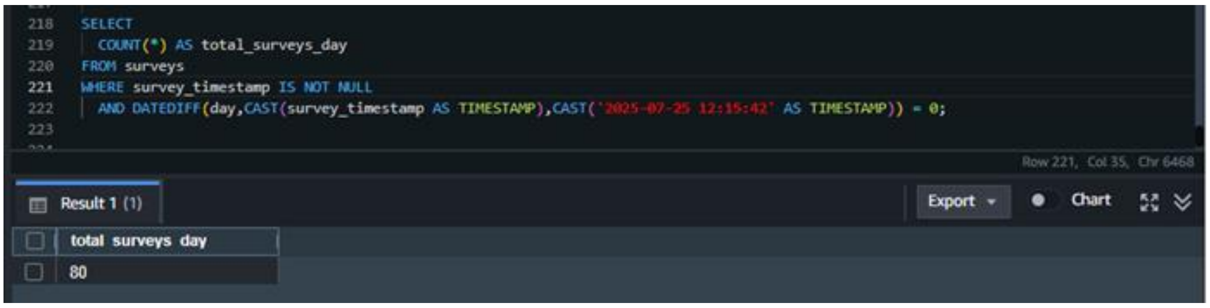
- Total positive/negative responses in a day/week/month



```
SELECT
  SUM(CASE WHEN Q4 = 'Y' THEN 1 ELSE 0 END) AS total_positive,
  SUM(CASE WHEN Q4 = 'N' THEN 1 ELSE 0 END) AS total_negative
FROM surveys
WHERE survey_timestamp IS NOT NULL
  AND DATEDIFF( day, CAST(survey_timestamp AS TIMESTAMP), CAST('2025-07-25 12:15:42' AS TIMESTAMP) ) = 0;
```

total positive	total negative
36	44

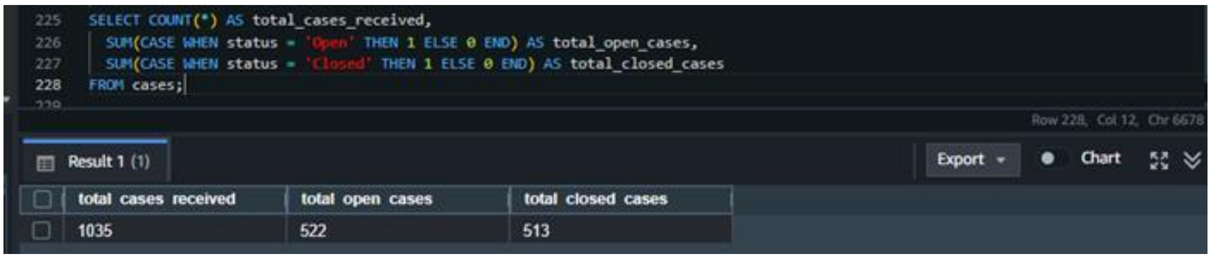
• Total number of surveys in a day/week/month Real-time KPIs



```sql
SELECT
  COUNT(*) AS total_surveys_day
FROM surveys
WHERE survey_timestamp IS NOT NULL
  AND DATEDIFF(day,CAST(survey_timestamp AS TIMESTAMP),CAST('2025-07-25 12:15:42' AS TIMESTAMP)) = 0;
```

total surveys day
80

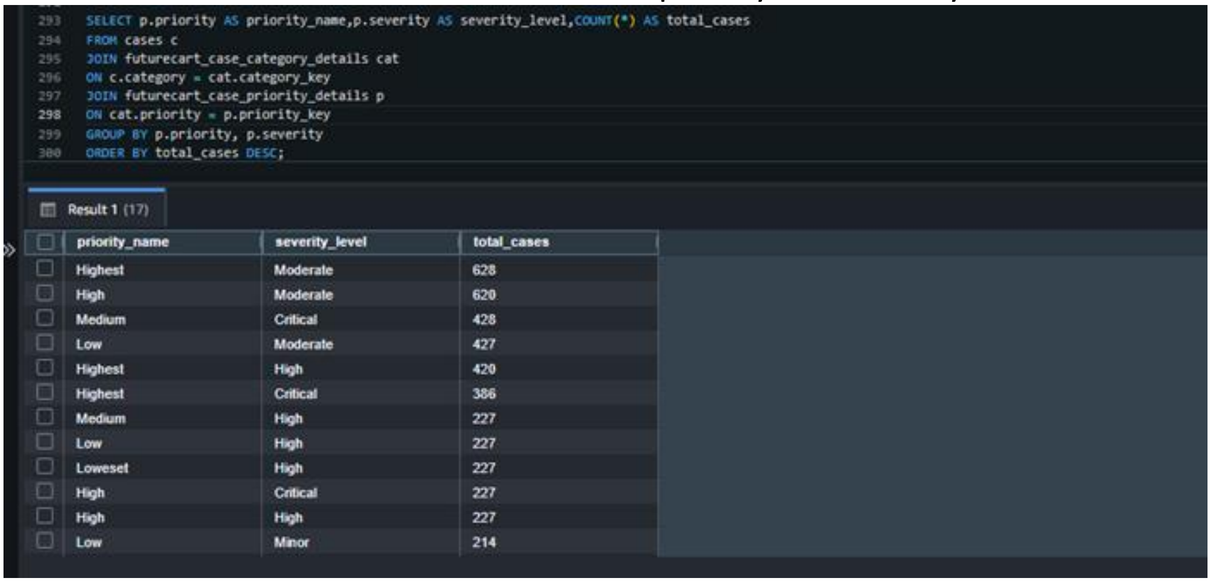
• Total numbers of cases that are open and closed out of the number of cases received



```sql
SELECT COUNT(*) AS total_cases_received,
  SUM(CASE WHEN status = 'Open' THEN 1 ELSE 0 END) AS total_open_cases,
  SUM(CASE WHEN status = 'Closed' THEN 1 ELSE 0 END) AS total_closed_cases
FROM cases;
```

total cases received	total open cases	total closed cases
1035	522	513

• Total number of cases received based on priority and severity



```sql
SELECT p.priority AS priority_name,p.severity AS severity_level,COUNT(*) AS total_cases
FROM cases c
JOIN futurecart_case_category_details cat
ON c.category = cat.category_key
JOIN futurecart_case_priority_details p
ON cat.priority = p.priority_key
GROUP BY p.priority, p.severity
ORDER BY total_cases DESC;
```

priority_name	severity_level	total_cases
Highest	Moderate	628
High	Moderate	620
Medium	Critical	428
Low	Moderate	427
Highest	High	420
Highest	Critical	386
Medium	High	227
Low	High	227
Lowest	High	227
High	Critical	227
High	High	227
Low	Minor	214