

FutureCart CRM Analytics Project Documentation

1. Business Objective

FutureCart Inc. aims to enhance customer satisfaction and loyalty through a robust CRM strategy. The company operates over 5000 retail stores and an e-commerce platform across India. A dedicated Customer Care team handles complaints and feedback, which are logged as cases and followed by customer surveys.

The goal is to:

- Improve after-sales service.
- Track and optimize customer support performance.
- Enable real-time and historical insights for leadership decision-making.

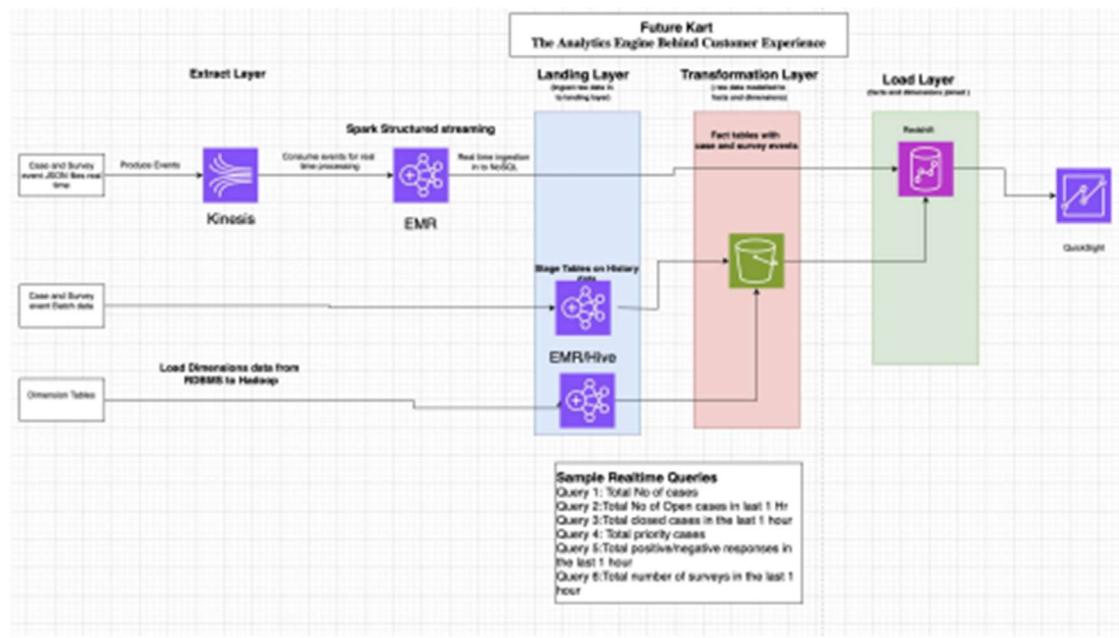
2. Business Process Overview

- Customers contact support via **calls, chat, or email**.
- A **Customer Care Representative (CCR)** logs each interaction as a **case**.
- Cases are categorized and prioritized based on business rules.
- After resolution, customers receive a **survey** to rate their experience.
- All data is collected and analyzed to monitor **KPIs** and improve service.

3. Key KPIs Tracked

Type	KPI
Real-Time	Open/closed cases in last hour, priority cases, survey sentiment
Batch	Daily/weekly/monthly case volumes, survey counts, sentiment trends

4. Technical Architecture: Data Flow Layers



Layer	Purpose	Tools
Extract	Ingest raw data	Kinesis (real-time), S3 (batch), EC2 MySQL (dimension)
Landing	Store raw data	EMR + Hive
Transformation	Clean & model data	Hive/Spark
Load	Store for analytics	Redshift
Visualization	Business insights	Quick Sight

5. Step-by-Step Execution Plan

Step 1: Dimension Data Setup

- Create Base Dimension table and load 8 datasets.
- Create MySQL tables on EC2 using SQL scripts.
- Load .txt files into MySQL using LOAD DATA INFILE.

Step 2: Batch Ingestion

- Use Sqoop to import MySQL dimension tables into Hive.
- Use Spark to export Hive tables to S3.
- Load S3 data into Redshift.

Step 3: Historical Data Setup

- Use generate_historical_data.py to simulate 10 days of case and survey data in JSON format.
- Load JSON files into HDFS.
- Create Hive tables for historical case and survey data.

Step 4: Real-Time Data Simulation

- Use stream_to_kinesis.py to generate real-time case and survey events.
- Send JSON data to Kinesis streams:
 - futurecart_case_event
 - futurecart_survey_event

Step 5: Real-Time Processing

- Create a consumer application to read from Kinesis.
- Parse incoming JSON:
 - If it's a case → load into Redshift case table.
 - If it's a survey → load into Redshift survey table.

Step 6: Data Export and Load

- Export transformed Hive data to S3 using Spark DataFrames.
- Load S3 data into Redshift using COPY or Glue.

Step 7: KPI Queries

- Write SQL queries on Redshift to calculate KPIs:
 - Case volumes
 - SLA compliance
 - Survey sentiment.

Dimension Data to S3

Dimension Data Workflow EC2 → MariaDB → EMR/Hive → Spark.

Phase	Description
1	Setup EC2 and install MariaDB
2	Create database and tables
3	Upload .txt files from local to EC2
4	Load data into MariaDB
5	Setup EMR and Hive
6	Transfer data from MariaDB to Hive using Sqoop
7	Export Hive tables to S3 using Spark

Phase 1: Setup EC2 and Install MariaDB

◆ Step 1.1: Launch EC2 Instance

- **AMI:** Amazon Linux 2
- **Instance Type:** t2.medium or higher
- **Storage:** 20 GB
- **Security Group:**
 - Allow **SSH (port 22)** from your IP
 - Allow **MySQL (port 3306)** from EMR subnet or your IP
 - Allow http
 - Allow redshift

◆ Step 1.2: Connect to EC2

```
ssh -i your-key.pem ec2-user@<EC2-Public-IP>
```

◆ Step 1.3 Install MariaDB 10.5 on Amazon Linux 2023

1. **Update packages:** sudo dnf update -y
2. **Install MariaDB 10.5:** sudo dnf install mariadb105-server mariadb105 -y

3. Start and enable MariaDB service:

```
sudo systemctl start mariadb  
sudo systemctl enable mariadb
```

4. Secure MariaDB installation:

```
sudo mysql_secure_installation
```

Follow the prompts to set root password and remove test users/databases.

5. Verify MariaDB is running:

```
sudo systemctl status mariadb
```

Phase 2: Create Database and Tables

◆ **Step 2.1: Login to MariaDB**

```
mysql -u root -p
```

◆ **Step 2.2: Create Database**

```
CREATE DATABASE futurecart_crm;
```

```
USE futurecart_crm;
```

◆ **Step 2.3: Create All 8 Tables**

-- SQL for futurecart_calendar_details

```
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)  
);
```

-- SQL for futurecart_call_center_details

```
CREATE TABLE futurecart_call_center_details (  
    call_center_id VARCHAR(10),  
    call_center_vendor VARCHAR(50),  
    location VARCHAR(50),  
    country VARCHAR(50)  
);
```

-- SQL for futurecart_case_category_details

```
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),  
);
```

-- SQL for futurecart_case_country_details

```
CREATE TABLE futurecart_case_country_details (  
    id INT,  
    name VARCHAR(75),  
    alpha_2 VARCHAR(2),  
    alpha_3 VARCHAR(3)  
);
```

-- SQL for futurecart_case_priority_details

```
CREATE TABLE futurecart_case_priority_details (
    priority_key VARCHAR(5),
    priority VARCHAR(20),
    severity VARCHAR(100),
    sla VARCHAR(100)
);
```

-- SQL for futurecart_employee_details

```
CREATE TABLE futurecart_employee_details (
    emp_key INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    gender VARCHAR(10),
    ldap VARCHAR(50),
    hire_date DATE,
    manager VARCHAR(50)
);
```

-- SQL for futurecart_product_details

```
CREATE TABLE futurecart_product_details (
    product_id VARCHAR(20),
    department VARCHAR(50),
    brand VARCHAR(50),
    commodity_desc VARCHAR(100),
    sub_commodity_desc VARCHAR(100)
);
```

```
-- SQL for futurecart_survey_question_details (// put range in `` as The problem here is  
that range is a reserved keyword)
```

```
CREATE TABLE futurecart_survey_question_details (  
    question_id VARCHAR(10),  
    question_desc VARCHAR(255),  
    response_type VARCHAR(50),  
    `range` VARCHAR(20),  
    negative_response_range VARCHAR(20),  
    neutral_response_range VARCHAR(20),  
    positive_response_range VARCHAR(20)  
);
```

```
MariaDB [futurecart_crm]> show tables  
    -> ;  
+-----+  
| Tables_in_futurecart_crm |  
+-----+  
| 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 |  
+-----+  
8 rows in set (0.001 sec)
```

📁 Phase 3: Upload 8 .txt Files to EC2

- ◆ **Step 3.1: From Local to EC2**

```
scp -i your-key.pem futurecart_*.txt ec2-user@<EC2-Public-IP>/home/ec2-user/
```

💻 Phase 4: Load Data into MariaDB

4.1: Upload files to S3 then extract it from ec2

```
aws s3 cp s3://tbsm-source/db/ ~/ --recursive
```

CHECK FOR THE FILES

```
[ec2-user@ip-172-31-1-66 ~]$ ls
futurecart_calendar_details.txt
futurecart_call_center_details.txt
futurecart_case_category_details.txt
futurecart_case_country_details.txt
futurecart_case_priority_details.txt
futurecart_employee_details.txt
futurecart_product_details.txt
futurecart_survey_question_details.txt
[ec2-user@ip-172-31-1-66 ~]$
```

Restart MariaDB

```
sudo systemctl restart mariadb
```

4.2: Login again and setup permission for loading for marida db

```
sudo mysql -u root -p
```

4.3. Allow MariaDB to load local files

Check MariaDB config allows local file loading. In your MariaDB session:

```
- SHOW VARIABLES LIKE 'local_infile';
```

4.4 If it's OFF, enable it:

```
- SET GLOBAL local_infile = 1;
```

Step 4.5:

Use database

```
use futurecart_crm;
```

4.6: Load One Table (Example: Calendar)

```
-LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_calendar_details.txt'
```

```
INTO TABLE futurecart_calendar_details
```

```
FIELDS TERMINATED BY ''
```

```
LINES TERMINATED BY '\n'
```

```
IGNORE 1 LINES;
```

```
LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_call_center_details.txt'
```

```
INTO TABLE futurecart_call_center_details
```

```
FIELDS TERMINATED BY ''
```

LINES TERMINATED BY '\n'

IGNORE 1 LINES;

LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_case_category_details.txt'

INTO TABLE futurecart_case_category_details

FIELDS TERMINATED BY ''

LINES TERMINATED BY '\n'

IGNORE 1 LINES;

LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_case_country_details.txt'

INTO TABLE futurecart_case_country_details

FIELDS TERMINATED BY ''

LINES TERMINATED BY '\n'

IGNORE 1 LINES;

LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_case_priority_details.txt'

INTO TABLE futurecart_case_priority_details

FIELDS TERMINATED BY ''

LINES TERMINATED BY '\n'

IGNORE 1 LINES;

LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_employee_details.txt'

INTO TABLE futurecart_employee_details

FIELDS TERMINATED BY ''

LINES TERMINATED BY '\n'

IGNORE 1 LINES;

LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_product_details.txt'

```
INTO TABLE futurecart_product_details  
FIELDS TERMINATED BY ''  
LINES TERMINATED BY '\n'  
IGNORE 1 LINES;
```

```
LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_survey_question_details.txt'  
INTO TABLE futurecart_survey_question_details  
FIELDS TERMINATED BY ''  
LINES TERMINATED BY '\n'  
IGNORE 1 LINES;
```

Cross check for count and run one or two select cmd to check data

```
MariaDB [futurecart_crm]> SELECT COUNT(*) FROM futurecart_employee_details;  
ls LIMIT 5;  
+-----+  
| COUNT(*) |  
+-----+  
| 300024 |  
+-----+  
1 row in set (0.077 sec)
```

ALTERNATIVE:- Clean the loading file and converted in csv and upload in table

Phase 5: Setup EMR Cluster

◆ Step 5.1: Launch EMR

- **Release:** EMR 6.15.0
- **Applications:** Hive, Hadoop, Spark, Sqoop
- **Instance Type:** m5.xlarge (3 nodes)
- **Log URI:** s3://your-bucket/emr-logs/
- **IAM Roles:** ec2-s3-full-access

Phase 6: Transfer Data via Sqoop

◆ Step 6.1: SSH into EMR Master

```
ssh -i your-key.pem hadoop@<EMR-Master-IP>
```

Verify HDFS directories and create user directory if needed:

```
hdfs dfs -ls
```

```
hdfs dfs -mkdir -p /user/ec2-user
```

```
hdfs dfs -chown ec2-user /user/ec2-user
```

Launch Hive CLI and create/use the database:

```
hive
```

Inside Hive CLI:

```
CREATE DATABASE IF NOT EXISTS dimension;
```

```
SHOW DATABASES;
```

```
USE dimension;
```

```
EXIT;
```

◆ Step 6.2: Run Sqoop Import (Example: Calendar Table)

```
sqoop import \
--connect jdbc:mysql://172.31.8.1:3306/futurecart_crm \
--username root \
--password tekavade123 \
--table futurecart_calendar_details \
--hive-import \
--hive-database dimension \
--hive-table futurecart_calendar_details \
--create-hive-table \
--fields-terminated-by '\t' \
-m 1 \
--driver org.mariadb.jdbc.Driver
```

◆ **Step 6.3: Import Remaining Tables**

Repeat the Sqoop import command for all other tables, changing --table and --hive-table accordingly:

```
sqoop import \
--connect jdbc:mysql://172.31.8.1:3306/futurecart_crm \
--username root \
--password tekavade123 \
--table futurecart_call_center_details \
--hive-import \
--hive-database dimension \
--hive-table futurecart_call_center_details \
--create-hive-table \
--fields-terminated-by '\t' \
-m 1 \
--driver org.mariadb.jdbc.Driver
```

Similarly for:

- futurecart_case_category_details
- futurecart_case_country_details
- futurecart_case_priority_details
- futurecart_employee_details
- futurecart_product_details
- futurecart_survey_question_details

Phase 7: Export Hive Tables to S3 via Spark

◆ **Step 7.1: Create Spark Script (export_hive_tables_to_s3.py)**

```
from pyspark.sql import SparkSession
```

```
def export_table(spark, hive_db, table_name, s3_path):

    full_table_name = f"{hive_db}.{table_name}"
    print(f"Exporting {full_table_name} to {s3_path} ...")

    # Read Hive table

    df = spark.table(full_table_name)

    # Write as Parquet to S3 (overwrite if exists)

    df.write.mode("overwrite").parquet(s3_path)

    print(f"Done exporting {full_table_name}")

def main():

    spark = SparkSession.builder \
        .appName("HiveToS3Export") \
        .enableHiveSupport() \
        .getOrCreate()

    hive_db = "dimension"

    s3_base_path = "s3://tbsm-destination/db/"

    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"
    ]

    for table in tables:

        s3_path = f"{s3_base_path}{table}"

        export_table(spark, hive_db, table, s3_path)
```

```
spark.stop()

if __name__ == "__main__":
    main()

◆ Step 7.2: Run Spark Job

spark-submit export_hive_tables_to_s3.py
```

Final Output

- All 8 dimension tables are:
 - Persisted in EC2 MariaDB
 - Imported into Hive on EMR
 - Exported to S3 in Parquet format
-

Historical Data Ingestion

Overview

- Generate 10 days of historical case & survey data as JSON Lines files locally on EMR.
 - Upload JSON data to HDFS.
 - Create Hive external tables on JSON data.
 - Export Hive tables to S3 in Parquet format using Spark.
-

Step 2.1: Generate Historical JSON Data

1. Create Python script to generate JSON data

```
nano generate_historical_data.py
```

2. Copy-paste this Python code into generate_historical_data.py:

```
import os  
  
import json  
  
import random  
  
from datetime import datetime, timedelta  
  
NUM_DAYS = 10  
  
CASES_PER_DAY = 100  
  
SURVEYS_PER_DAY = 80  
  
OUTPUT_DIR = "historical_data_json"  
  
def random_case_no():  
  
    return str(random.randint(600000, 700000))  
  
def random_timestamp(day_offset):  
  
    date = datetime.now() - timedelta(days=day_offset)  
  
    return date.strftime("%Y-%m-%d %H:%M:%S")  
  
def generate_case(case_no, day_offset):  
  
    return {  
  
        "status": random.choice(["Open", "Closed"]),
```

```

    "category": f"CAT{random.randint(1,5)}",
    "sub_category": f"SCAT{random.randint(1,20)}",
    "last_modified_timestamp": random_timestamp(day_offset),
    "case_no": case_no,
    "create_timestamp": random_timestamp(day_offset),
    "created_employee_key": str(random.randint(200000, 300000)),
    "call_center_id": f"C-{random.randint(100,120)}",
    "product_code": str(random.randint(9000000, 9999999)),
    "country_cd": random.choice(["IN", "US", "BR", "DE", "AU"]),
    "communication_mode": random.choice(["Email", "Call", "Chat"])
}

def generate_survey(case_no, day_offset):
    return {
        "survey_id": f"S-{random.randint(500000, 599999)}",
        "case_no": case_no,
        "survey_timestamp": random_timestamp(day_offset),
        "Q1": random.randint(1, 10),
        "Q2": random.randint(1, 10),
        "Q3": random.randint(1, 10),
        "Q4": random.choice(["Y", "N"]),
        "Q5": random.randint(1, 10)
    }

def ensure_dirs():
    os.makedirs(f"{OUTPUT_DIR}/cases", exist_ok=True)
    os.makedirs(f"{OUTPUT_DIR}/surveys", exist_ok=True)

def main():
    ensure_dirs()
    for day in range(1, NUM_DAYS + 1):

```

```

case_nos = [random_case_no() for _ in range(CASES_PER_DAY)]

cases = [generate_case(cn, day) for cn in case_nos]

surveys = [generate_survey(cn, day) for cn in random.sample(case_nos,
SURVEYS_PER_DAY)]

with open(f"{OUTPUT_DIR}/cases/case_data_day{day}.json", "w") as f:

    for record in cases:

        f.write(json.dumps(record) + "\n")

with open(f"{OUTPUT_DIR}/surveys/survey_data_day{day}.json", "w") as f:

    for record in surveys:

        f.write(json.dumps(record) + "\n")

print(f" ✅ JSON Lines generated for {NUM_DAYS} days in '{OUTPUT_DIR}'")

```

if __name__ == "__main__":

 main()

3. Run the script to generate JSON data locally

python3 generate_historical_data.py

- This will create a folder historical_data_jsonl with two subfolders: cases and surveys.
 - Each contains 10 JSON files — one for each day.
-

Step 2.2: Upload JSON Data to HDFS

1. Create necessary HDFS directories (run these commands):

sudo -u hdfs hdfs dfs -mkdir -p /data/historical/cases

sudo -u hdfs hdfs dfs -mkdir -p /data/historical/surveys

2. Give your user ownership of these directories:

sudo -u hdfs hdfs dfs -chown -R ec2-user:ec2-user /data/historical

3. Upload JSON files to HDFS:

hdfs dfs -put -f historical_data_jsonl/cases/*.json /data/historical/cases/

hdfs dfs -put -f historical_data_jsonl/surveys/*.json /data/historical/surveys/

Step 2.3: Create Hive External Tables

1. Open Hive shell

```
>>Hive  
>>Create database database_name;
```

2. Run these commands in Hive shell to create external tables

```
CREATE EXTERNAL TABLE case_details (
```

```
    status STRING,  
    category STRING,  
    sub_category STRING,  
    last_modified_timestamp STRING,  
    case_no STRING,  
    create_timestamp STRING,  
    created_employee_key STRING,  
    call_center_id STRING,  
    product_code STRING,  
    country_cd STRING,  
    communication_mode STRING
```

```
)
```

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  
LOCATION '/data/historical/cases';
```

```
CREATE EXTERNAL TABLE case_survey_details (
```

```
    survey_id STRING,  
    case_no STRING,  
    survey_timestamp STRING,  
    Q1 INT,  
    Q2 INT,  
    Q3 INT,
```

```
Q4 STRING,  
Q5 INT  
)  
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  
LOCATION '/data/historical/surveys';
```

3. Check the data loaded

```
SELECT COUNT(*) FROM case_details; (800)  
SELECT COUNT(*) FROM case_survey_details; (1000)
```

Step 2.4: Export Hive Tables to S3 as Parquet Using Spark

1. Create Spark export script

```
nano export_historical_to_s3.py
```

2. Paste this Spark Python code

```
from pyspark.sql import SparkSession  
  
def main():  
    spark = SparkSession.builder \  
        .appName("ExportHistoricalToS3") \  
        .enableHiveSupport() \  
        .getOrCreate()  
  
    tables = {  
        "case_details": "s3://tbsm-core/historical/case/",  
        "case_survey_details": "s3://tbsm-core/historical/survey/"  
    }  
  
    for table, path in tables.items():  
        print(f"Exporting {table} to {path} ...")  
        df = spark.sql(f"SELECT * FROM {table}")  
        df.write.mode("overwrite").parquet(path)  
        print(f"Exported {table} successfully.")
```

```
spark.stop()

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

3. Run the Spark job

```
spark-submit export_historical_to_s3.py
```

All done!

- Historical data JSON generated locally.
- Data uploaded to HDFS.
- Hive tables created on JSON data.
- Data exported from Hive tables to S3 as Parquet.

Real-Time Streaming

EC2 → Kinesis → Spark Streaming on EMR → Redshift (No S3)

Prerequisites

- EC2 Instance with boto3 and Kinesis permissions
 - EMR Cluster with Spark
 - Redshift Serverless Workgroup created and Publicly Accessible
 - IAM Role with Redshift and Kinesis access
-

Step 1: Launch EC2 for Data Generation

1.1 Launch EC2 Instance

- Amazon Linux 2
- Attach IAM Role with Kinesis write access
- Install packages:

```
sudo yum update -y
```

```
pip3 install boto3
```

1.2 Upload 000000_0 and stream_to_kinesis.py

Use scp or AWS Console to upload the files to your EC2.

1.3 Edit your Python script

In stream_to_kinesis.py, ensure:

```
stream_name = 'case-survey-stream'
```

```
region_name = 'ap-south-1'
```

1.4 Run the generator

```
python3 stream_to_kinesis.py
```

This continuously sends random case/survey JSON records to your Kinesis stream.

Step 2: Kinesis Setup

2.1 Create Stream

- Go to AWS Console > Kinesis > Data Streams

- Stream name: case-survey-stream
 - 1 shard
-

Step 3: EMR Setup with Spark

3.1 Launch EMR Cluster

- Select **Amazon EMR 6.x**
- Applications: Select **Spark, Hadoop**
- Instance type: m5.xlarge (or t3.medium for testing)
- Edit networking:
 - Place in **same VPC** as Redshift
 - Set **Security Group** with **outbound to Redshift port 5439**

3.2 SSH into EMR Master Node

```
ssh -i your-key.pem hadoop@<EMR-MASTER-PUBLIC-IP>
```

3.3 Create Spark Streaming script kinesis_to_red.py

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import from_json, col, expr

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

# Initialize Spark Session

spark = SparkSession.builder.appName("KinesisToRedshift").getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

# Define schema for case and survey records

case_schema = StructType() \

    .add("case_no", StringType()) \

    .add("create_timestamp", StringType()) \

    .add("last_modified_timestamp", StringType()) \

    .add("created_employee_key", StringType()) \

    .add("call_center_id", StringType()) \

    .add("status", StringType()) \
```

```

    .add("category", StringType()) \
    .add("sub_category", StringType()) \
    .add("communication_mode", StringType()) \
    .add("country_cd", StringType()) \
    .add("product_code", StringType())

survey_schema = StructType() \
    .add("survey_id", StringType()) \
    .add("case_no", StringType()) \
    .add("survey_timestamp", StringType()) \
    .add("Q1", IntegerType()) \
    .add("Q2", StringType()) \
    .add("Q3", IntegerType()) \
    .add("Q4", StringType()) \
    .add("Q5", IntegerType())

# Read from Kinesis Stream

raw_stream = spark.readStream \
    .format("aws-kinesis") \
    .option("kinesis.region", "ap-south-1") \
    .option("kinesis.streamName", "futurekart-tbsm") \
    .option("kinesis.consumerType", "GetRecords") \
    .option("kinesis.endpointUrl", "https://kinesis.ap-south-1.amazonaws.com") \
    .option("kinesis.startingposition", "LATEST") \
    .load()

# Parse JSON strings

json_str_df = raw_stream.selectExpr("CAST(data AS STRING) as json_data")

# Filter and parse Survey records

survey_df = json_str_df \
    .filter(expr("json_data LIKE '%survey_id%'"))

```

```

.withColumn("parsed", from_json(col("json_data"), survey_schema)) \
.select("parsed.*")

# Filter and parse Case records

case_df = json_str_df \
.filter(expr("NOT json_data LIKE '%survey_id%'")) \
.withColumn("parsed", from_json(col("json_data"), case_schema)) \
.select("parsed.*")

# Redshift connection details

redshift_jdbc_url = "jdbc:redshift://tbsm-wg.008673239246.ap-south-1.redshift-
serverless.amazonaws.com:5439/tbsm-red"

redshift_user = "admin"

redshift_password = "Tekavade123"

temp_s3_dir = "s3://tbsm-encore/redshift-temp/"

aws_iam_role = "arn:aws:iam::008673239246:role/Redshift-S3Role-tbsm"

# Batch write function

def write_to_redshift(df, batch_id, table_name, tempdir):

    if df.isEmpty():

        return

    print(f"Writing batch {batch_id} to Redshift table: {table_name}")

    df.write \
        .format("io.github.spark_redshift_community.spark.redshift") \
        .option("url", f"{redshift_jdbc_url}?user={redshift_user}&password={redshift_password}") \
        \
        .option("dbtable", table_name) \
        .option("tempdir", tempdir) \
        .option("aws_iam_role", aws_iam_role) \
        .mode("append") \
        .save()

```

```

# Write case data to Redshift

case_query = case_df.writeStream \
    .foreachBatch(lambda df, id: write_to_redshift(df, id, "public.case_table", "s3://tbsm-red/case/")) \
    .start()

# Write survey data to Redshift

survey_query = survey_df.writeStream \
    .foreachBatch(lambda df, id: write_to_redshift(df, id, "public.survey_table", "s3://tbsm-red/survey/")) \
    .start()

# Await termination

spark.streams.awaitAnyTermination()

```

3.4 Submit the Spark Job

```

spark-submit \
--jars s3://awslabs-code-us-east-1/spark-sql-kinesis-connector/spark-streaming-sql-kinesis-connector_2.12-1.0.0.jar \
--packages com.amazon.redshift:redshift-jdbc42:2.1.0.33,org.apache.spark:spark-avro_2.12:3.5.6,io.github.spark-redshift-community:spark-redshift_2.12:6.4.3-spark_3.5 \
kinesis_stream_to_redshift.py

```

Step 4: Redshift Setup

4.1 Create Tables

In Redshift SQL editor:

```

CREATE TABLE case_table (
    case_no VARCHAR,
    status VARCHAR,
    category VARCHAR,

```

```
sub_category VARCHAR,  
last_modified_timestamp VARCHAR,  
create_timestamp VARCHAR,  
created_employee_key VARCHAR,  
call_center_id VARCHAR,  
product_code VARCHAR,  
country_cd VARCHAR,  
communication_mode VARCHAR  
);  
  
CREATE TABLE survey_table (  
    survey_id VARCHAR,  
    case_no VARCHAR,  
    survey_timestamp VARCHAR,  
    Q1 INT,  
    Q2 INT,  
    Q3 INT,  
    Q4 VARCHAR,  
    Q5 INT  
);
```

4.2 Confirm Streaming Ingestion

After ~1 min of running Spark job, run:

```
SELECT * FROM case_table ORDER BY create_timestamp DESC LIMIT 10;  
SELECT * FROM survey_table ORDER BY survey_timestamp DESC LIMIT 10;
```

You're Done

You now have:

- Real-time data flowing from EC2 → Kinesis → EMR Spark → Redshift
- Case and Survey data written to Redshift

- Redshift ready for QuickSight or queries

```
ec2-user@ip-172-31-8-1:~      X + ~
Y', 'Q5': 9}
Sent to Kinesis: {'case_no': '600992', 'created_employee_key': '100993', 'call_center_id': 'C-109', 'status': 'Closed', 'category': 'CAT3', 'sub_cat
egory': 'SCAT9', 'communication_mode': 'Chat', 'country_cd': 'VE', 'product_code': '920539', 'last_modified_timestamp': '2025-07-30 11:33:32', 'crea
te_timestamp': '2025-07-30 11:13:32'}
Sent survey to Kinesis: {'survey_id': 'S-500029', 'case_no': '600992', 'survey_timestamp': '2025-07-30 12:03:32', 'Q1': 6, 'Q2': 1, 'Q3': 8, 'Q4': 'Y', 'Q5': 4}
Sent to Kinesis: {'case_no': '600993', 'created_employee_key': '220239', 'call_center_id': 'C-107', 'status': 'Closed', 'category': 'CAT3', 'sub_cat
egory': 'SCAT9', 'communication_mode': 'Chat', 'country_cd': 'TK', 'product_code': '5565433', 'last_modified_timestamp': '2025-07-30 11:33:32', 'cre
ate_timestamp': '2025-07-30 11:13:32'}
Sent survey to Kinesis: {'survey_id': 'S-500030', 'case_no': '600993', 'survey_timestamp': '2025-07-30 12:03:32', 'Q1': 2, 'Q2': 10, 'Q3': 10, 'Q4': 'Y', 'Q5': 2}
Sent to Kinesis: {'case_no': '600994', 'created_employee_key': '295955', 'call_center_id': 'C-108', 'status': 'Closed', 'category': 'CAT3', 'sub_cat
egory': 'SCAT9', 'communication_mode': 'Chat', 'country_cd': 'PY', 'product_code': '5586537', 'last_modified_timestamp': '2025-07-30 11:33:32', 'cre
ate_timestamp': '2025-07-30 11:13:32'}
Sent survey to Kinesis: {'survey_id': 'S-500031', 'case_no': '600994', 'survey_timestamp': '2025-07-30 12:03:32', 'Q1': 6, 'Q2': 2, 'Q3': 2, 'Q4': 'N', 'Q5': 6}
Sent to Kinesis: {'case_no': '600995', 'created_employee_key': '74582', 'call_center_id': 'C-112', 'status': 'Open', 'category': 'CAT3', 'sub_catego
ry': 'SCAT15', 'communication_mode': 'Email', 'country_cd': 'SR', 'product_code': '12949404', 'last_modified_timestamp': '2025-07-30 11:03:37', 'cre
ate_timestamp': '2025-07-30 11:03:37'}
Sent to Kinesis: {'case_no': '600996', 'created_employee_key': '288184', 'call_center_id': 'C-108', 'status': 'Open', 'category': 'CAT3', 'sub_categ
ory': 'SCAT15', 'communication_mode': 'Chat', 'country_cd': 'AO', 'product_code': '8068722', 'last_modified_timestamp': '2025-07-30 11:03:37', 'cre
ate_timestamp': '2025-07-30 11:03:37'}
Sent to Kinesis: {'case_no': '600997', 'created_employee_key': '283915', 'call_center_id': 'C-106', 'status': 'Open', 'category': 'CAT3', 'sub_categ
ory': 'SCAT15', 'communication_mode': 'Email', 'country_cd': 'BF', 'product_code': '1135204', 'last_modified_timestamp': '2025-07-30 11:03:37', 'cre
ate_timestamp': '2025-07-30 11:03:37'}
Sent to Kinesis: {'case_no': '600998', 'created_employee_key': '224604', 'call_center_id': 'C-116', 'status': 'Open', 'category': 'CAT3', 'sub_categ
ory': 'SCAT15', 'communication_mode': 'Email', 'country_cd': 'KR', 'product_code': '819430', 'last_modified_timestamp': '2025-07-30 11:03:37', 'cre
ate_timestamp': '2025-07-30 11:03:37'}
Sent to Kinesis: {'case_no': '600999', 'created_employee_key': '240604', 'call_center_id': 'C-116', 'status': 'Open', 'category': 'CAT3', 'sub_categ
ory': 'SCAT8', 'communication_mode': 'Chat', 'country_cd': 'PR', 'product_code': '9829787', 'last_modified_timestamp': '2025-07-30 11:13:42', 'creat
e_timestamp': '2025-07-30 11:13:42'}
Sent to Kinesis: {'case_no': '601000', 'created_employee_key': '215285', 'call_center_id': 'C-114', 'status': 'Open', 'category': 'CAT3', 'sub_categ
ory': 'SCAT11', 'communication_mode': 'Call', 'country_cd': 'EE', 'product_code': '12457101', 'last_modified_timestamp': '2025-07-30 11:23:47', 'cre
ate_timestamp': '2025-07-30 11:23:47'}
[ec2-user@ip-172-31-8-1 ~]$ |
```

```
25/07/30 12:01:56 INFO YarnClientSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
🔗 Redshift Connection: jdbc:redshift://tbsm-red.008673239246.ap-southeast-2.redshift-serverless.amazonaws.com:5439/tbsm-red
  User: admin
● Testing Redshift connection...
✓ Redshift connection successful!
/home/hadoop/.local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To c
ontinue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/b
logs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
🕒 Starting Kinesis data collection thread...=====
🕒 WORKER-SAFE STREAMING PIPELINE STATUS
=====

✓ Kinesis Stream: tbsm-stream
✓ Method: Background thread collection + micro-batch processing
✓ Use Events > fact_bases_historical
✓ Sum Events > fact_bases_historical
✓ Processing Interval: 15 seconds
✓ Worker Safety: No UDF dependencies
=====

🕒 Pipeline running... Press Ctrl+C to stop
=====
```

```
ec2-user@ip-172-31-8-1:~          X  Windows PowerShell          X  hadoop@ip-172-31-7-43:~  X  +  ▾
[✓] Worker Safety: No UDF dependencies
=====
[?] Pipeline running... Press Ctrl+C to stop
=====
[?] No new data, waiting...
[?] Collected 1 records from shard shardId-000000000000
[?] Processing batch 0 with 1 records...
[?] Found 1 case events
[✓] Successfully wrote 1 records to case_table
[?] Collected 2 records from shard shardId-000000000000
[?] Collected 2 records from shard shardId-000000000000
[?] Processing batch 1 with 4 records...
[?] Found 3 case events
[?] Collected 3 records from shard shardId-000000000000
[✓] Successfully wrote 3 records to case_table
[?] Found 1 survey events
[✓] Successfully wrote 1 records to survey_table
[?] Collected 2 records from shard shardId-000000000000
[?] Processing batch 2 with 5 records...
[?] Found 5 case events
[✓] Successfully wrote 5 records to case_table
[?] Collected 4 records from shard shardId-000000000000
[?] Processing batch 3 with 4 records...
[?] Found 2 case events
[✓] Successfully wrote 2 records to case_table
[?] Found 2 survey events
[✓] Successfully wrote 2 records to survey_table
[?] Collected 1 records from shard shardId-000000000000
[?] Collected 2 records from shard shardId-000000000000
[?] Processing batch 4 with 3 records...
[?] Found 3 case events
[?] Collected 1 records from shard shardId-000000000000
[✓] Successfully wrote 3 records to case_table
[?] Processing batch 5 with 1 records...
[?] Found 1 case events
[✓] Successfully wrote 1 records to case_table
[?] No new data, waiting...
[?] No new data, waiting...
[?] No new data, waiting...
^X
^C
[?] Pipeline stopped by user
[?] Stopping Kinesis collector...
[hadoop@ip-172-31-7-43 ~]$ |
```

Redshift Serverless & Analytical Queries

1. Redshift Serverless Setup

- **Workgroup Name:** tbsm-red
- **Namespace Name:** tbsm-red1
- **Database Name:** tbsm-red
- **Admin User:** admin
- **Password:** Tekavade123

IAM Role Details

- **IAM Role Name:** tbsm-redshiftfullaccessss
- **Role ARN:** arn:aws:iam::008673239246:role/tbsm-redshiftfullaccessss
- **Attached Policies:**
 - AmazonRedshiftFullAccess
 - AmazonS3FullAccess
 - AmazonKinesisFullAccess

Ensure this IAM role is attached to your Redshift workgroup via **Workgroup > Permissions > Associate IAM Role.**

2. Connect to Redshift Serverless Database

3. Create Public Schema (if not exists)

CREATE SCHEMA IF NOT EXISTS public;

4. Create Tables in public Schema

Example: Create your main tables based on schema (adjust types as per your design):

```
CREATE TABLE public.case_details (
    case_no VARCHAR(256) NOT NULL,
    status VARCHAR(256) NOT NULL,
    category VARCHAR(256) NOT NULL,
```

```
    sub_category VARCHAR(256) NOT NULL,  
    last_modified_timestamp TIMESTAMP NOT NULL,  
    create_timestamp TIMESTAMP NOT NULL,  
    created_employee_key VARCHAR(256) NOT NULL,  
    call_center_id VARCHAR(256) NOT NULL,  
    product_code VARCHAR(256) NOT NULL,  
    country_cd VARCHAR(256) NOT NULL,  
    communication_mode VARCHAR(256) NOT NULL  
);
```

```
CREATE TABLE public.dim_case_category (  
    category_key VARCHAR(256) NOT NULL,  
    sub_category_key VARCHAR(256),  
    category_description VARCHAR(512),  
    sub_category_description VARCHAR(512),  
    priority VARCHAR(256)  
);
```

```
CREATE TABLE public.survey_details (  
    survey_id VARCHAR(256) NOT NULL,  
    case_no VARCHAR(256) NOT NULL,  
    survey_timestamp TIMESTAMP NOT NULL,  
    Q1 INT,  
    Q2 INT,  
    Q3 INT,  
    Q4 VARCHAR(256),  
    Q5 INT  
);
```

The screenshot shows the AWS Redshift query editor interface. On the left, the sidebar includes 'Editor', 'Queries', 'Notebooks', 'Charts', 'History', and 'Scheduled queries'. The main area displays a 'Redshift query editor v2' window. A tree view on the left shows database structures: dev, sample_data_dev, tbsm-red, public, and Tables. Under Tables, 'case_details' is selected. Below it, a table structure is shown with columns: case_no, status, category, sub_category, and last_modified_timestamp. The distribution key is listed as 'created_employee_key'. To the right, a code editor contains the SQL script for creating the table and loading data from S3. The code includes comments for creating the table, defining columns with specific data types and constraints (NN, Izo), and specifying the distribution key. It also includes a COPY command to load data from an S3 bucket, specifying the IAM role and format as PARQUET.

```

18 -- FORMAT AS PARQUET;
19
20
21
22 -- Create survey details table
23 CREATE TABLE public.futurecart_case_survey_details (
24     survey_id VARCHAR(50),
25     case_no VARCHAR(50),
26     survey_timestamp VARCHAR(50),
27     q1 VARCHAR(10),
28     q2 VARCHAR(10),
29     q3 VARCHAR(10),
30     q4 VARCHAR(10),
31     q5 VARCHAR(10)
32 );
33
34 -- Load survey details data from S3
35 COPY public.futurecart_case_survey_details
36 FROM 's3://tbsm-encore/historical/surveys_historical/'
37 IAM_ROLE 'arn:aws:iam::008673239246:role/tbsm-redshiftfullaccessss'
38 FORMAT AS PARQUET;
39
40 SELECT COUNT(*) FROM public.futurecart_case_details;
41 SELECT COUNT(*) FROM public.futurecart_case_survey_details;
42
43 CREATE EXTERNAL SCHEMA IF NOT EXISTS public

```

5. Upload Data from S3 to Redshift

COPY Command to Load Data into Redshift Tables

Example COPY command for CSV data:

COPY public.case_details

FROM 's3://tbsm-encore/dimensional/casecases_historical/'

IAM_ROLE 'arn:aws:iam::008673239246:role/tbsm-redshiftfullaccessss'

FORMAT AS PARQUET;

Repeat for each table:

COPY public.dim_case_category

FROM 's3://tbsm-encore/dimensional/dim_case_category/'

IAM_ROLE 'arn:aws:iam::008673239246:role/tbsm-redshiftfullaccessss'

FORMAT AS PARQUET;

COPY public.survey_details

FROM 's3://tbsm-encore/dimensional/surveysurveys_historical/'

IAM_ROLE 'arn:aws:iam::008673239246:role/tbsm-redshiftfullaccessss'

FORMAT AS PARQUET;

6. Verify Data Upload

```
SELECT COUNT(*) FROM public.case_details;
```

```
SELECT COUNT(*) FROM public.dim_case_category;
```

```
SELECT COUNT(*) FROM public.survey_details;
```

The screenshot shows the Redshift query editor interface. The left sidebar displays the navigation pane with 'Queries' selected, showing three serverless environments: 'zaur-wg', 'tan-wg', and 'tbsm-red'. The 'tbsm-red' environment is expanded, showing its native databases ('dev', 'sample_data_dev', 'tbsm-red') and tables ('case_details', 'dim_calendar', etc.). The main area contains a code editor with the following SQL script:

```
61 CREATE EXTERNAL SCHEMA IF NOT EXISTS tbsm_dim_ext
62   FROM DATA CATALOG
63   DATABASE 'tbsm-dim'
64   IAM_ROLE 'arn:aws:iam::008673239246:role/tbsm-redshiftfullaccessss'
65   REGION 'ap-southeast-2';
66
67
68
69
70 SELECT * FROM public.dim_calendar LIMIT 10;
71 SELECT * FROM public.dim_call_center LIMIT 10;
72
73
74
```

Below the code editor is a results panel titled 'Result 1 (10)'. It displays a table with columns: calendar_date, date_desc, week_day_nbr, week_number, and week_name. The data is as follows:

calendar_date	date_desc	week_day_nbr	week_number	week_name
2005-11-26	Saturday, November 26, 2005	1	44	Week 44
2005-11-27	Sunday, November 27, 2005	2	44	Week 44
2005-11-28	Monday, November 28, 2005	3	44	Week 44
2005-11-29	Tuesday, November 29, 2005	4	44	Week 44
2005-11-30	Wednesday, November 30, 2005	5	44	Week 44
2005-12-01	Thursday, December 01, 2005	6	44	Week 44
2005-12-02	Friday, December 02, 2005	7	44	Week 44
2005-12-03	Saturday, December 03, 2005	1	45	Week 45
2005-12-04	Sunday, December 04, 2005	2	45	Week 45

At the bottom of the results panel, it says 'Query ID 11111 Elapsed time: 3762 ms Total rows: 10'.

The screenshot shows the Redshift query editor interface. The left sidebar displays the navigation pane with 'Queries' selected, showing three serverless environments: 'zaur-wg', 'tan-wg', and 'tbsm-red'. The 'tbsm-red' environment is expanded, showing its native databases ('dev', 'sample_data_dev', 'tbsm-red') and tables ('case_details', 'dim_calendar', etc.). The main area contains a code editor with the following SQL script:

```
58   ) WITH NO DATA;
59
60
61 CREATE EXTERNAL SCHEMA IF NOT EXISTS tbsm_dim_ext
62   FROM DATA CATALOG
63   DATABASE 'tbsm-dim'
64   IAM_ROLE 'arn:aws:iam::008673239246:role/tbsm-redshiftfullaccessss'
65   REGION 'ap-southeast-2';
66
67
68
69
70 SELECT * FROM public.case_details LIMIT 10;
71 SELECT * FROM public.dim_call_center LIMIT 10;
```

Below the code editor is a results panel titled 'Result 1 (10)'. It displays a table with columns: case_no, status, category, sub_category, and last_modified_timestamp. The data is as follows:

case_no	status	category	sub_category	last_modified_timestamp
658751	Closed	CAT4	SCAT2	2025-07-20 04:30:04
637817	Open	CAT5	SCAT10	2025-07-28 04:30:04
668797	Closed	CAT5	SCAT3	2025-07-21 04:30:04
676578	Closed	CAT2	SCAT7	2025-07-26 04:30:04
674992	Closed	CAT5	SCAT18	2025-07-26 04:30:04
636100	Closed	CAT5	SCAT14	2025-07-25 04:30:04
669223	Closed	CAT4	SCAT14	2025-07-25 04:30:04
648284	Open	CAT1	SCAT4	2025-07-27 04:30:04
663626	Closed	CAT3	SCAT17	2025-07-27 04:30:04

At the bottom of the results panel, it says 'Query ID 11143 Elapsed time: 3770 ms Total rows: 10'.

7. Run Analytical Queries

-- 1. Total number of cases

```
SELECT COUNT(*) AS total_cases
```

```
FROM public.case_details;
```

	Result 1 (1)	Result 2 (1)
	total_cases	
	1000	

-- 2. Total open cases in the last 1 hour

```
SELECT COUNT(*) AS open_cases_last_1hr
FROM public.case_details
WHERE status = 'Open'
AND create_timestamp >= DATEADD(hour, -1, GETDATE());
```

1	SELECT COUNT(*) AS open_cases FROM cases	
2	WHERE status = 'Open'	
3	AND DATEDIFF (minute ,CAST(create_timestamp as timestamp), CAST(last_modified_timestamp AS timestamp)) <=60 ;	
	Result 1 (1)	
	open cases	
	510	

-- 3. Total closed cases in the last 1 hour

```
SELECT COUNT(*) AS closed_cases_last_1hr
FROM public.case_details
WHERE status = 'Closed'
AND last_modified_timestamp >= DATEADD(hour, -1, GETDATE());
```

▶ Run	Run	Limit 100	Explain	Isolated session	Serverless: pri...	dev	
1	SELECT COUNT(*) AS open_cases FROM cases						
2	WHERE status = 'Closed'						
3	AND DATEDIFF (minute ,CAST(create_timestamp as timestamp), CAST(last_modified_timestamp AS timestamp)) <=60 ;						
		Result 1 (1)					
		open cases					
		490					

-- 4. Total priority cases (join with category)

```
SELECT COUNT(*) AS total_priority_cases
FROM public.case_details cd
JOIN public.dim_case_category ccat
ON cd.category = ccat.category_key
```

```
WHERE ccat.priority IS NOT NULL;
```

Result 1 (1)	Result 2 (1)	Result 3 (1)	Result 4 (1)
total_priority_cases			
5077			

```
-- 5. Total positive and negative survey responses in the last 1 hour
```

```
SELECT
```

```
    SUM(  
        CASE WHEN Q1 BETWEEN 8 AND 10 THEN 1 ELSE 0 END +  
        CASE WHEN Q2 BETWEEN 8 AND 10 THEN 1 ELSE 0 END +  
        CASE WHEN Q3 BETWEEN 8 AND 10 THEN 1 ELSE 0 END +  
        CASE WHEN Q5 BETWEEN 8 AND 10 THEN 1 ELSE 0 END  
) AS total_positive,
```

```
    SUM(  
        CASE WHEN Q1 BETWEEN 1 AND 4 THEN 1 ELSE 0 END +  
        CASE WHEN Q2 BETWEEN 1 AND 4 THEN 1 ELSE 0 END +  
        CASE WHEN Q3 BETWEEN 1 AND 4 THEN 1 ELSE 0 END +  
        CASE WHEN Q5 BETWEEN 1 AND 4 THEN 1 ELSE 0 END  
) AS total_negative
```

```
FROM public.survey_details
```

```
WHERE survey_timestamp >= DATEADD(hour, -1, GETDATE());
```

```
-- 6. Total number of surveys in the last 1 hour
```

```
SELECT COUNT(*) AS total_surveys_last_1hr
```

```
FROM public.survey_details
```

```
WHERE survey_timestamp >= DATEADD(hour, -1, GETDATE());
```

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

total_surveys
240

-- 7. Total open cases by day

```
SELECT
DATE_TRUNC('day', create_timestamp) AS period,
COUNT(*) AS open_cases
FROM public.case_details
WHERE status = 'Open'
GROUP BY 1
ORDER BY 1 DESC;
```

period	open_cases
0035-01-15 00:00:00+00	48
0034-01-15 00:00:00+00	55
0033-01-14 00:00:00+00	54
0032-01-15 00:00:00+00	51
0031-01-15 00:00:00+00	47
0030-01-15 00:00:00+00	49
0029-01-14 00:00:00+00	49

-- 8. Total closed cases by day

```
SELECT
DATE_TRUNC('day', last_modified_timestamp) AS period,
COUNT(*) AS closed_cases
FROM public.case_details
WHERE status = 'Closed'
```

GROUP BY 1

ORDER BY 1 DESC;

I)	Result 5 (1)	Result 6 (1)	Result 7 (10)	Result 8 (10)
	period	closed_cases		
	0035-01-15 00:00:00+00	52		
	0034-01-15 00:00:00+00	45		
	0033-01-14 00:00:00+00	46		
	0032-01-15 00:00:00+00	49		
	0031-01-15 00:00:00+00	53		
	0030-01-15 00:00:00+00	51		
	0029-01-14 00:00:00+00	51		

-- 9. Total positive and negative survey responses by day

SELECT

```
DATE_TRUNC('day', survey_timestamp) AS period,  
SUM(  
    CASE WHEN Q1 BETWEEN 8 AND 10 THEN 1 ELSE 0 END +  
    CASE WHEN Q2 BETWEEN 8 AND 10 THEN 1 ELSE 0 END +  
    CASE WHEN Q3 BETWEEN 8 AND 10 THEN 1 ELSE 0 END +  
    CASE WHEN Q5 BETWEEN 8 AND 10 THEN 1 ELSE 0 END  
) AS total_positive,  
SUM(  
    CASE WHEN Q1 BETWEEN 1 AND 4 THEN 1 ELSE 0 END +  
    CASE WHEN Q2 BETWEEN 1 AND 4 THEN 1 ELSE 0 END +  
    CASE WHEN Q3 BETWEEN 1 AND 4 THEN 1 ELSE 0 END +  
    CASE WHEN Q5 BETWEEN 1 AND 4 THEN 1 ELSE 0 END  
) AS total_negative  
FROM public.survey_details  
GROUP BY 1  
ORDER BY 1 DESC;
```

The screenshot shows a PostgreSQL query interface with multiple result tabs. The current tab is 'Result 9 (10)'. The data is presented in a table with three columns: 'period', 'total_positive', and 'total_negative'. The data rows are:

	period	total_positive	total_negative
	0035-01-15 00:00:00+00	99	118
	0034-01-15 00:00:00+00	100	129
	0033-01-14 00:00:00+00	98	125
	0032-01-15 00:00:00+00	100	129
	0031-01-15 00:00:00+00	107	118
	0030-01-15 00:00:00+00	91	130
	0029-01-14 00:00:00+00	94	128

Query ID: [REDACTED]

-- 10. Total number of surveys by day

SELECT

```
DATE_TRUNC('day', survey_timestamp) AS period,
```

```
COUNT(*) AS total_surveys
```

```
FROM public.survey_details
```

```
GROUP BY 1
```

```
ORDER BY 1 DESC;
```

The screenshot shows a PostgreSQL query interface with multiple result tabs. The current tab is 'Result 10 (10)'. The data is presented in a table with two columns: 'period' and 'total_surveys'. The data rows are:

	period	total_surveys
	0035-01-15 00:00:00+00	80
	0034-01-15 00:00:00+00	80
	0033-01-14 00:00:00+00	80
	0032-01-15 00:00:00+00	80
	0031-01-15 00:00:00+00	80
	0030-01-15 00:00:00+00	80
	0029-01-14 00:00:00+00	80

-- 11. Total open and closed cases out of all cases

SELECT

```
COUNT(*) AS total_cases,
```

```
SUM(CASE WHEN status = 'Open' THEN 1 ELSE 0 END) AS total_open,
```

```
SUM(CASE WHEN status = 'Closed' THEN 1 ELSE 0 END) AS total_closed
```

```
FROM public.case_details;
```

	total_cases	total_open	total_closed
	1000	512	488

-- 12. Total number of cases received by country

```

SELECT
country_cd,
COUNT(*) AS total_cases
FROM public.case_details
GROUP BY country_cd
ORDER BY total_cases DESC;
```

	country_cd	total_cases
	US	206
	AU	203
	IN	199
	BR	197
	DE	195

THANK YOU!

-if any doubt email me- yashtekavade@gmail.com