

PROJECT-1 FutureCart Documentation

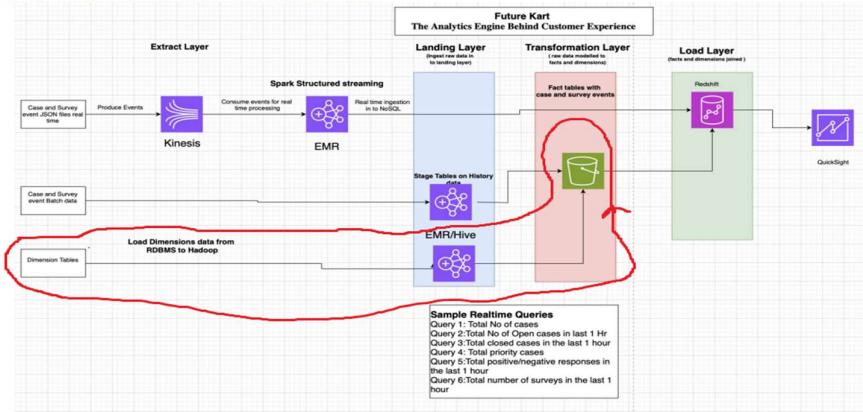
Name: Khushi Baurasi

Data Engineer Bootcamp 2025

Date: 31 July 2025

Historical data for Dimension Tables Step-by-Step -

3. Data Flow Architecture/Process Flow



1. Launch EC2 Instance

- Launched an **Amazon EC2 instance** (Amazon Linux 2) for initial setup.
- Configured **security groups** to allow inbound SSH (port 22) and MySQL (port 3306).
- Connected to the instance using:

```
ssh -i "khushi-seoul".pem ec2-user@ ec2-3-35-79-218.ap-northeast-2.compute.amazonaws.com
```

2. Install MySQL Server

- Installed MySQL on the EC2 instance:

```
sudo yum update -y
```

```
sudo yum install mysql-server -y  
sudo systemctl start mysqld  
sudo mysql -u root -p
```

3. Create MySQL Database and Tables

```
CREATE DATABASE futurecart_crm;
```

```
USE futurecart_crm;
```

- Created 8 dimension tables:

1. 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)  
);
```

2. futurecart_call_center_details

```
CREATE TABLE futurecart_call_center_details (  
    call_center_id INT PRIMARY KEY,  
    call_center_name VARCHAR(100),  
    location VARCHAR(100),
```

```
    manager_name VARCHAR(100),  
    contact_number VARCHAR(20),  
    opened_date DATE,  
    is_active BOOLEAN  
);
```

3. futurecart_case_category_details

```
sql  
CopyEdit  
CREATE TABLE futurecart_case_category_details (  
    category_id INT PRIMARY KEY,  
    category_name VARCHAR(100),  
    description TEXT,  
    parent_category_id INT  
);
```

4. futurecart_case_country_details

```
sql  
CopyEdit  
CREATE TABLE futurecart_case_country_details (  
    country_code CHAR(2) PRIMARY KEY,  
    country_name VARCHAR(100),  
    region VARCHAR(100),  
    language VARCHAR(50)  
);
```

5. futurecart_case_priority_details

```
CREATE TABLE futurecart_case_priority_details (
```

```
priority_id INT PRIMARY KEY,  
priority_level VARCHAR(50),  
description TEXT,  
escalation_time_hours INT  
);
```

6. futurecart_employee_details

```
CREATE TABLE futurecart_employee_details (  
employee_id INT PRIMARY KEY,  
first_name VARCHAR(50),  
last_name VARCHAR(50),  
department VARCHAR(100),  
designation VARCHAR(100),  
email VARCHAR(100),  
contact_number VARCHAR(20),  
hire_date DATE,  
is_active BOOLEAN  
);
```

7. futurecart_product_details

```
CREATE TABLE futurecart_product_details (  
product_id INT PRIMARY KEY,  
product_name VARCHAR(100),  
category VARCHAR(100),  
brand VARCHAR(100),  
price DECIMAL(10,2),  
launch_date DATE,  
is_active BOOLEAN
```

```
);
```

8. futurecart_survey_question_details

sql

CopyEdit

```
CREATE TABLE futurecart_survey_question_details (
    question_id INT PRIMARY KEY,
    question_text TEXT,
    question_type VARCHAR(50), -- e.g., multiple-choice, rating, open-ended
    is_mandatory BOOLEAN,
    category VARCHAR(50)
);
```

The screenshot shows a MySQL terminal window. The terminal output is as follows:

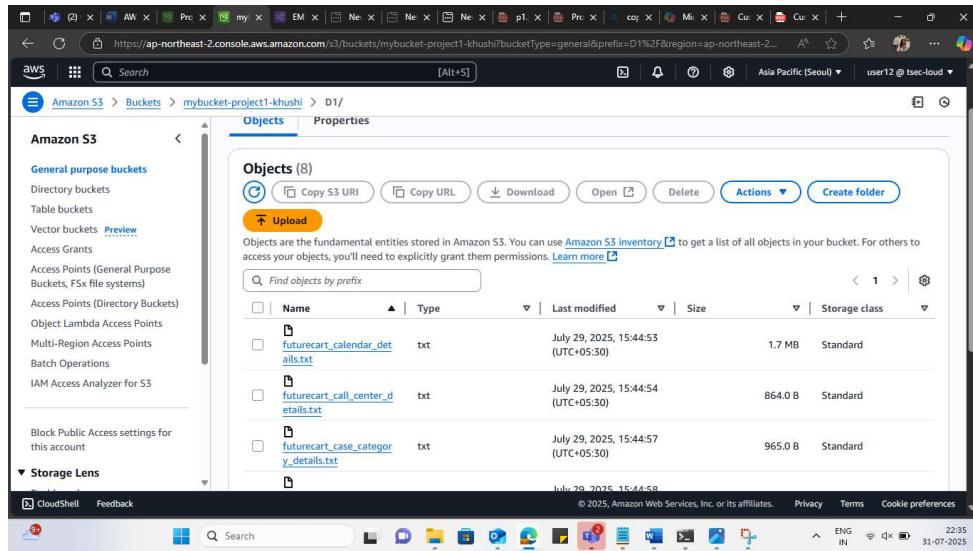
```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_futurecart_crm |
+-----+
| futurecart_calendar_details |
| futurecart_call_center_details |
| futurecart_case_category_details |
| futurecart_case_country_details |
| futurecart_case_priority_details |
| futurecart_employee_details |
| futurecart_product_details |
| futurecart_survey_question_details |
+-----+
8 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM each_table_name;
```

The terminal is part of a larger interface with a sidebar containing icons for Calend, Yammer, Ambie, InfoAs, ..., and Apps. The status bar at the bottom shows the date as 29-07-2025, the time as 18:33, and system information like ENG IN.

4. Upload Input Files to S3

- Uploaded .txt files provided by instructor to my S3 Bucket



5. Move Files from S3 to EC2

- Connected to EC2 via CLI.
- Downloaded .txt files from S3 to EC2 instance:

```
scp -i khushi-seoul.pem futurecart_calendar_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

```
scp -i khushi-seoul.pem futurecart_call_center_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

```
scp -i khushi-seoul.pem futurecart_case_category_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

```
scp -i khushi-seoul.pem futurecart_case_country_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

```
scp -i khushi-seoul.pem futurecart_case_priority_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

```
scp -i khushi-seoul.pem futurecart_employee_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

```
scp -i khushi-seoul.pem futurecart_product_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

```
scp -i khushi-seoul.pem futurecart_survey_question_details.txt ec2-user@ec2-3-36-70-214.ap-northeast-2.compute.amazonaws.com:~
```

6. Load Data into MySQL

- Loaded each .txt file into the corresponding MySQL table using:

```
USE futurecart_crm;
```

```
LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_calendar_details.txt'  
INTO TABLE futurecart_calendar_details  
FIELDS TERMINATED BY '\t'  
LINES TERMINATED BY '\n';  
IGNORE 1 LINES;
```

```
LOAD DATA LOCAL INFILE '/home/ec2-user/futurecart_calendar_details.txt'  
INTO TABLE futurecart_calendar_details  
FIELDS TERMINATED BY '\t'  
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 '\t'  
LINES TERMINATED BY '\n'
```

```
IGNORE 1 LINES;
```

```
TRUNCATE TABLE futurecart_case_country_details;
```

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

```
INTO TABLE futurecart_case_country_details
```

```
FIELDS TERMINATED BY '\t'
```

```
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 '\t'  
LINES TERMINATED BY '\n'  
IGNORE 1 LINES;
```

7. Launch Amazon EMR Cluster

- Launched a new **EMR Cluster** with the following:
 - Applications: Hadoop, Hive, Spark
 - Cluster name: futurecart-emr-cluster-khushi
 - Created new EC2 instance for CLI access.
-

8. Connect to EMR Master Node

- Used EMR master node's SSH endpoint to connect via CLI:

```
ssh -i "khushi-seoul.pem" ec2-user@ec2-3-35-79-218.ap-northeast-  
2.compute.amazonaws.com
```

9. Install JDBC Driver and Set Up Hive

- Downloaded **MySQL JDBC driver**:

```
Sudo wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-  
8.0.33.tar.gz
```

```
Sudo tar -xvzf mysql-connector-java-8.0.33.tar.gz
```

```
cp mysql-connector-java-8.0.33/mysql-connector-java-8.0.33.jar /usr/lib/hive/lib/
```

10. Import Data from MySQL to Hive

- Created Hive tables matching MySQL schema.
- Imported MySQL data into Hive using:

```

2025-07-29 13:02:34,194 INFO tez.TezSessionState: Attempting to clean up resources for efc795c9-1573-48c5-9403-6e15ad4c5824: hdfs://ip-172-31-50-148.ap-northeast-2.compute.internal:8020/tmp/hive/root/_tez_session_dir/efc795c9-1573-48c5-9403-6e15ad4c5824/resources; 0 additional files, 0 localized resources
2025-07-29 13:02:34,201 INFO session.SessionState: Deleted directory: /tmp/hive/root/efc795c9-1573-48c5-9403-6e15ad4c5824 on fs with scheme file
2025-07-29 13:02:34,205 INFO session.SessionState: Deleted directory: /tmp/root/efc795c9-1573-48c5-9403-6e15ad4c5824 on fs with scheme file
[ec2-user@ip-172-31-50-140 ~]$ hive
Hive Session ID = 7e66ac2b-0cc8-4485-b8dc-f3fc0c2679a6
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: 0.829 seconds, Fetched: 8 row(s)
hive> select * from futurecart_case_country_details limit 5;
OK
4      Afghanistan
8      Albania
12     Algeria
20     Andorra
24     Angola
Time taken: 1.358 seconds, Fetched: 5 row(s)
hive> |

```

11. Export Hive Data to S3 Using Spark

- Wrote and ran a **PySpark job** to export Hive tables to CSV format in S3:

```

df = spark.sql("SELECT * FROM futurecart_call_center_details")

df.write.option("header", "true").csv("s3:// mybucket-project1-khushi /
futurecart_call_center_details /")

```

- Repeated for each Hive table all rest 7 tables data.

Status

- EC2, MySQL, and S3 configured
- All dimension data uploaded from .txt → MySQL → Hive → S3
- EMR and Spark setup complete
- Hive to S3 export via Spark working with CSV output

Amazon S3 > Buckets > mybucket-project1-khushi

Objects (11)

Name	Type	Last modified	Size	Storage class
futurecart_calendar_details/	Folder			
futurecart_call_center_details/	Folder			
futurecart_case_category_details/	Folder			
futurecart_case_country_details/	Folder			
_SUCCESS	CSV	July 29, 2025, 18:45:26 (UTC+05:30)	0 B	Standard
part-00000-6896f85d-5c11-41d3-80ff-26a9090bda6c-c000.csv	CSV	July 31, 2025, 13:15:01 (UTC+05:30)	1.8 MB	Standard

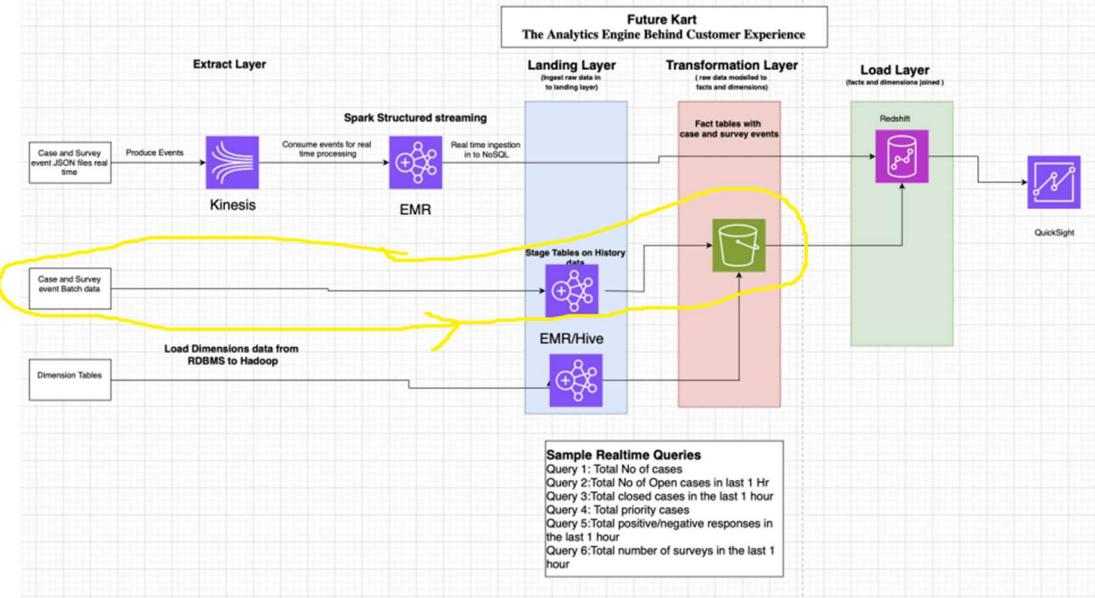
Amazon S3 > Buckets > mybucket-project1-khushi > futurecart_calendar_details/

Objects (2)

Name	Type	Last modified	Size	Storage class
_SUCCESS	CSV	July 29, 2025, 18:45:26 (UTC+05:30)	0 B	Standard
part-00000-6896f85d-5c11-41d3-80ff-26a9090bda6c-c000.csv	CSV	July 31, 2025, 13:15:01 (UTC+05:30)	1.8 MB	Standard

Historical Batch Data Generation & Ingestion Workflow -

3. Data Flow Architecture/Process Flow



1. Batch Data Script Uploaded to S3

- Uploaded your script generate_historical_data.py to S3.

2. Historical Data Generation

- Script generated:
 - case_data_day1.json to case_data_day10.json under historical_data_jsonl/cases/
 - survey_data_day1.json to survey_data_day10.json under historical_data_jsonl/surveys/

3. Uploaded JSON Files to S3

- Copied the generated JSON files to your S3 bucket, e.g.:

s3://mybucket-project1-khushi/historical_data_jsonl/cases/

s3://mybucket-project1-khushi/historical_data_jsonl/surveys/

The screenshot shows the AWS S3 console interface. The left sidebar lists 'General purpose buckets' including 'Directory buckets', 'Table buckets', 'Vector buckets', 'Access Grants', 'Access Points (General Purpose Buckets, FSx file systems)', 'Access Points (Directory Buckets)', 'Object Lambda Access Points', 'Multi-Region Access Points', 'Batch Operations', and 'IAM Access Analyzer for S3'. Below this is a section for 'Block Public Access settings for this account' and a 'Storage Lens' section.

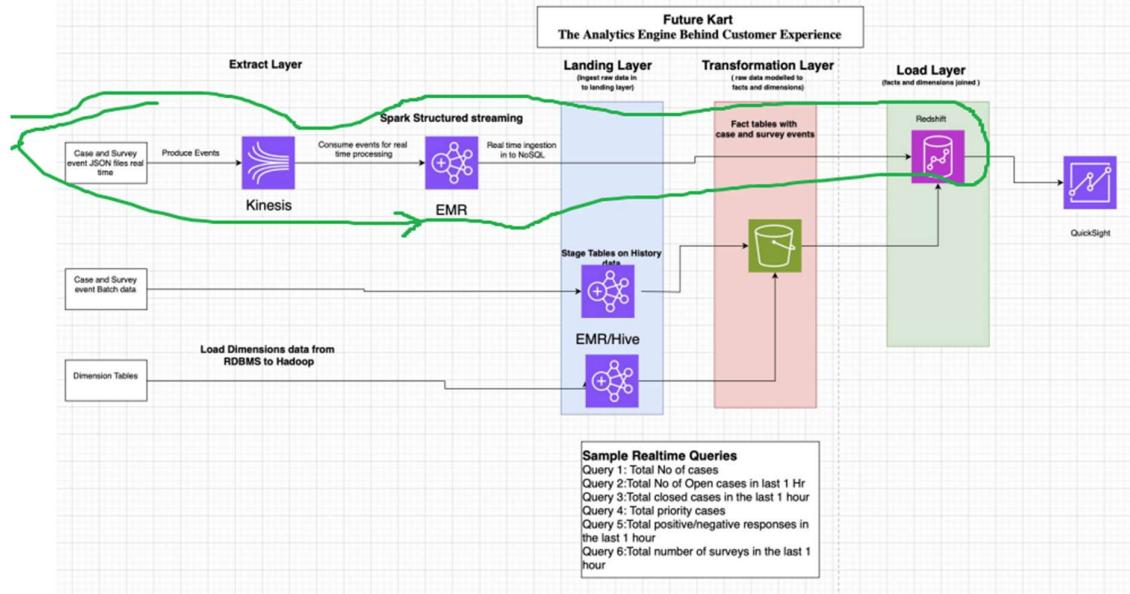
The main content area is titled 'historical_data_jsonl/'. It has tabs for 'Objects' (selected) and 'Properties'. A 'Copy S3 URI' button is at the top right. Below is a toolbar with 'Upload' (highlighted in orange), 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Actions', and 'Create folder'. A search bar says 'Find objects by prefix'. A table lists two objects:

Name	Type	Last modified	Size	Storage class
cases/	Folder	-	-	-
surveys/	Folder	-	-	-

At the bottom, there's a footer with links for 'CloudShell', 'Feedback', and 'Cookie preferences', along with copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates.' and language/region settings: 'ENG IN 20:41 31-07-2025'.

Real Time Data : Generate Real-Time JSON Events

3. Data Flow Architecture/Process Flow



Script used: stream_to_kinesis.py

```
nano stream_to_kinesis.py
```

```
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-2')
stream_name = 'case-survey-stream-Khushi-P1'

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

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_categories = ['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_categories)
    number_of_cases = random.choice(number_of_cases_counts)
    cases = all_case_data_lines[:i + number_of_cases]

    current_timestamp = datetime.now()
    case_created_ts = str(current_timestamp - timedelta(minutes=random.choice(open_case_time_diff_mins)))[-19]
    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())

    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)

    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)

```

pip3 install boto3

python3 stream_to_kinesis.py

```

d operation: Stream case-survey-stream-Khushi-P1 under account 008673239246 not found.
[ec2-user@ip-172-31-46-186 ~]$ python3 stream_to_kinesis.py
Sent to Kinesis: {'case_no': '600901', 'created_employee_key': '263886', 'call_center_id': 'C-123', 'status': 'Closed', 'category': 'CAT3', 'sub_category': 'SCAT15', 'communication_mode': 'Chat', 'country_cd': 'PN', 'product_code': '13164066', 'last_modified_timestamp': '2025-07-31 04:52:11', 'create_timestamp': '2025-07-31 04:12:11'}
Sent survey to Kinesis: {'survey_id': 'S-500000', 'case_no': '600901', 'survey_timestamp': '2025-07-31 05:02:11', 'Q1': 1, 'Q2': 1, 'Q3': 9, 'Q4': 'Y', 'Q5': 10}
Sent to Kinesis: {'case_no': '600902', 'created_employee_key': '406314', 'call_center_id': 'C-125', 'status': 'Closed', 'category': 'CAT3', 'sub_category': 'SCAT15', 'communication_mode': 'Call', 'country_cd': 'TF', 'product_code': '10149940', 'last_modified_timestamp': '2025-07-31 04:52:11', 'create_timestamp': '2025-07-31 04:12:11'}
Sent survey to Kinesis: {'survey_id': 'S-500001', 'case_no': '600902', 'survey_timestamp': '2025-07-31 05:02:11', 'Q1': 4, 'Q2': 2, 'Q3': 2, 'Q4': 'N', 'Q5': 1}
Sent to Kinesis: {'case_no': '600903', 'created_employee_key': '401883', 'call_center_id': 'C-119', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT8', 'communication_mode': 'Call', 'country_cd': 'BO', 'product_code': '2254829', 'last_modified_timestamp': '2025-07-31 04:12:16', 'create_timestamp': '2025-07-31 04:12:16'}
Sent to Kinesis: {'case_no': '600904', 'created_employee_key': '39803', 'call_center_id': 'C-116', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT8', 'communication_mode': 'Email', 'country_cd': 'GL', 'product_code': '1895883', 'last_modified_timestamp': '2025-07-31 04:12:16', 'create_timestamp': '2025-07-31 04:12:16'}
Sent to Kinesis: {'case_no': '600905', 'created_employee_key': '253642', 'call_center_id': 'C-122', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT8', 'communication_mode': 'Email', 'country_cd': 'SN', 'product_code': '9527413', 'last_modified_timestamp': '2025-07-31 04:12:16', 'create_timestamp': '2025-07-31 04:12:16'}
Sent to Kinesis: {'case_no': '600906', 'created_employee_key': '426173', 'call_center_id': 'C-121', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT8', 'communication_mode': 'Chat', 'country_cd': 'BQ', 'product_code': '644275', 'last_modified_timestamp': '2025-07-31 04:12:16', 'create_timestamp': '2025-07-31 04:12:16'}
Sent to Kinesis: {'case_no': '600907', 'created_employee_key': '446122', 'call_center_id': 'C-103', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT8', 'communication_mode': 'Call', 'country_cd': 'TG', 'product_code': '131972', 'last_modified_timestamp': '2025-07-31 04:12:16', 'create_timestamp': '2025-07-31 04:12:16'}
Sent to Kinesis: {'case_no': '600908', 'created_employee_key': '440062', 'call_center_id': 'C-105', 'status': 'Open', 'category': 'CAT3', 'sub_category': 'SCAT12', 'communication_mode': 'Call', 'country_cd': 'EC', 'product_code': '852698', 'last_modified_timestamp': '2025-07-31 04:12:21', 'create_timestamp': '2025-07-31 04:12:21'}

```

Step 2: Kinesis Data Stream (Input Source for Streaming)

You created a stream named:

case-survey-stream-Khushi-P1

Amazon Kinesis > Data streams > case-survey-stream-Khushi-P1

case-survey-stream-Khushi-P1 Info

Data stream summary

Status: Active | Capacity mode: On-demand | Data retention period: 1 day | ARN: arn:aws:kinesis:ap-northeast-2:000673239246:stream/case-survey-stream-Khushi-P1 | Creation time: July 31, 2025 at 10:31 GMT+5:30

Monitoring Applications Configuration Enhanced fan-out (0) Data viewer Data analytics - new

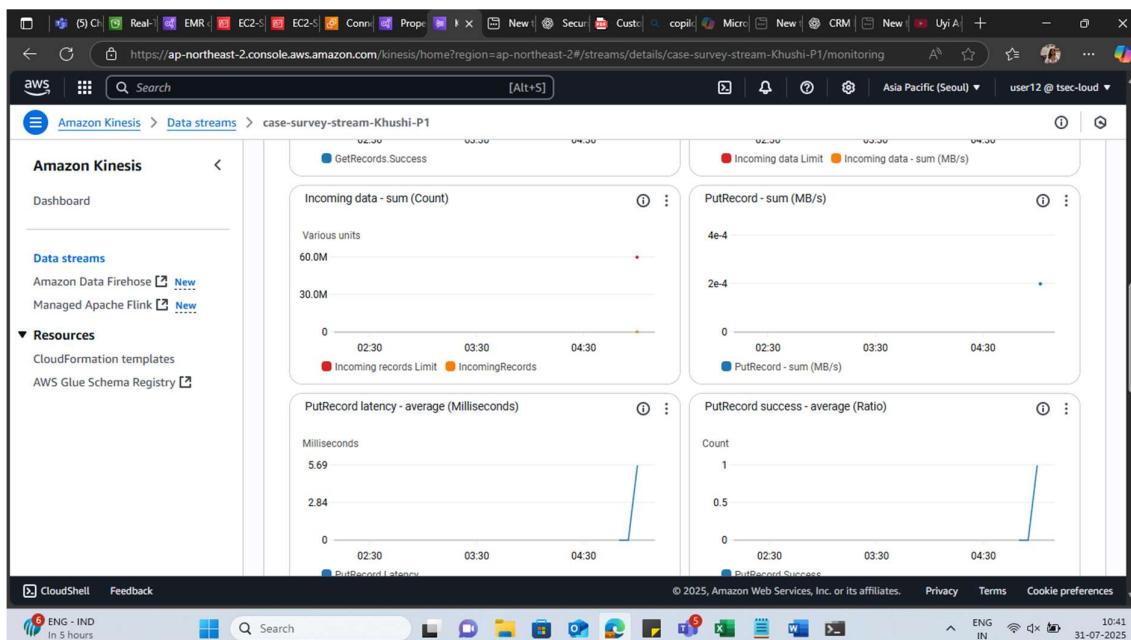
Stream metrics Info

GetRecords.sum (MB/s) | GetRecords.iterator.age.max (Milliseconds)

GetRecords.sum (MB/s): 400 | GetRecords.iterator.age.max (Milliseconds): 1

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 14:20 31-07-2025

This stream receives the real-time case/survey events from the EC2 script.



Step 3: Spark Streaming Job on EMR

nano kinesis_to_redshift.py

```

#!/usr/bin/env python3
"""
FutureKart Real-time Data Pipeline - Worker-Safe Streaming Version
=====

This version provides streaming without UDF dependencies that cause issues
in distributed Spark environments. It uses a micro-batch approach with
direct DataFrame operations for maximum compatibility.

USAGE:
-----
spark-submit \
--packages "com.amazon.redshift:redshift-jdbc42:2.1.0.9" \
kinesis_to_redshift_worker_safe.py

Features:
- No UDF dependencies (worker-safe)
- Uses micro-batch processing with direct Kinesis calls
- True structured streaming with writeStream
- Built-in error handling and recovery
- Works in any Spark environment

Author: FutureKart CRM Data Pipeline
Date: July 2025
"""

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

def create_spark_session():
    """Create optimized Spark session for streaming"""
    return SparkSession.builder \
        .appName("FutureKart-Worker-Safe-Pipeline") \
        .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 get_schemas():
    """Define precise schemas for case and survey events"""

    case_schema = StructType([
        StructField("case_no", StringType(), True),
        StructField("created_employee_key", StringType(), True),
        StructField("call_center_id", StringType(), True),
        StructField("status", StringType(), True),
        StructField("category", StringType(), True),
        StructField("sub_category", StringType(), True),
        StructField("communication_mode", StringType(), True),
        StructField("country_cd", StringType(), True),
        StructField("product_code", StringType(), True),
        StructField("last_modified_timestamp", StringType(), True),
    ])

```

```

        StructField("create_timestamp", StringType(), True)
    ])

survey_schema = StructType([
    StructField("survey_id", StringType(), True),
    StructField("case_no", StringType(), True),
    StructField("survey_timestamp", StringType(), True),
    StructField("Q1", IntegerType(), True),
    StructField("Q2", IntegerType(), True),
    StructField("Q3", IntegerType(), True),
    StructField("Q4", StringType(), True),
    StructField("Q5", IntegerType(), True)
])

return case_schema, survey_schema

class KinesisDataCollector:
    """Thread-safe Kinesis data collector"""

    def __init__(self, stream_name, region):
        self.stream_name = stream_name
        self.region = region
        self.kinesis_client = boto3.client('kinesis', region_name=region)
        self.data_queue = Queue()
        self.running = False
        self.shard_iterators = {}

    def start_collection(self):
        """Start collecting data from Kinesis"""
        self.running = True
        thread = threading.Thread(target=self._collect_data)
        thread.daemon = True
        thread.start()
        return thread

    def stop_collection(self):
        """Stop collecting data"""
        self.running = False

    def get_batch_data(self):
        """Get a batch of collected data"""
        data = []
        while not self.data_queue.empty() and len(data) < 100:
            data.append(self.data_queue.get())
        return data

    def _collect_data(self):
        """Background thread to collect data from Kinesis"""
        print("⌚ Starting Kinesis data collection thread...")

        while self.running:
            try:
                # Get stream description
                stream_desc = self.kinesis_client.describe_stream(StreamName=self.stream_name)
                shards = stream_desc['StreamDescription']['Shards']

                for shard in shards[:1]: # Process first shard for now
                    shard_id = shard['ShardId']

                    # Get or create shard iterator
                    if shard_id not in self.shard_iterators:

```

```

response = self.kinesis_client.get_shard_iterator(
    StreamName=self.stream_name,
    ShardId=shard_id,
    ShardIteratorType='LATEST'
)
self.shard_iterators[shard_id] = response['ShardIterator']

# Get records
try:
    response = self.kinesis_client.get_records(
        ShardIterator=self.shard_iterators[shard_id],
        Limit=50
    )

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

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

            record_data = {
                "json_data": str(data),
                "partitionKey": record.get('PartitionKey', ''),
                "sequenceNumber": record.get('SequenceNumber', ''),
                "approximateArrivalTimestamp": str(record.get('ApproximateArrivalTimestamp', ''))
            }
            self.data_queue.put(record_data)

        except Exception as e:
            print(f"⚠️ Error processing record: {str(e)}")

    # Update shard iterator
    if response.get('NextShardIterator'):
        self.shard_iterators[shard_id] = response['NextShardIterator']
    else:
        # Iterator expired, get a new one
        response = self.kinesis_client.get_shard_iterator(
            StreamName=self.stream_name,
            ShardId=shard_id,
            ShardIteratorType='LATEST'
        )
        self.shard_iterators[shard_id] = response['ShardIterator']

    except Exception as e:
        print(f"⚠️ Error reading from shard {shard_id}: {str(e)}")

    # Wait before next collection
    time.sleep(5)

except Exception as e:
    print(f"⚠️ Error in data collection: {str(e)}")
    time.sleep(10)

def create_rate_stream(spark):
    """Create a simple rate stream for triggering"""
    return spark \
        .readStream \

```

```

=format("rate") \
.option("rowsPerSecond", 1) \
.option("numPartitions", 1) \
.load()

def process_kinesis_data(records_data, case_schema, survey_schema):
    """Process Kinesis records and separate into case and survey events"""
    case_events = []
    survey_events = []

    for record in records_data:
        json_data = record['json_data']

        # Event type detection
        if any(field in json_data for field in ['survey_id', 'survey_timestamp', "Q1":, "Q2":']):
            # Survey event
            try:
                event_data = json.loads(json_data)
                survey_events.append(event_data)
            except:
                print(f"⚠️ Invalid survey JSON: {json_data[:50]}...")
        elif any(field in json_data for field in ['call_center_id', 'communication_mode', 'created_employee_key', "status":']):
            # Case event
            try:
                event_data = json.loads(json_data)
                case_events.append(event_data)
            except:
                print(f"⚠️ Invalid case JSON: {json_data[:50]}...")

    return case_events, survey_events

def write_to_redshift_batch(spark, events, table_name, schema, redshift_config):
    """Write a batch of events to Redshift"""
    if not events:
        return

    try:
        # Create DataFrame from events
        df = spark.createDataFrame(events, schema)

        # Transform timestamps for case events
        if table_name == "futurecart_case_details":
            df = df.select(
                col("case_no"),
                col("created_employee_key"),
                col("call_center_id"),
                col("status"),
                col("category"),
                col("sub_category"),
                col("communication_mode"),
                col("country_cd"),
                col("product_code"),
                to_timestamp(col("last_modified_timestamp"), "yyyy-MM-dd HH:mm:ss").alias("last_modified_timestamp"),
                to_timestamp(col("create_timestamp"), "yyyy-MM-dd HH:mm:ss").alias("create_timestamp")
            )
        )

        # Transform timestamps for survey events
        elif table_name == "futurecart_case_survey_details":
            df = df.select(
                col("survey_id"),
                col("case_no"),

```

```

        to_timestamp(col("survey_timestamp"), "yyyy-MM-dd HH:mm:ss").alias("survey_timestamp"),
        col("Q1"),
        col("Q2"),
        col("Q3"),
        col("Q4"),
        col("Q5")
    )

# Write to Redshift
df.write \
    .format("jdbc") \
    .option("url", redshift_config["url"]) \
    .option("dbtable", table_name) \
    .option("user", redshift_config["user"]) \
    .option("password", redshift_config["password"]) \
    .option("driver", redshift_config["driver"]) \
    .mode("append") \
    .save()

print(f" ✅ Successfully wrote {len(events)} records to {table_name}")

except Exception as e:
    print(f" ❌ Error writing to {table_name}: {str(e)}")

def test_redshift_connection(spark, redshift_config):
    """Test Redshift connection"""
    print("🌐 Testing Redshift connection...")

    try:
        test_df = spark.read \
            .format("jdbc") \
            .option("url", redshift_config["url"]) \
            .option("dbtable", "(SELECT 1 as test_col) as test_query") \
            .option("user", redshift_config["user"]) \
            .option("password", redshift_config["password"]) \
            .option("driver", redshift_config["driver"]) \
            .load()

        result = test_df.collect()
        print(f" ✅ Redshift connection successful!")
        return True

    except Exception as e:
        print(f" ❌ Redshift connection failed: {str(e)}")
        return False

def main():
    """Main streaming pipeline"""

    print("⚡ Starting FutureKart Worker-Safe Streaming Pipeline...")
    print("=" * 60)

    # Initialize Spark
    spark = create_spark_session()
    spark.sparkContext.setLogLevel("WARN")

    # Get schemas
    case_schema, survey_schema = get_schemas()

    # Redshift configuration
    redshift_config = {

```

```

"url": "jdbc:redshift://futurecart-wg-khushi.008673239246.ap-northeast-2.redshift-serverless.amazonaws.com:5439/dev",
"user": "admin",
"password": "Khushi12345",
"driver": "com.amazon.redshift.jdbc.Driver"
}

print(f"🔗 Redshift Connection: {redshift_config['url']}")
print(f"👤 User: {redshift_config['user']}")

# Test Redshift connection
if not test_redshift_connection(spark, redshift_config):
    print("🔴 Redshift connection test failed. Exiting...")
    return

# Initialize Kinesis collector
collector = KinesisDataCollector("case-survey-stream-Khushi-P1", "ap-northeast-2")
collector_thread = collector.start_collection()

print("=" * 60)
print("⌚ WORKER-SAFE STREAMING PIPELINE STATUS")
print("=" * 60)
print("✅ Kinesis Stream: sahil-proj1-futurekart-stream")
print("✅ Method: Background thread collection + micro-batch processing")
print("✅ Case Events → fact_cases_historical")
print("✅ Survey Events → fact_surveys_historical")
print("✅ Processing Interval: 15 seconds")
print("✅ Worker Safety: No UDF dependencies")
print("=" * 60)
print("⌚ Pipeline running... Press Ctrl+C to stop")
print("=" * 60)

try:
    batch_count = 0
    while True:
        # Get batch data from collector
        batch_data = collector.get_batch_data()

        if batch_data:
            print(f"📦 Processing batch {batch_count} with {len(batch_data)} records...")

            # Process the data
            case_events, survey_events = process_kinesis_data(batch_data, case_schema, survey_schema)

            # Write to Redshift
            if case_events:
                print(f"📝 Found {len(case_events)} case events")
                write_to_redshift_batch(spark, case_events, "fact_cases_historical", case_schema, redshift_config)

            if survey_events:
                print(f"📊 Found {len(survey_events)} survey events")
                write_to_redshift_batch(spark, survey_events, "fact_surveys_historical", survey_schema, redshift_config)

            if not case_events and not survey_events:
                print(f"⚠️ No valid events found in batch")

            batch_count += 1
        else:
            print("⌚ No new data, waiting...")

    # Wait before next batch

```

```

time.sleep(15)

except KeyboardInterrupt:
    print("\n🔴 Pipeline stopped by user")
except Exception as e:
    print("🔴 Pipeline error: {str(e)}")
    import traceback
    traceback.print_exc()
finally:
    print("🟡 Stopping Kinesis collector...")
    collector.stop_collection()
    collector_thread.join(timeout=5)

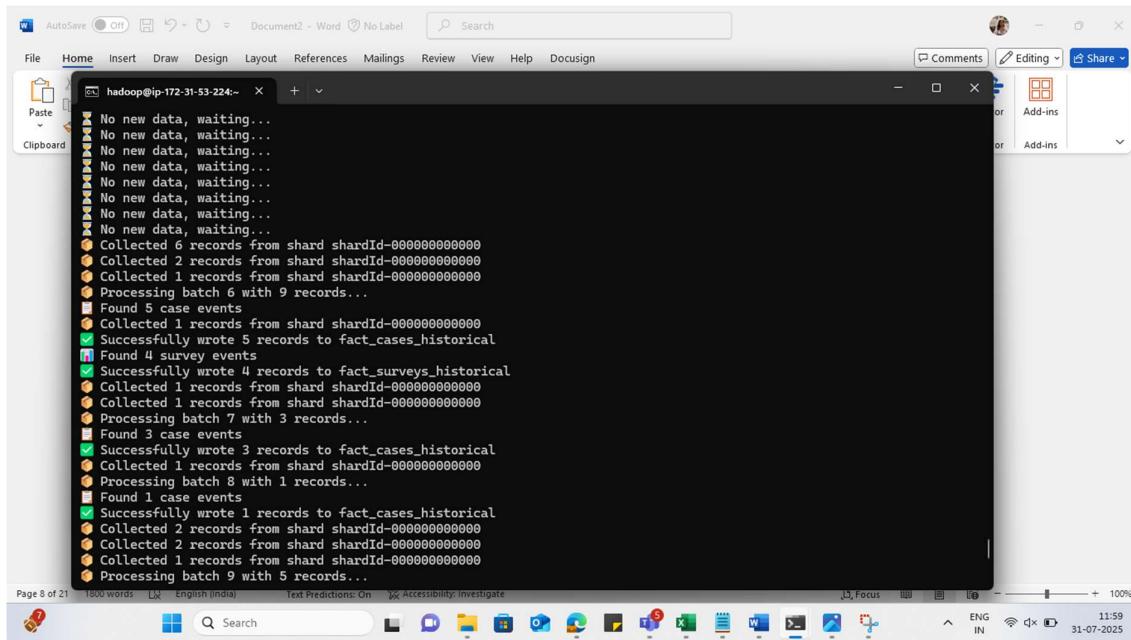
    print("🔵 Cleaning up Spark session...")
    spark.stop()
    print("🟢 Pipeline shutdown complete")

if __name__ == "__main__":
    main()

pip3 install boto3
sudo su -hadoop

spark-submit --packages "com.amazon.redshift:redshift-jdbc42:2.1.0.9" kinesis_to_redshift.py
25/07/31 05:59:40 INFO YarnClientSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minR
egisteredResourcesRatio: 0.0
🔗 Redshift Connection: jdbc:redshift://futurecart-wg-khushi.008673239246.ap-northeast-2.redshift-serverless.amazonaws.c
om:5439/dev
👤 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 suppo
rt Python 3.7 starting December 13, 2023. To continue 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/blogs/developer/python-suppor
t-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
🕒 Starting Kinesis data collection thread...=====
🕒 WORKER-SAFE STREAMING PIPELINE STATUS
=====
✅ Kinesis Stream: sahil-proj1-futurekart-stream
✅ Method: Background thread collection + micro-batch processing
✅ Case Events → fact_cases_historical
✅ Survey Events → fact_surveys_historical
✅ Processing Interval: 15 seconds
✅ Worker Safety: No UDF dependencies
=====
🕒 Pipeline running... Press Ctrl+C to stop
=====
⌚ No new data, waiting...

```



A screenshot of a Microsoft Word document window titled "hadoop@ip-172-31-53-224:~". The document contains a log of Hadoop operations. The log includes messages like "No new data, waiting...", "Collected 6 records from shard shardId-000000000000", "Successfully wrote 5 records to fact_cases_historical", and "Processing batch 9 with 5 records...". The Word interface shows standard ribbon tabs (File, Home, Insert, Draw, Design, Layout, References, Mailings, Review, View, Help, Docusign) and a ribbon bar with various icons.

```
No new data, waiting...
Collected 6 records from shard shardId-000000000000
Collected 2 records from shard shardId-000000000000
Collected 1 records from shard shardId-000000000000
Processing batch 6 with 9 records...
Found 5 case events
Collected 1 records from shard shardId-000000000000
Successfully wrote 5 records to fact_cases_historical
Found 4 survey events
Successfully wrote 4 records to fact_surveys_historical
Collected 1 records from shard shardId-000000000000
Collected 1 records from shard shardId-000000000000
Processing batch 7 with 3 records...
Found 3 case events
Successfully wrote 3 records to fact_cases_historical
Collected 1 records from shard shardId-000000000000
Processing batch 8 with 1 records...
Found 1 case events
Successfully wrote 1 records to fact_cases_historical
Collected 2 records from shard shardId-000000000000
Collected 2 records from shard shardId-000000000000
Collected 1 records from shard shardId-000000000000
Processing batch 9 with 5 records...
```

✓ Step 4: Amazon Redshift (Final Destination)

✓ 1. Created Redshift Tables

You used SQL DDL (Data Definition Language) to create dimension and fact tables in your Redshift Serverless workgroup futurecart-wg-khushi.

These include:

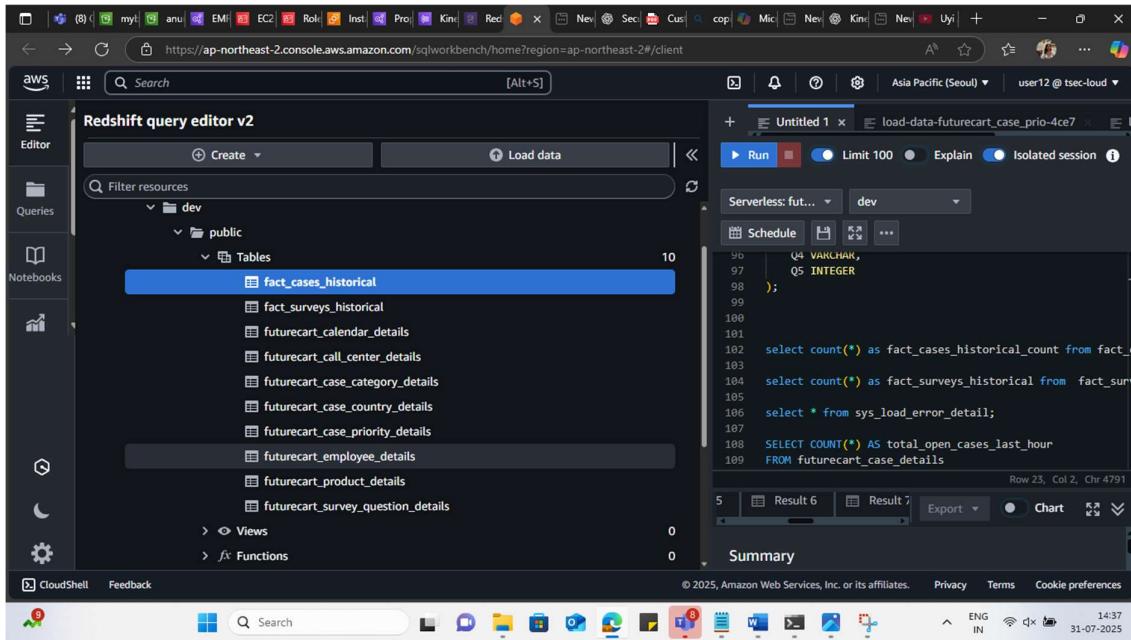
- ◆ Dimension Tables:

- 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

- ◆ Fact Tables:

- futurecart_case_details — holds case data

- futurecart_case_survey_details — holds survey data



```

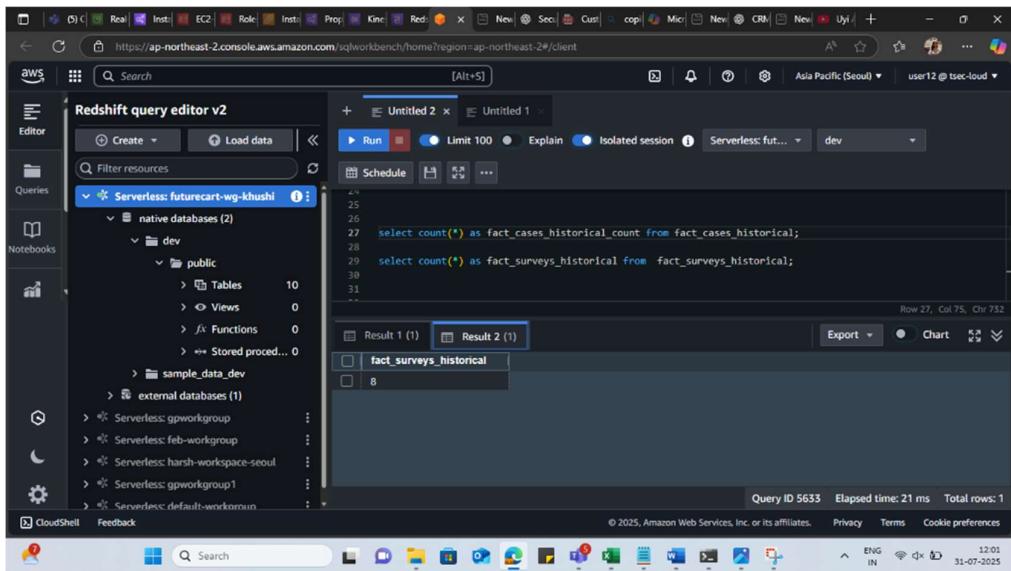
    fact_cases_historical
    fact_surveys_historical
    futurecart_calendar_details
    futurecart_call_center_details
    futurecart_case_category_details
    futurecart_case_country_details
    futurecart_case_priority_details
    futurecart_employee_details
    futurecart_product_details
    futurecart_survey_question_details
  
```

The screenshot shows the AWS Redshift Query Editor interface. On the left, the sidebar lists databases (dev, public) and tables (fact_cases_historical, fact_surveys_historical, etc.). The 'fact_cases_historical' table is selected. The main pane contains a SQL script for loading data into this table.

```

10
96   Q4_VAKCHAN,
97   Q5_INTEGER
98  );
99
100
101
102 select count(*) as fact_cases_historical_count from fact_c
103
104 select count(*) as fact_surveys_historical from fact_surv
105
106 select * from sys_load_error_detail;
107
108 SELECT COUNT(*) AS total_open_cases_last_hour
109 FROM futurecart_case_details
  
```

Checked getting the data or not:



```

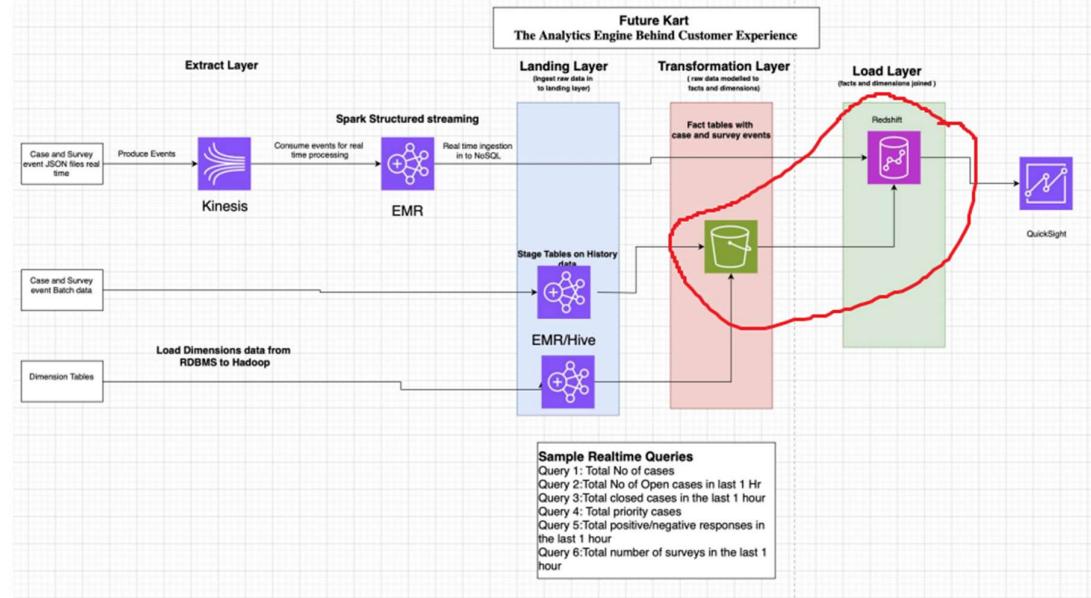
native databases (2)
  dev
    public
      Tables 10
      Views 0
      Functions 0
      Stored procedures 0
      sample_data_dev
      external databases (1)
        Serverless: gpworkgroup
        Serverless: feb-workgroup
        Serverless: harsh-workspace-seoul
        Serverless: gpworkgroup1
        Serverless: default-workgroup
  
```

The screenshot shows the AWS Redshift Query Editor interface. On the left, the sidebar lists databases (native databases, dev, public) and tables (fact_surveys_historical). The 'fact_surveys_historical' table is selected. The main pane displays the results of a query to count records in this table.

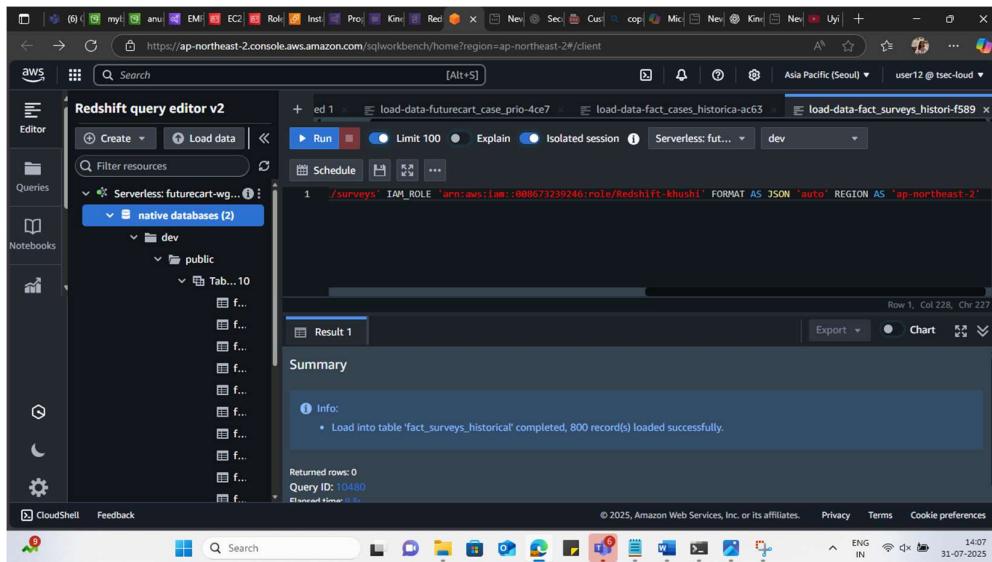
	fact_surveys_historical
8	8

Now Uploading all the data present in s3 to Redshift tables:

3. Data Flow Architecture/Process Flow



Then loaded all my data from s3 bucket to Redshift Dimension tables:



Checked if the data loaded or not:

A screenshot of the AWS SQL Workbench interface. The main area shows a query window with the following SQL code:

```
4
5   SELECT COUNT(*) AS calendar_count FROM futurecart_calendar_details;
6   SELECT COUNT(*) AS call_center_count FROM futurecart_call_center_details;
7   SELECT COUNT(*) AS case_category_count FROM futurecart_case_category_details;
8   SELECT COUNT(*) AS case_country_count FROM futurecart_case_country_details;
9   SELECT COUNT(*) AS case_priority_count FROM futurecart_case_priority_details;
10  SELECT COUNT(*) AS employee_count FROM futurecart_employee_details;
11  SELECT COUNT(*) AS product_count FROM futurecart_product_details;
12  SELECT COUNT(*) AS survey_question_count FROM futurecart_survey_question_details;
13  SELECT COUNT(*) AS fact_cases_historical_count FROM fact_cases_historical;
14  SELECT COUNT(*) AS fact_surveys_historical_count FROM fact_surveys_historical;
```

The results pane shows a single row:

calendar_count
21553

At the bottom, the status bar indicates: Query ID 10535 Elapsed time: 56 ms Total rows: 1.

Executed Queries:

A screenshot of the AWS SQL Workbench interface. The main area shows a query window with the following SQL code:

```
10
11  -- Total open cases in the last hour
12  SELECT COUNT(*) AS total_open_cases_last_hour
13  FROM fact_cases_historical
14  WHERE status = 'Open'
15  AND create_timestamp >= GETDATE() - INTERVAL '1 hour';
16
17  -- Total closed cases in the last hour
18  SELECT COUNT(*) AS total_closed_cases_last_hour
19  FROM fact_cases_historical
20  WHERE status = 'Closed'
21  AND create_timestamp >= GETDATE() - INTERVAL '1 hour';
22
23  -- Total priority cases (category/sub_category matches)
24  SELECT COUNT(*) AS total_priority_cases
25  FROM fact_cases_historical
26  WHERE category IN ('CAT1', 'CAT2')
27  OR sub_category IN ('SCAT1', 'SCAT2', 'SCAT3', 'SCAT4', 'SCAT5');
```

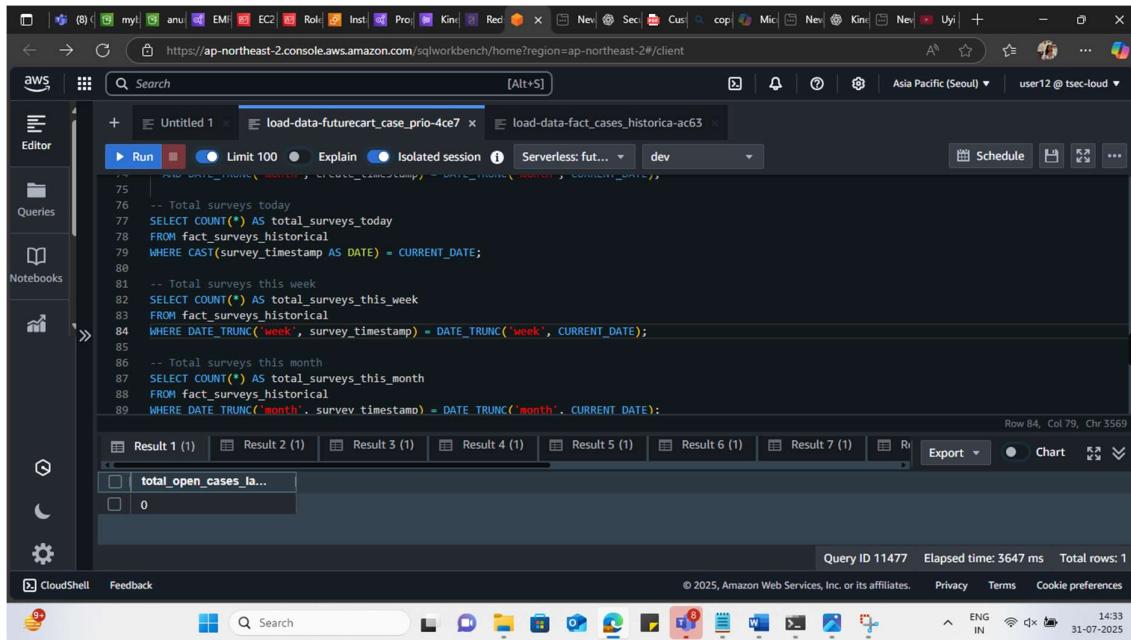
The results pane shows a single row:

closed_cases_this_w...

At the bottom, the status bar indicates: Query ID 11527 Elapsed time: 71 ms Total rows: 1.

```
10
17 -- Total open cases in the last hour
18 SELECT COUNT(*) AS total_open_cases_last_hour
19 FROM fact_cases_historical
20 WHERE status = 'Open'
21 | AND create_timestamp >= GETDATE() - INTERVAL '1 hour';
22
22 |
23 -- Total closed cases in the last hour
24 SELECT COUNT(*) AS total_closed_cases_last_hour
25 FROM fact_cases_historical
26 WHERE status = 'Closed'
27 | AND create_timestamp >= GETDATE() - INTERVAL '1 hour';
28
28 |
29 -- Total priority cases (category/sub_category matches)
30 SELECT COUNT(*) AS total_priority_cases
31 FROM fact_cases_historical
32 WHERE category IN ('CAT1', 'CAT2')
33 | OR sub_category IN ('SCAT1', 'SCAT2', 'SCAT3', 'SCAT4', 'SCAT5');
34
35 -- Total surveys in the last hour
36 SELECT COUNT(*) AS total_surveys_last_hour
37 FROM fact_surveys_historical
38 WHERE survey_timestamp >= GETDATE() - INTERVAL '1 hour';
39
40
40 -- Open cases today
41 SELECT COUNT(*) AS open_cases_today
42 FROM fact_cases_historical
43 WHERE status = 'Open'
44 | AND CAST(create_timestamp AS DATE) = CURRENT_DATE;
45
46
46 -- Open cases this week
47 SELECT COUNT(*) AS open_cases_this_week
48 FROM fact_cases_historical
49 WHERE status = 'Open'
50 | AND DATE_TRUNC('week', create_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
51
```

```
52  -- Open cases this month
53  SELECT COUNT(*) AS open_cases_this_month
54  FROM fact_cases_historical
55  WHERE status = 'Open'
56  | AND DATE_TRUNC('month', create_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
57
58  -- Closed cases today
59  SELECT COUNT(*) AS closed_cases_today
60  FROM fact_cases_historical
61  WHERE status = 'Closed'
62  | AND CAST(create_timestamp AS DATE) = CURRENT_DATE;
63
64  -- Closed cases this week
65  SELECT COUNT(*) AS closed_cases_this_week
66  FROM fact_cases_historical
67  WHERE status = 'Closed'
68  | AND DATE_TRUNC('week', create_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
69
70  -- Closed cases this month
71  SELECT COUNT(*) AS closed_cases_this_month
72  FROM fact_cases_historical
73  WHERE status = 'Closed'
74  | AND DATE_TRUNC('month', create_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
75
76  -- Total surveys today
77  SELECT COUNT(*) AS total_surveys_today
78  FROM fact_surveys_historical
79  WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;
80
81  -- Total surveys this week
82  SELECT COUNT(*) AS total_surveys_this_week
83  FROM fact_surveys_historical
84  WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
85
85  -- Total surveys this month
86  SELECT COUNT(*) AS total_surveys_this_month
87  FROM fact_surveys_historical
88  WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
89
90
91
```



A screenshot of the AWS SQL Workbench interface. The query editor contains the following SQL code:

```
-- Total surveys today
SELECT COUNT(*) AS total_surveys_today
FROM fact_surveys_historical
WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;

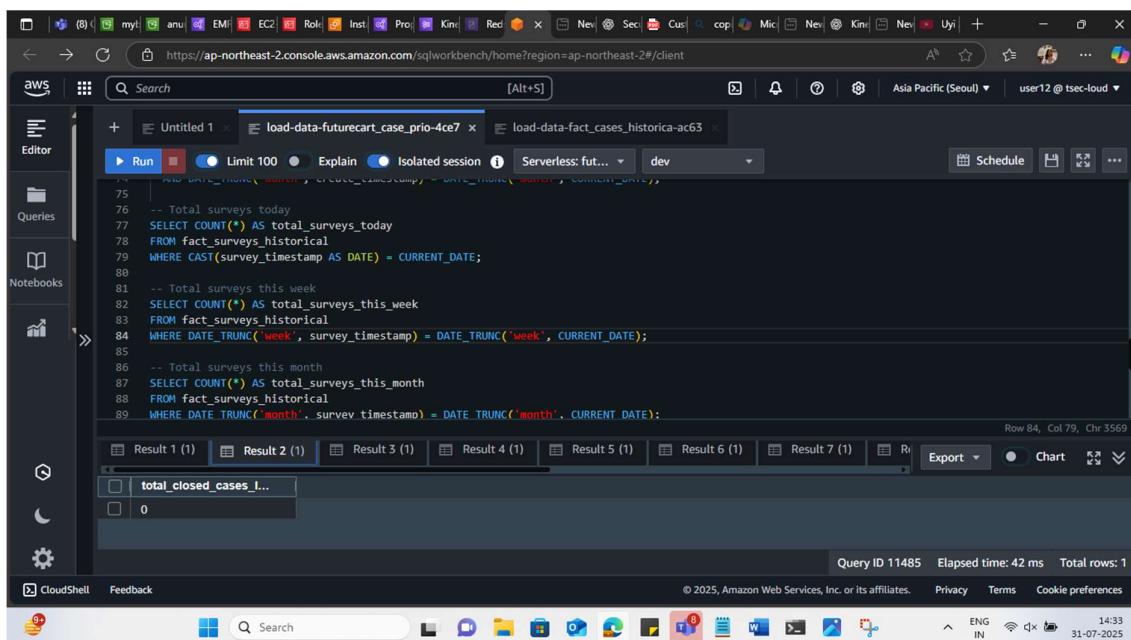
-- Total surveys this week
SELECT COUNT(*) AS total_surveys_this_week
FROM fact_surveys_historical
WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);

-- Total surveys this month
SELECT COUNT(*) AS total_surveys_this_month
FROM fact_surveys_historical
WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

The results pane shows one row of data:

total_open_cases_la...
0

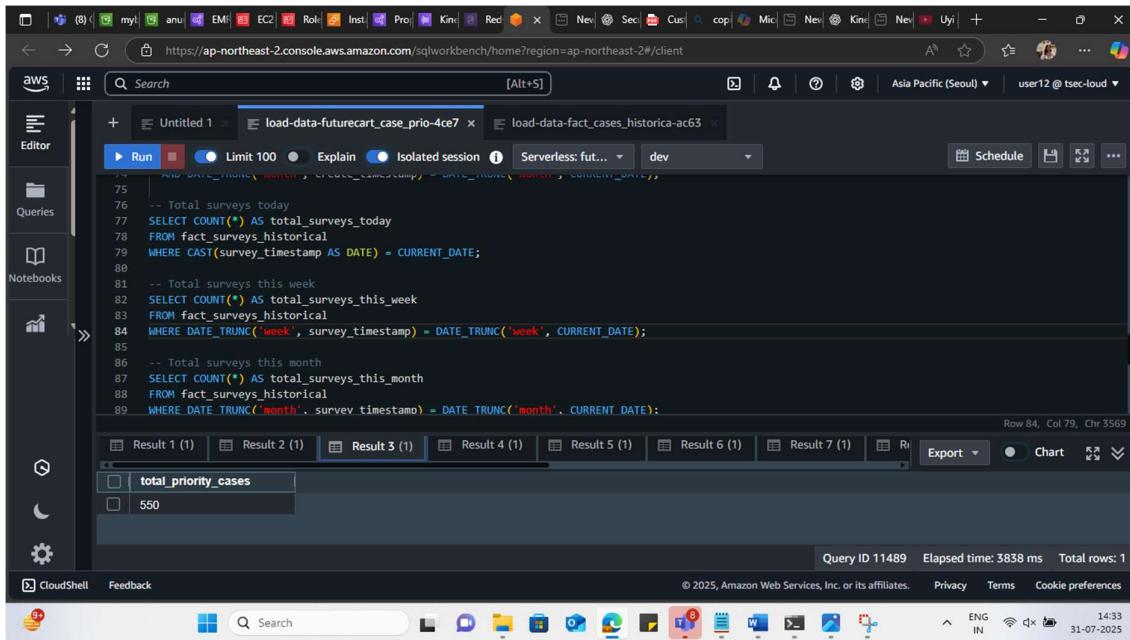
Query ID: 11477, Elapsed time: 3647 ms, Total rows: 1



A screenshot of the AWS SQL Workbench interface, identical to the one above, showing the same SQL code and results for total open cases. The results pane shows one row of data:

total_closed_cases_i...
0

Query ID: 11485, Elapsed time: 42 ms, Total rows: 1



A screenshot of the AWS SQL Workbench interface. The query editor contains the following SQL code:

```
-- Total surveys today
SELECT COUNT(*) AS total_surveys_today
FROM fact_surveys_historical
WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;

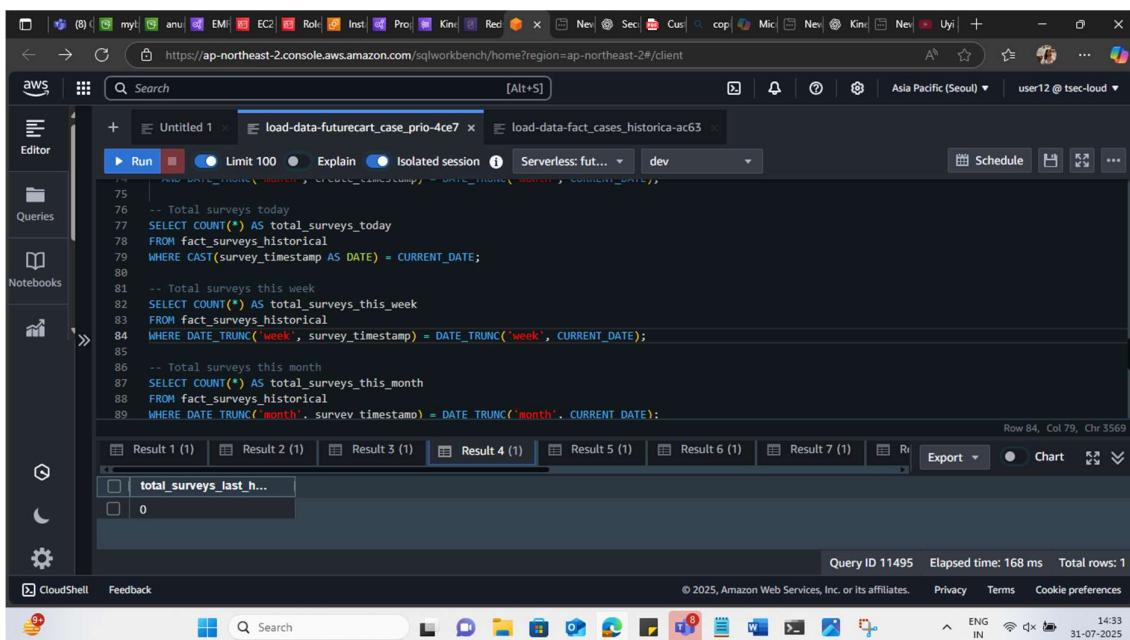
-- Total surveys this week
SELECT COUNT(*) AS total_surveys_this_week
FROM fact_surveys_historical
WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);

-- Total surveys this month
SELECT COUNT(*) AS total_surveys_this_month
FROM fact_surveys_historical
WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

The results pane shows a single row:

total_priority_cases
550

Query ID: 11489, Elapsed time: 3838 ms, Total rows: 1



A screenshot of the AWS SQL Workbench interface. The query editor contains the same SQL code as the previous screenshot:

```
-- Total surveys today
SELECT COUNT(*) AS total_surveys_today
FROM fact_surveys_historical
WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;

-- Total surveys this week
SELECT COUNT(*) AS total_surveys_this_week
FROM fact_surveys_historical
WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);

-- Total surveys this month
SELECT COUNT(*) AS total_surveys_this_month
FROM fact_surveys_historical
WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

The results pane shows a single row:

total_surveys_last_h...
0

Query ID: 11495, Elapsed time: 168 ms, Total rows: 1

The screenshot shows the AWS SQL Workbench interface. A query has been run, resulting in one row of data:

open_cases_today
36

Query ID: 11501 | Elapsed time: 3580 ms | Total rows: 1

The screenshot shows the AWS SQL Workbench interface. A query has been run, resulting in one row of data:

open_cases_this_week
86

Query ID: 11507 | Elapsed time: 3773 ms | Total rows: 1

AWS SQL Workbench interface showing a query results page. The query counts survey counts across different time periods. The results table shows one row for 'open_cases_this_mo...' with a value of 546.

```
75 -- Total surveys today
76 SELECT COUNT(*) AS total_surveys_today
77 FROM fact_surveys_historical
78 WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;
79
80 -- Total surveys this week
81 SELECT COUNT(*) AS total_surveys_this_week
82 FROM fact_surveys_historical
83 WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
84
85 -- Total surveys this month
86 SELECT COUNT(*) AS total_surveys_this_month
87 FROM fact_surveys_historical
88 WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

	Result 3 (1)	Result 4 (1)	Result 5 (1)	Result 6 (1)	Result 7 (1)	Result 8 (1)	Result 9 (1)	Export	Chart
open_cases_this_mo...									
	546								

Query ID 11517 Elapsed time: 3609 ms Total rows: 1

AWS SQL Workbench interface showing a query results page. The query counts survey counts across different time periods. The results table shows one row for 'closed_cases_today' with a value of 8.

```
75 -- Total surveys today
76 SELECT COUNT(*) AS total_surveys_today
77 FROM fact_surveys_historical
78 WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;
79
80 -- Total surveys this week
81 SELECT COUNT(*) AS total_surveys_this_week
82 FROM fact_surveys_historical
83 WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
84
85 -- Total surveys this month
86 SELECT COUNT(*) AS total_surveys_this_month
87 FROM fact_surveys_historical
88 WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

	Result 3 (1)	Result 4 (1)	Result 5 (1)	Result 6 (1)	Result 7 (1)	Result 8 (1)	Result 9 (1)	Export	Chart
closed_cases_today									
	8								

Query ID 11523 Elapsed time: 139 ms Total rows: 1

The screenshot shows two identical queries running in the AWS SQL Workbench interface. Both queries are titled "closed_cases_this_w..." and have a result count of 58.

```
75 -- Total surveys today
76 SELECT COUNT(*) AS total_surveys_today
77 FROM fact_surveys_historical
78 WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;
79
80 -- Total surveys this week
81 SELECT COUNT(*) AS total_surveys_this_week
82 FROM fact_surveys_historical
83 WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
84
85 -- Total surveys this month
86 SELECT COUNT(*) AS total_surveys_this_month
87 FROM fact_surveys_historical
88 WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

Both results show a single row with the value 58.

Query ID 11527 Elapsed time: 71 ms Total rows: 1

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 14:34 31-07-2025

The screenshot shows two identical queries running in the AWS SQL Workbench interface. Both queries are titled "closed_cases_this_month" and have a result count of 498.

```
75 -- Total surveys today
76 SELECT COUNT(*) AS total_surveys_today
77 FROM fact_surveys_historical
78 WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;
79
80 -- Total surveys this week
81 SELECT COUNT(*) AS total_surveys_this_week
82 FROM fact_surveys_historical
83 WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
84
85 -- Total surveys this month
86 SELECT COUNT(*) AS total_surveys_this_month
87 FROM fact_surveys_historical
88 WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

Both results show a single row with the value 498.

Query ID 11531 Elapsed time: 86 ms Total rows: 1

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 14:34 31-07-2025

A screenshot of the AWS SQL Workbench interface. The main area shows a query editor with the following SQL code:

```
75 -- Total surveys today
76 SELECT COUNT(*) AS total_surveys_today
77 FROM fact_surveys_historical
78 WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;
79
80 -- Total surveys this week
81 SELECT COUNT(*) AS total_surveys_this_week
82 FROM fact_surveys_historical
83 WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
84
85 -- Total surveys this month
86 SELECT COUNT(*) AS total_surveys_this_month
87 FROM fact_surveys_historical
88 WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

The results pane shows one row of data:

total_surveys_today
8

At the bottom, the status bar indicates: Query ID 11535 Elapsed time: 109 ms Total rows: 1.

A screenshot of the AWS SQL Workbench interface, identical to the one above but with a different result. The results pane shows one row of data:

total_surveys_this_week
88

At the bottom, the status bar indicates: Query ID 11539 Elapsed time: 3606 ms Total rows: 1.

AWS SQL Workbench interface showing a query in the 'Editor' tab. The query counts surveys for different time periods: today, this week, and this month. The results show 808 total surveys this month.

```
75 -- Total surveys today
76 SELECT COUNT(*) AS total_surveys_today
77 FROM fact_surveys_historical
78 WHERE CAST(survey_timestamp AS DATE) = CURRENT_DATE;
79
80 -- Total surveys this week
81 SELECT COUNT(*) AS total_surveys_this_week
82 FROM fact_surveys_historical
83 WHERE DATE_TRUNC('week', survey_timestamp) = DATE_TRUNC('week', CURRENT_DATE);
84
85 -- Total surveys this month
86 SELECT COUNT(*) AS total_surveys_this_month
87 FROM fact_surveys_historical
88 WHERE DATE_TRUNC('month', survey_timestamp) = DATE_TRUNC('month', CURRENT_DATE);
```

Result 7 (1) | Result 8 (1) | Result 9 (1) | Result 10 (1) | Result 11 (1) | Result 12 (1) | Result 13 (1) | Export | Chart

	total_surveys_this_month
	808

Query ID 11543 Elapsed time: 101 ms Total rows: 1

Redshift Query Editor v2 interface showing a query in the 'Editor' tab. The query counts positive and negative responses by day. The results show daily counts from July 20 to 25, 2025.

```
5
6 SELECT
7   CAST(survey_timestamp AS DATE) AS day,
8
9   ... Positive responses (6-10)
10  SUM(CASE WHEN TRY_CAST("Q1" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
11  ..... CASE WHEN TRY_CAST("Q2" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
12  ..... CASE WHEN TRY_CAST("Q3" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
```

Result 1 (10)

day	total_positive	total_negative
2025-07-20	162	158
2025-07-21	157	163
2025-07-22	170	150
2025-07-23	175	145
2025-07-24	162	158
2025-07-25	126	194

Query ID 6440 Elapsed time: 3819 ms Total rows: 10

Redshift Query Editor v2 interface showing a query in the 'Editor' tab. The query counts positive and negative responses by week. The results show weekly counts from July 14 to 28, 2025.

```
25
26 SELECT
27   DATE_TRUNC('week', CAST(survey_timestamp AS DATE)) AS week,
28
29   ... Positive responses (6-10)
30  SUM(CASE WHEN TRY_CAST("Q1" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
31  ..... CASE WHEN TRY_CAST("Q2" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
32  ..... CASE WHEN TRY_CAST("Q3" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
```

Result 1 (3)

week	total_positive	total_negative
2025-07-14 00:00:00	162	158
2025-07-21 00:00:00	1126	1114
2025-07-28 00:00:00	318	322

Redshift query editor v2

+ Untitled 1 × load-data-futurecart_case_surv-a393 × load-data-futurecart_case_surv-5c86 ×

Run Limit 100 Explain Isolated session Serverless: fe... fe-db

Schedule

45
46 SELECT
47 DATE_TRUNC('month', CAST(survey_timestamp AS DATE)) AS month,
48 -----Positive_responses-(6-10)
49 SUM(CASE WHEN TRY_CAST("Q1" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
50 CASE WHEN TRY_CAST("Q2" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
51 CASE WHEN TRY_CAST("Q3" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
52 CASE WHEN TRY_CAST("Q4" AS INT) BETWEEN 6 AND 10 THEN 1 ELSE 0 END +
Row 46, Col 1, Chr 2921

Result 1 (1)

	month	total_positive	total_negative
1	2025-07-01 00:00:00	1606	1594

Export Chart

The screenshot shows the Redshift query editor interface. On the left is a sidebar with 'Editor', 'Queries', 'Notebooks', and a search bar. The main area has tabs for 'Untitled 1' and 'load-data-futurecart_case_surv-a393'. The current tab is 'load-data-futurecart_case_surv-5c86'. The toolbar includes 'Run', 'Limit 100', 'Explain', 'Isolated session', 'Serverless: fe...', and 'feb-db'. Below the toolbar are buttons for 'Schedule', 'Run', 'Stop', and '...'. The code editor contains a multi-line SQL query. The result pane shows a single row of data from the query.