Customer Data Platform (CDP) ON GCP

Name	Employee ID	Cohort Code
P Suhas Rao	2392975	INTAIA25GCP002

Table of Contents

1. Introduction

1.1 About this Document

1.1.1 Purpose & Scope of the Document

This document presents the implementation details of a Customer Data Platform (CDP) built on Google Cloud Platform (GCP). It outlines the system architecture, design components, data security mechanisms, KPIs, dashboards, and monitoring setup.

1.2 About the Software System

1.2.1 Scope of the System

The system ingests real-time customer and web analytics data, processes and stores it in BigQuery, applies encryption to sensitive fields using KMS, and visualizes business KPIs through interactive dashboards.

1.2.2 Exclusions

This system does not include advanced machine learning modeling, third-party marketing integrations, or external data source connectors beyond CRM and web analytics.

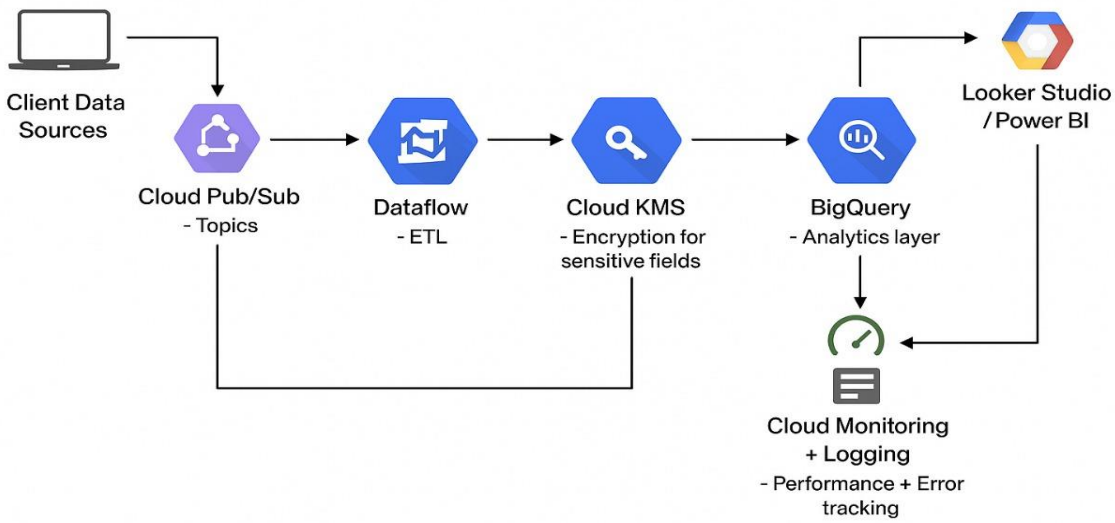
1.2.3 System Perspective

The CDP system is a cloud-native solution integrated into the GCP ecosystem, interacting with services like Pub/Sub, Dataflow, BigQuery, Cloud KMS, IAM, and Looker Studio.

1.2.4 System Environment

- Cloud Platform: Google Cloud Platform (GCP)
- Storage: BigQuery
- Data Ingestion: Cloud Pub/Sub
- Processing: Dataflow
- Security: Cloud KMS, IAM
- Visualization: Looker Studio
- Monitoring: Cloud Monitoring & Logging

1.2.5 Architecture Diagram



(1) High Level Architecture

1.2.6 User Characteristics

- Data Analysts: Use dashboards for business insights
- Engineers: Maintain pipelines and security
- Managers: Track KPIs via dashboards
- Admins: Configure access and monitor performance

1.2.7 Impact of the System

- Real-time insight into customer behavior
- Improved data governance and security
- Centralized data source for marketing and business intelligence

1.2.8 Assumptions, Risks / Constraints

- Assumes stable cloud resource availability
- Risk of pipeline failure if Pub/Sub/Dataflow is misconfigured
- Constraints include cost limitations and dashboard tool scope

1.2.9 Design Constraints

- Must use only GCP-native services
- Sensitive fields must be encrypted before storage
- Dashboards must be built on Looker Studio

2 System Requirements

2.1 Functional Requirements

- Real-time ingestion of CRM and Web Analytics data via Pub/Sub
- Transformation of data using Dataflow
- Secure storage of data in BigQuery
- Encryption of sensitive transaction fields using Cloud KMS
- Implementation of 8 business KPIs using BigQuery
- Creation of 3 Looker Studio dashboards
- Email alerting on BigQuery job errors using Monitoring
- Access control using IAM and VPC Service Controls

3 Implementation

Data Sources

- **CRM Data** (CSV format) – 9 tables -> customers, addresses, demographics, preferences, loyalty program, engagement, customer feedback, customer segments, data login activity.

- **Web Analytics Data** (JSON format) - 5 tables - > click events, conversion events, device information, page views, session data.

- **Transactional Data** (CSV format) – 15 tables -> orders, payments, invoices, shipments, returns, refunds, shipment tracking, order returns, payment transaction, order items, inventory, product catalog, promotions, supplier data, inventory logs.

Note: All data are generated using faker library of python.

Data Ingestion and Storage

Raw data files were initially stored in Google Drive. These files were programmatically transferred to Google Cloud Storage (GCS) Buckets namely crm_sample, web_analy, suhas_transaction using a Python script executed in Google Colab. This automated approach ensured secure and efficient migration of data from Drive to the cloud.

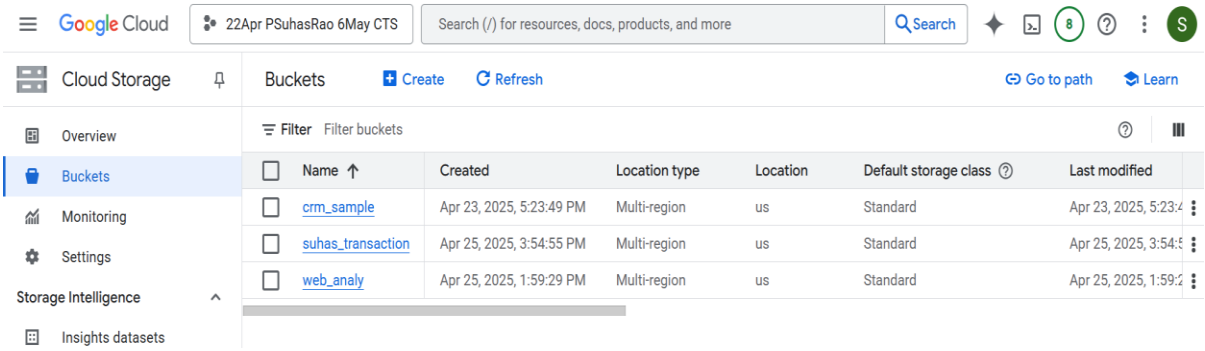
Once the data was available in GCS, it was imported into BigQuery by creating three separate datasets corresponding to the source systems:

CRM

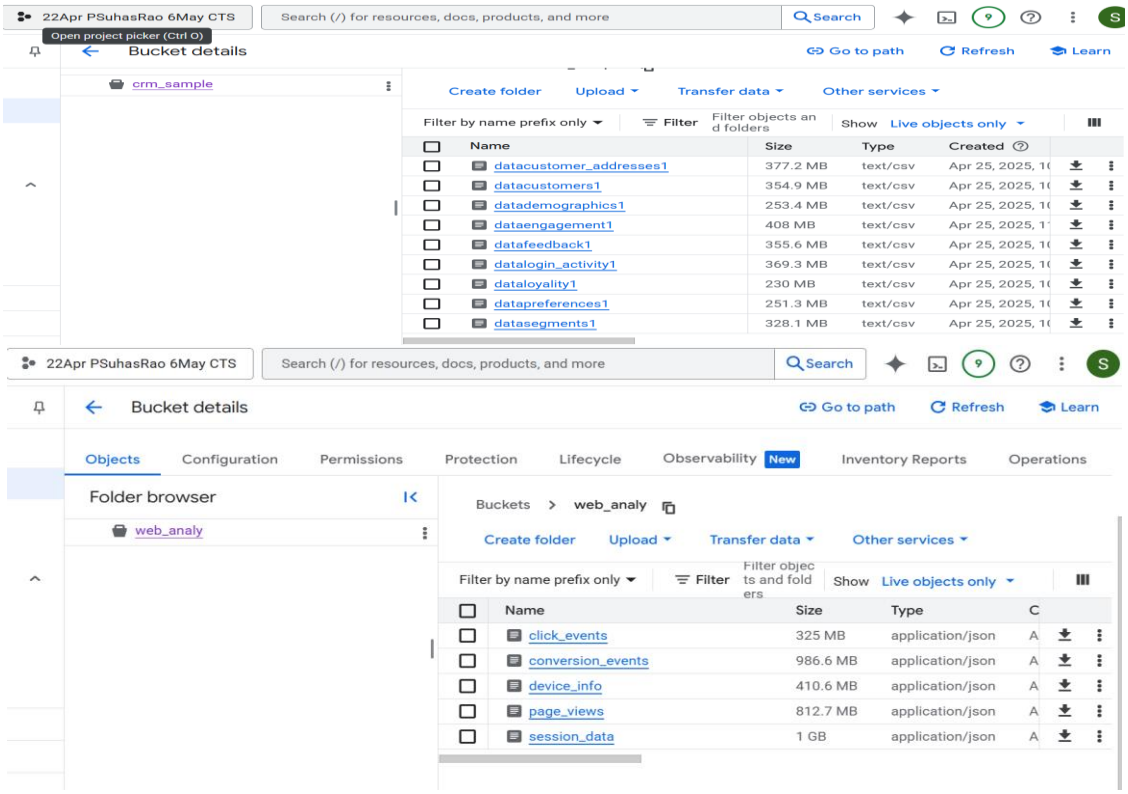
Web

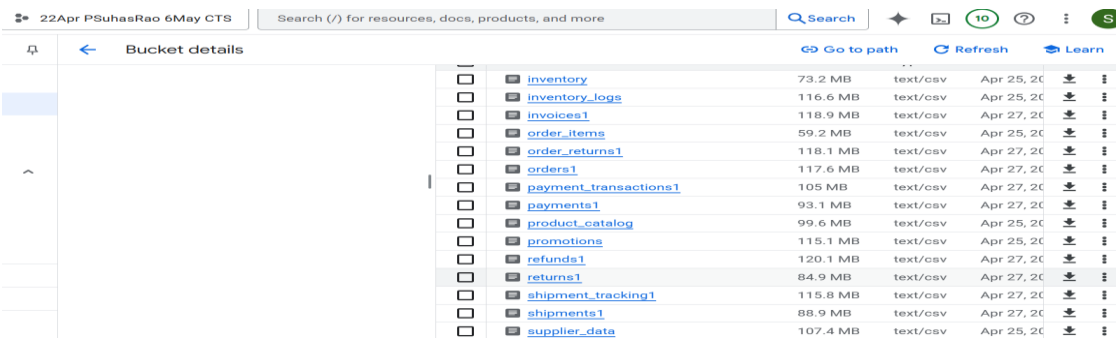
Transaction

Each dataset was structured with appropriate tables reflecting the schema of the original source data. This served as the raw data layer for the analytics workflow.

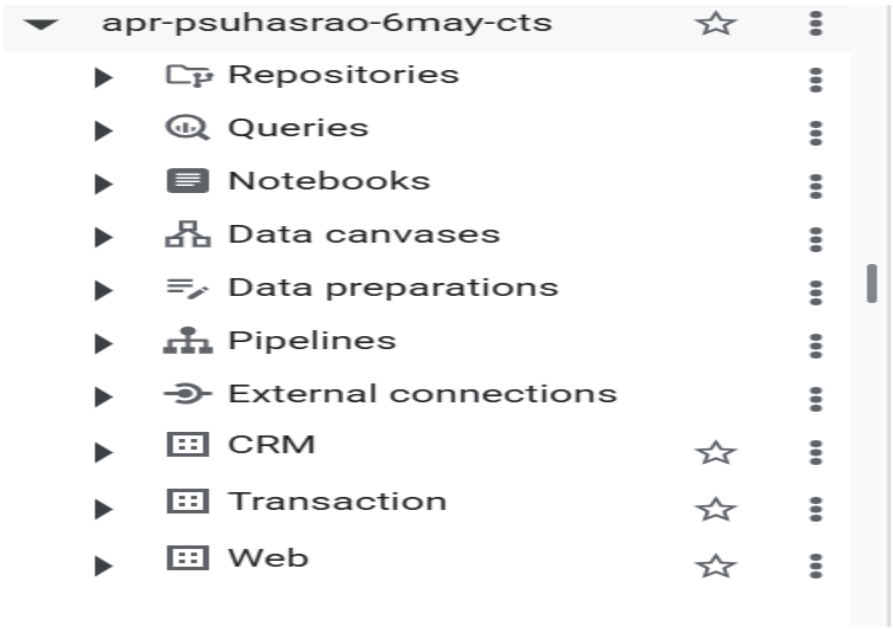


(2) Buckets

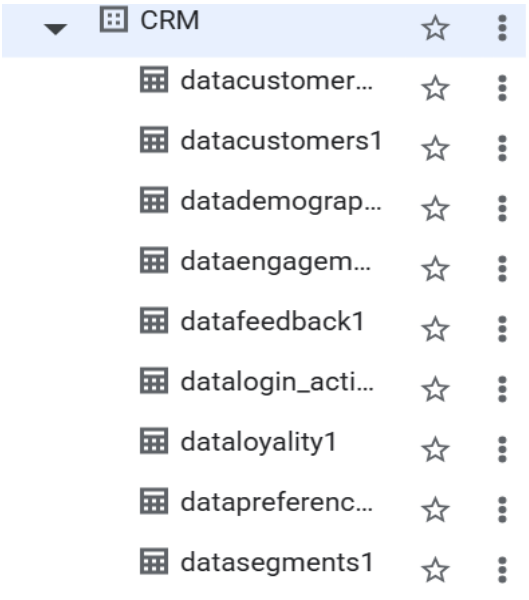




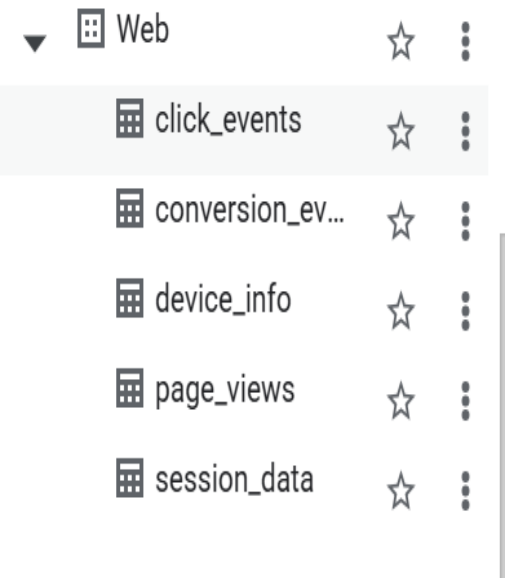
(3) Data in respective buckets



(4) BigQuery Datasets



(5) Tables in CRM



(6) Tables in Web

(7) Tables in Transaction

Real-time Streaming Simulation Using Pub/Sub and Dataflow

To simulate a real-time data streaming pipeline, I utilized Google Cloud Pub/Sub and Dataflow. However, due to the high cost and time involved in streaming the full dataset of 20 lakh records, a scaled-down version was implemented using a sample CSV file (employee.csv) for demonstration purposes.

Pub/Sub Topic Creation and Data Publishing

- A Pub/Sub topic named emp_ex was created.
- A Python script was developed to read the employee.csv file and publish records one by one to the emp_ex topic.
- A delay of 10 seconds was introduced between each record to simulate real-time ingestion behavior.

Streaming Dataflow Job

- An **empty table** was created in BigQuery with the schema corresponding to the structure of the employee data.
- A **Dataflow job** was launched using the **"Pub/Sub to BigQuery" template**.

  Input: Pub/Sub topic emp_ex

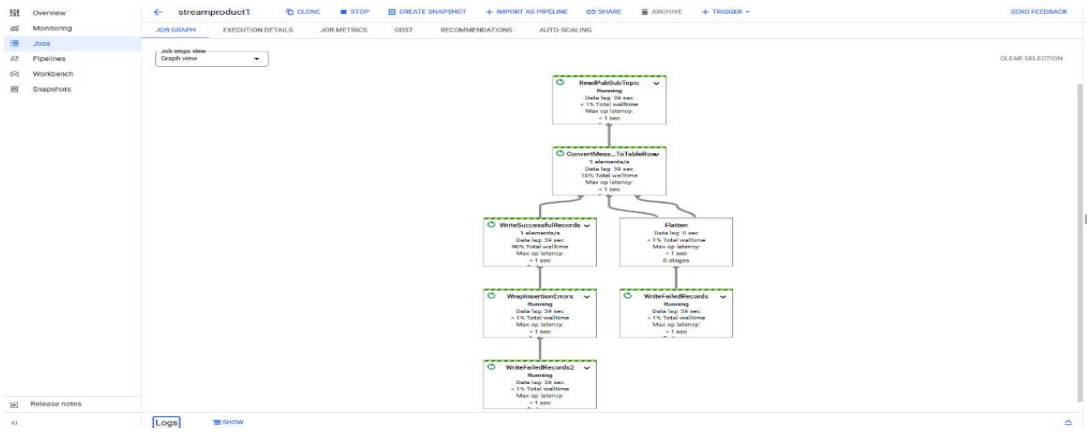
Output: BigQuery table

- As the script ran, data published to Pub/Sub **was ingested in near real-time** by Dataflow and populated into the BigQuery table.

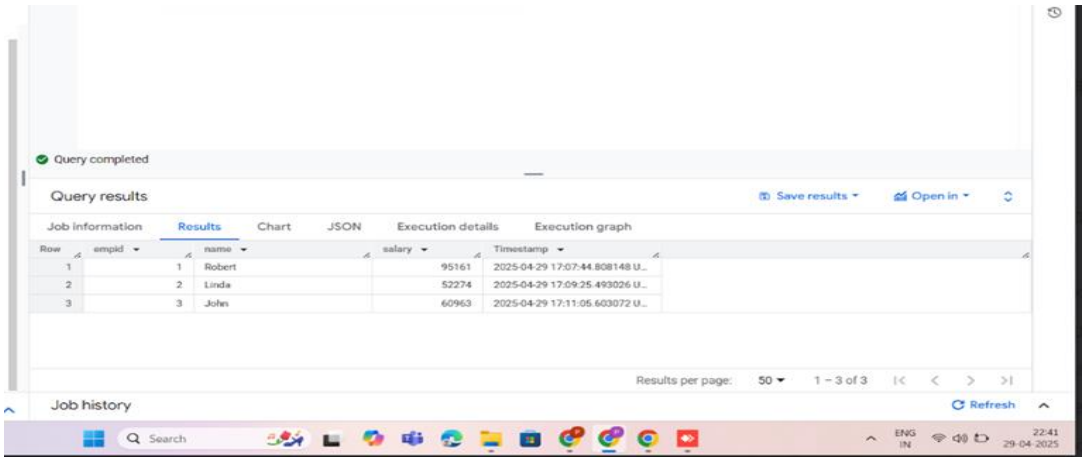
```
import csv
import time
from google.cloud import pubsub_v1
import json
from datetime import datetime

# Replace with your GCP project ID and Pub/Sub topic name
project_id = "apr-psuhasrao-6may-cts"
topic_id = "emp_ex"

# Pub/Sub Publisher Client
publisher = pubsub_v1.PublisherClient()
topic_path = publisher.topic_path(project_id, topic_id)

# Function to publish a message to Pub/Sub
def publish_message(empid, firstname, salary):
    timestamp = datetime.utcnow().isoformat() + "Z"  # Add current UTC timestamp
    message = {
        "empid": empid,
        "name": firstname,
        "salary": salary,
        "timestamp": timestamp,
    }
    # Convert the message to JSON and encode as bytes
    message_json = json.dumps(message).encode("utf-8")
    # Publish the message
    publish_status = publisher.publish(topic_path, message_json)
    print(f"Published message with ID: {publish_status.result()} - {message}")

# Path to your CSV file
csv_file_path = "/content/drive/MyDrive/employee.csv"

# Publish records one by one
with open(csv_file_path, mode="r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        empid = row["EmployeeID"]
        firstname = row["FirstName"]
        salary = row["Salary"]
        # Publish the record
        publish_message(empid, firstname, salary)
        # Wait for 30 seconds before publishing the next record
        time.sleep(10)  # Adjusted time to 30 seconds
```

```
Published message with ID: 14671473530799644 - {'empid': '1', 'name': 'Robert', 'salary': '95161', 'timestamp': '2025-05-05T06:09:22.128566Z'}
Published message with ID: 14671181590153440 - {'empid': '2', 'name': 'Linda', 'salary': '52274', 'timestamp': '2025-05-05T06:09:32.358080Z'}
Published message with ID: 14671217893092414 - {'empid': '3', 'name': 'John', 'salary': '60963', 'timestamp': '2025-05-05T06:09:42.392974Z'}
```

(8) Code For Publishing Data to Pub/Sub Topic



(9) Employee Streaming Dataflow Job



(10) Data Streamed into BigQuery Table

Data Processing with Dataflow (Batch Mode)

Although the intended architecture involved real-time streaming using Pub/Sub and Dataflow, the available data was static. Hence, a batch processing approach was adopted to simulate the ingestion pipeline.

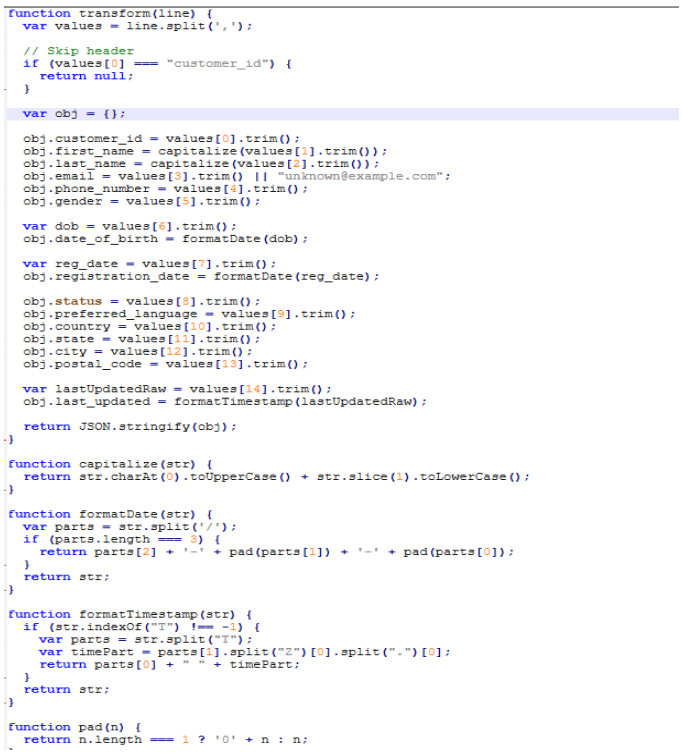
Each source (e.g., Customer, Click Events) included:

- A **JavaScript (.js) transformation file** defining how each record should be parsed and cleaned.
- A **JSON schema file** specifying the structure for the BigQuery table.

The **Cloud Dataflow "Text Files on Cloud Storage to BigQuery" template** was used to launch Dataflow jobs. These jobs:

- Pulled raw data from Cloud Storage.
- Applied the transformation logic using the .js files.
- Loaded the clean and structured data into BigQuery according to the defined .json schema.

This method effectively mimicked a real-time pipeline using batch jobs, ensuring schema enforcement and preprocessing before storage in BigQuery.

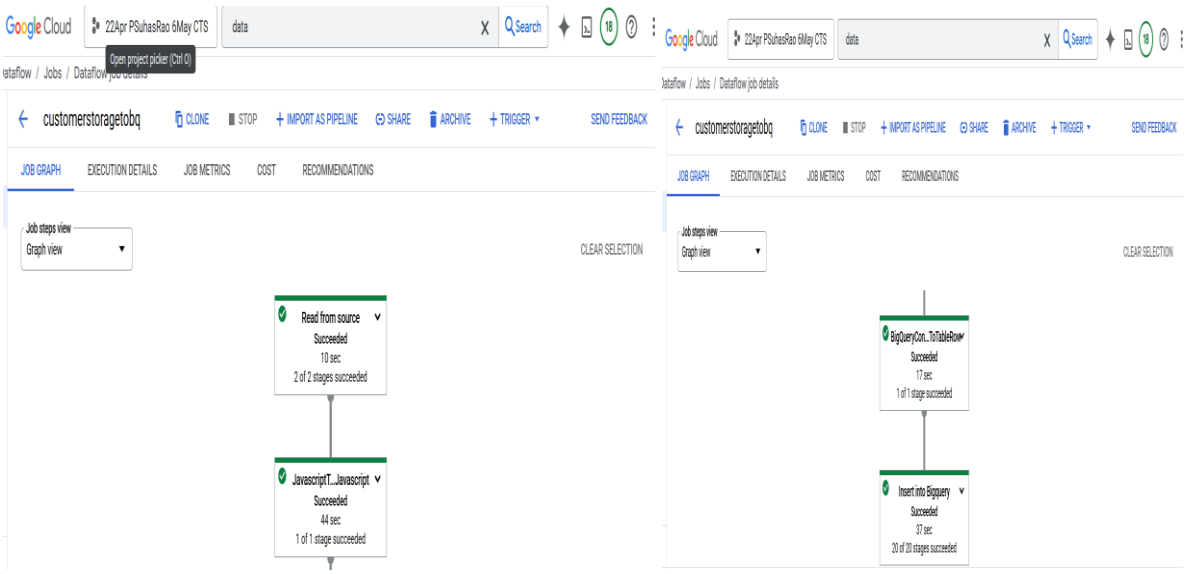


```
function transform(line) {
  var values = line.split(',');

  // Skip header
  if (values[0] === "customer_id") {
    return null;
  }

  var obj = {};

  obj.customer_id = values[0].trim();
  obj.first_name = capitalize(values[1].trim());
  obj.last_name = capitalize(values[2].trim());
  obj.email = values[3].trim() || "unknown@example.com";
  obj.phone_number = values[4].trim();
  obj.gender = values[5].trim();

  var dob = values[6].trim();
  obj.date_of_birth = formatDate(dob);

  var reg_date = values[7].trim();
  obj.registration_date = formatDate(reg_date);

  obj.status = values[8].trim();
  obj.preferred_language = values[9].trim();
  obj.country = values[10].trim();
  obj.state = values[11].trim();
  obj.city = values[12].trim();
  obj.postal_code = values[13].trim();

  var lastUpdatedRaw = values[14].trim();
  obj.last_updated = formatTimestamp(lastUpdatedRaw);

  return JSON.stringify(obj);
}

function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1).toLowerCase();
}

function formatDate(str) {
  var parts = str.split('/');
  if (parts.length === 3) {
    return parts[2] + '-' + pad(parts[1]) + '-' + pad(parts[0]);
  }
  return str;
}

function formatTimestamp(str) {
  if (str.indexOf("T") !== -1) {
    var parts = str.split("T");
    var timePart = parts[1].split("Z")[0].split(".")[0];
    return parts[0] + " " + timePart;
  }
  return str;
}

function pad(n) {
  return n.length === 1 ? '0' + n : n;
}
```

(11) Customer.js

```
{
  "BigQuery Schema": [
    { "name": "customer_id", "type": "STRING" },
    { "name": "first_name", "type": "STRING" },
    { "name": "last_name", "type": "STRING" },
    { "name": "email", "type": "STRING" },
    { "name": "phone_number", "type": "STRING" },
    { "name": "gender", "type": "STRING" },
    { "name": "date_of_birth", "type": "DATE" },
    { "name": "registration_date", "type": "DATE" },
    { "name": "status", "type": "STRING" },
    { "name": "preferred_language", "type": "STRING" },
    { "name": "country", "type": "STRING" },
    { "name": "state", "type": "STRING" },
    { "name": "city", "type": "STRING" },
    { "name": "postal_code", "type": "STRING" },
    { "name": "last_updated", "type": "TIMESTAMP" }
  ]
}
```

(12) Customer.json



(13) Customer Streaming Dataflow Job

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags
customer_id	STRING	NULLABLE	-	-	-	-
first_name	STRING	NULLABLE	-	-	-	-
last_name	STRING	NULLABLE	-	-	-	-
email	STRING	NULLABLE	-	-	-	-
phone_number	STRING	NULLABLE	-	-	-	-
gender	STRING	NULLABLE	-	-	-	-
date_of_birth	DATE	NULLABLE	-	-	-	-
registration_date	DATE	NULLABLE	-	-	-	-
status	STRING	NULLABLE	-	-	-	-
preferred_language	STRING	NULLABLE	-	-	-	-

(14) Customer Table Schema

```
1  SELECT * FROM `apr-psuhasrao-6may-cts.Crm_temp.Customer` LIMIT 100;
```

Query completed

Query results

Save results | Open in

Job information | Results | Chart | JSON | Execution details | Execution graph

Row	customer_id	first_name	last_name	email
92	CUST1977793	Maria	Webb	janewhite@example.org
93	CUST0766991	Jessica	Rowe	jessicabeck@example.org
94	CUST0107578	Michael	White	watsonkristie@example.net
95	CUST1732257	Rebecca	Pearson	bullocklaura@example.com
96	CUST1138230	Rebecca	Hill	owilliams@example.org
97	CUST1346032	John	Lamb	sandra28@example.org
98	CUST1639055	Cynthia	Marshall	marcus31@example.org
99	CUST1404915	Jamie	Poole	dharper@example.org
100	CUST0628466	Tiffany	Dudley	cunninghampaige@example.c.

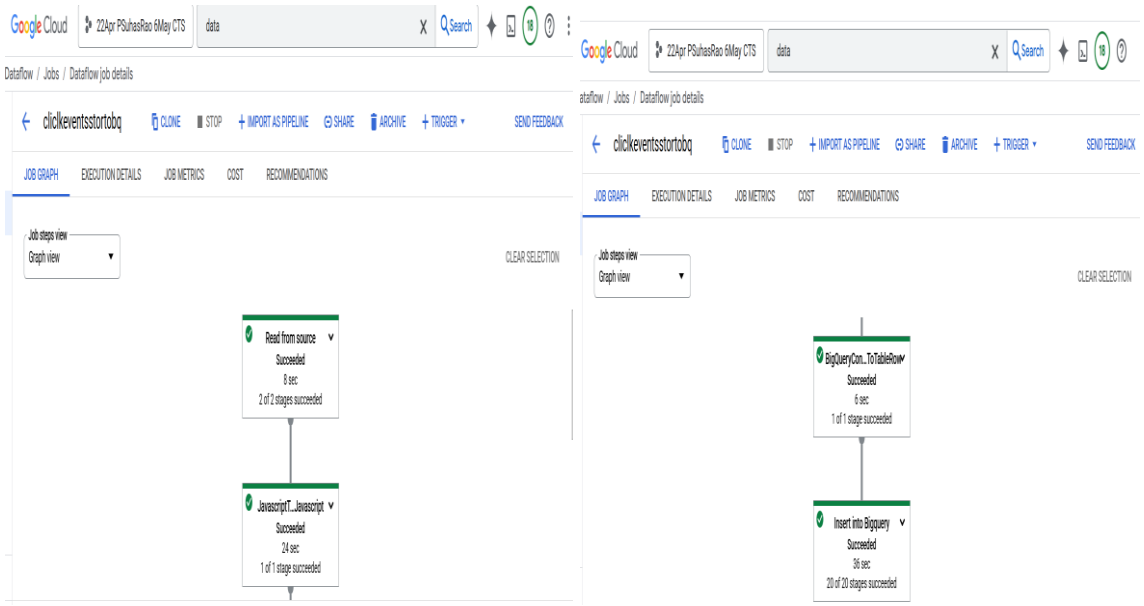
(15) Transformed Customer Data

```javascript
function transform(jsonMessage) {
  var message = JSON.parse(jsonMessage);
  var obj = {};

  obj.click_id = message.click_id ? message.click_id.trim() : null;
  obj.customer_id = message.customer_id ? message.customer_id.trim() : null;
  obj.element_clicked = message.element_clicked ? message.element_clicked.trim() : null;

  // Convert timestamp if present
  var ts = message.timestamp ? message.timestamp.trim() : null;
  if (ts && ts.indexOf("T") !== -1) {
    var parts = ts.split("T");
    var timePart = parts[1].split("Z")[0].split(".")[0];
    obj.timestamp = parts[0] + " " + timePart;
  } else {
    obj.timestamp = ts;
  }

  obj.page_url = message.page_url ? message.page_url.trim() : null;
  obj.click_position_x = message.click_position && message.click_position.x !== undefined ? parseInt(message.click_position.x) : null;
  obj.click_position_y = message.click_position && message.click_position.y !== undefined ? parseInt(message.click_position.y) : null;
  obj.session_id = message.session_id ? message.session_id.trim() : null;
  obj.device_type = message.device_type ? message.device_type.trim() : null;
  obj.browser = message.browser ? message.browser.trim() : null;
  obj.cta_flag = (typeof message.cta_flag === "boolean") ? message.cta_flag : null;

  return JSON.stringify(obj);
}
```

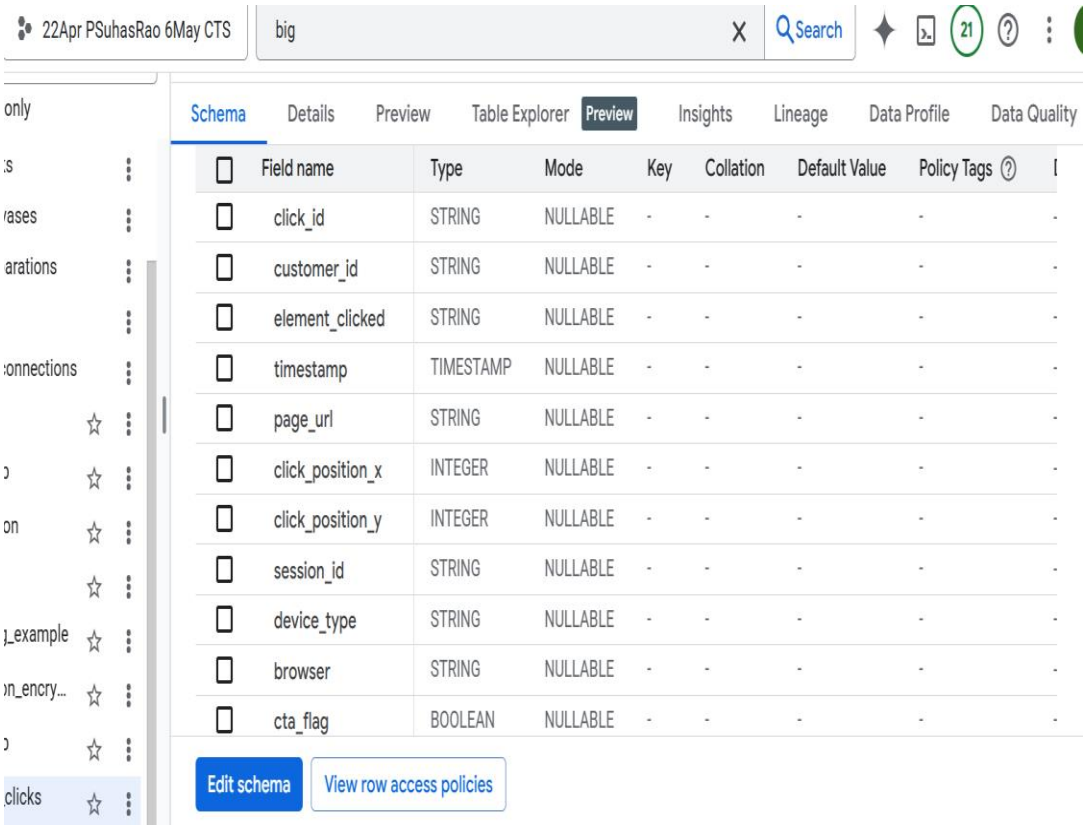
(16) Click_Events.js

```json
{
  "BigQuery Schema": [
    { "name": "click_id", "type": "STRING" },
    { "name": "customer_id", "type": "STRING" },
    { "name": "element_clicked", "type": "STRING" },
    { "name": "timestamp", "type": "TIMESTAMP" },
    { "name": "page_url", "type": "STRING" },
    { "name": "click_position_x", "type": "INTEGER" },
    { "name": "click_position_y", "type": "INTEGER" },
    { "name": "session_id", "type": "STRING" },
    { "name": "device_type", "type": "STRING" },
    { "name": "browser", "type": "STRING" },
    { "name": "cta_flag", "type": "BOOLEAN" }
  ]
}
```

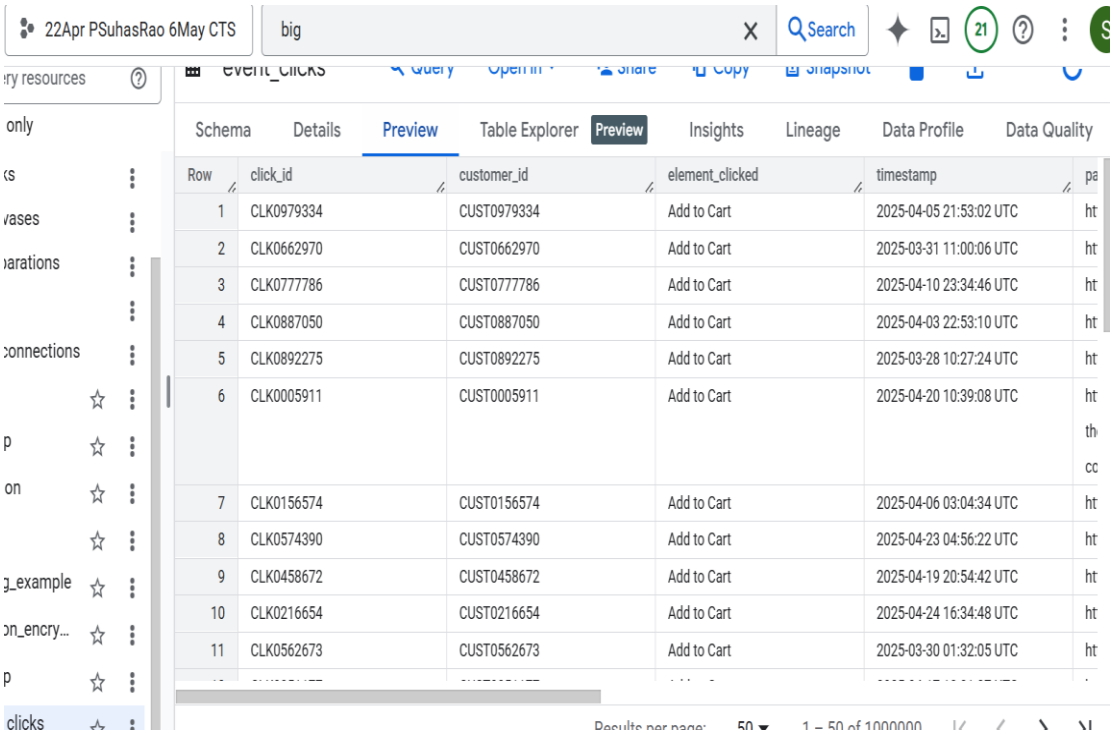
(17) Click_Events.json



(18) Click events Streaming Dataflow Job



(19) Click Events Table Schema



(20) Transformed Click Events Data

Encryption of Sensitive fields in Transactional Data

The following sensitive fields were encrypted before storage:

orders1: customer_id, shipping_address, billing_address, payment_method

order_items: supplier_id, supplier_name

payments1: customer_id, payment_method, transaction_id, authorization_code, account_number_masked, payment_ip

invoices1: customer_id, billing_address, payment_method, bank_name, account_number_masked

shipments1: customer_id, tracking_number, origin, destination

returns1: customer_id, return_reason, processed_by

refunds1: customer_id, bank_name, account_number_masked, refund_reason

inventory: supplier_id, batch_number

product_catalog: supplier_id

promotions: promotion_code

payment_transactions1: customer_id, account_number_masked, transaction_reference

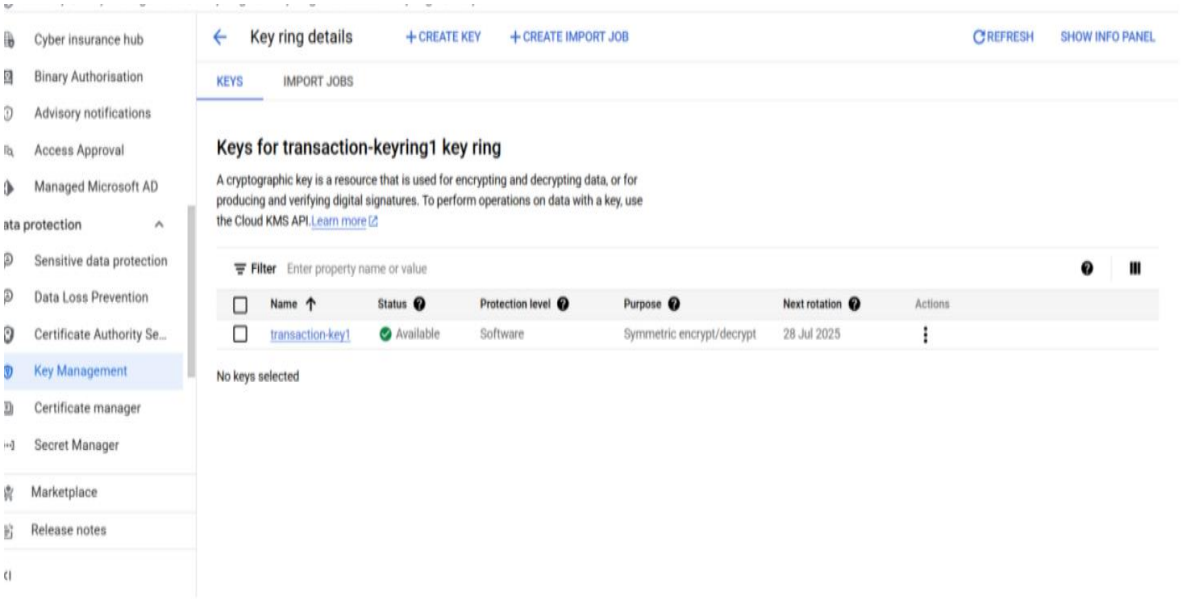
shipment_tracking1: customer_id, tracking_number, delivery_address

order_returns1: customer_id, pickup_address

supplier_data: supplier_name, contact_name, contact_email, phone_number, address

inventory_logs: logged_by, approved_by

These fields were identified as containing personally identifiable information (PII) or confidential data and were encrypted to ensure data security during storage and processing.



(21) Key Generation in KMS

```
from google.colab import files
!pip install google-cloud-storage google-cloud-kms
```

```
uploaded=files.upload()
```

```
 Choose Files  No file chosen            Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
iaving apr-psuhasrao-6may-cts-f6a3dec60377.json to apr-psuhasrao-6may-cts-f6a3dec60377.json
```

```
import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "apr-psuhasrao-6may-cts-f6a3dec60377.json"

import asyncio
import base64
import csv
import io
import nest_asyncio
from google.cloud import storage, kms_v1
from concurrent.futures import ThreadPoolExecutor
```

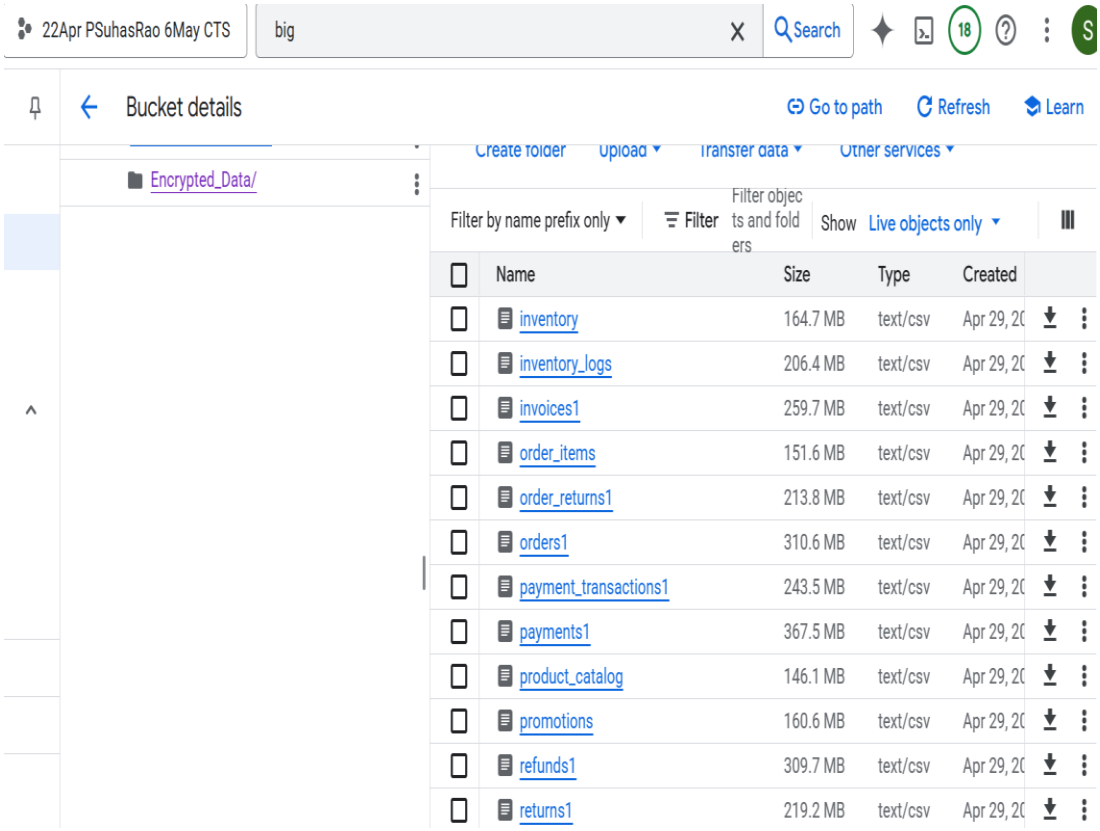
```
# CONFIGURATION
BUCKET_NAME = "suhas_transaction"
DESTINATION_FOLDER = "Encrypted_Data/"
destination_bucket_name = "encrypted_transaction"
KEY_NAME = "projects/apr-psuhasrao-6may-cts/locations/global/keyRings/transaction-keyring1/cryptoKeys/transaction-key1"
```

```
file_list = [
    "orders1", "order_items", "payments1", "invoices1", "shipments1",
    "returns1", "refunds1", "inventory", "product_catalog", "promotions",
    "payment_transactions1", "shipment_tracking1", "order_returns1",
    "supplier_data", "inventory_logs"
]
```

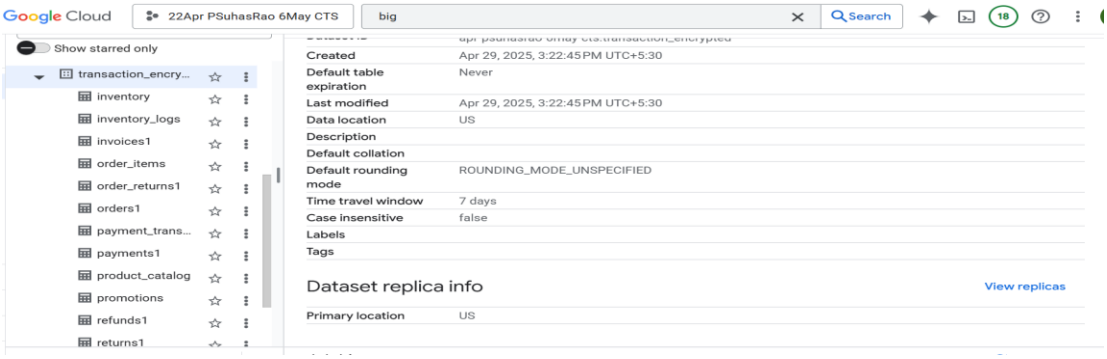
```
sensitive_fields = {
    "orders1": ["customer_id", "shipping_address", "billing_address", "payment_method"],
    "order_items": ["supplier_id", "supplier_name"],
    "payments1": ["customer_id", "payment_method", "transaction_id", "authorization_code", "account_number_masked", "payment_ip"],
    "invoices1": ["customer_id", "billing_address", "payment_method", "bank_name", "account_number_masked"],
    "shipments1": ["customer_id", "tracking_number", "origin", "destination"],
    "returns1": ["customer_id", "return_reason", "processed_by"],
    "refunds1": ["customer_id", "bank_name", "account_number_masked", "refund_reason"],
    "inventory": ["supplier_id", "batch_number"],
    "product_catalog": ["supplier_id"],
    "promotions": ["promotion_code"],
    "payment_transactions1": ["customer_id", "account_number_masked", "transaction_reference"],
    "shipment_tracking1": ["customer_id", "tracking_number", "delivery_address"],
    "order_returns1": ["customer_id", "pickup_address"],
    "supplier_data": ["supplier_name", "contact_name", "contact_email", "phone_number", "address"],
    "inventory_logs": ["logged_by", "approved_by"]
}
```

```
# Clients
storage_client = storage.Client()
kms_client = kms_v1.KeyManagementServiceClient()

# ThreadPoolExecutor for async operations
executor = ThreadPoolExecutor(max_workers=10)

def encrypt_field(value: str) -> str:
    """Encrypt the field value using KMS"""
    if not value:
        return value
    response = kms_client.encrypt(
        request={"name": KEY_NAME, "plaintext": value.encode("utf-8")}
    )
    return base64.b64encode(response.ciphertext).decode("utf-8")

def process_csv(blob, fields_to_encrypt):
    """Process CSV and encrypt sensitive fields"""
    data = blob.download_as_text()
    input_io = io.StringIO(data)
    output_io = io.StringIO()

    reader = csv.DictReader(input_io)
    writer = csv.DictWriter(output_io, fieldnames=reader.fieldnames)
    writer.writeheader()

    for row in reader:
        for field in fields_to_encrypt:
            if field in row and row[field]:
                row[field] = encrypt_field(row[field])
        writer.writerow(row)

    # Destination path
    file_name = blob.name.split("/")[-1]
    destination_blob_path = f"{DESTINATION_FOLDER}{file_name}"

    # Upload encrypted file to the destination folder
    encrypted_blob = storage_client.bucket(BUCKET_NAME).blob(destination_blob_path)
    encrypted_blob.upload_from_string(output_io.getvalue(), content_type="text/csv")
    print(f"Uploaded {filename} successfully.")

async def process_file_async(blob, fields_to_encrypt):
    """Wrapper for processing CSV asynchronously with a thread pool executor"""
    await asyncio.get_event_loop().run_in_executor(executor, process_csv, blob, fields_to_encrypt)

async def main():
    """Main function to process all files in parallel"""
    bucket = storage_client.bucket(BUCKET_NAME)
    blobs = bucket.list_blobs(prefix=SOURCE_FOLDER)

    tasks = []
    for blob in blobs:
        file_name = blob.name.split("/")[-1]
        if file_name in sensitive_fields:
            print(f"Encrypting: {file_name}")
            tasks.append(process_file_async(blob, sensitive_fields[file_name]))
        else:
            print(f"Skipping: {file_name} (no sensitive fields defined)")

    # Wait for all tasks to finish
    await asyncio.gather(*tasks)

# Run the main async function
if __name__ == "__main__":
    asyncio.run(main())
```

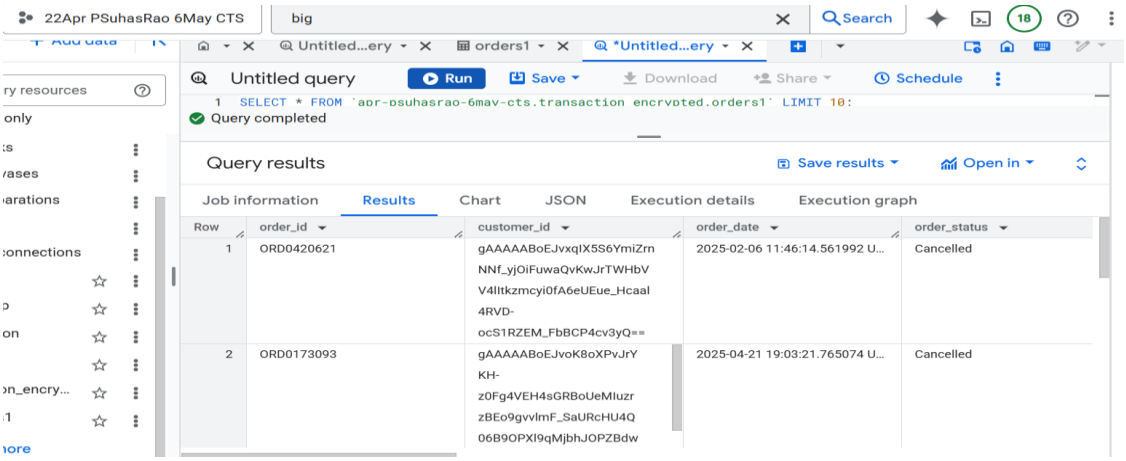
(22) Python Code for Encryption



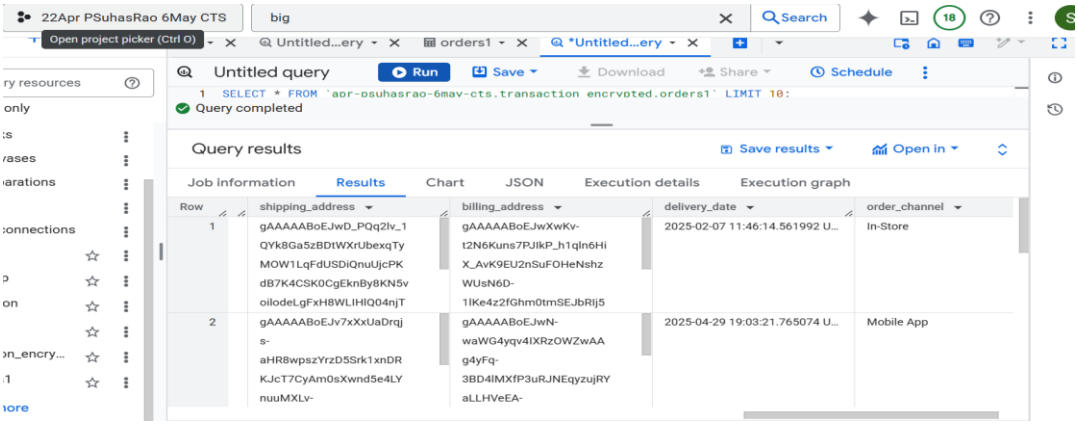
(23) Bucket in which Encrypted Data is stored



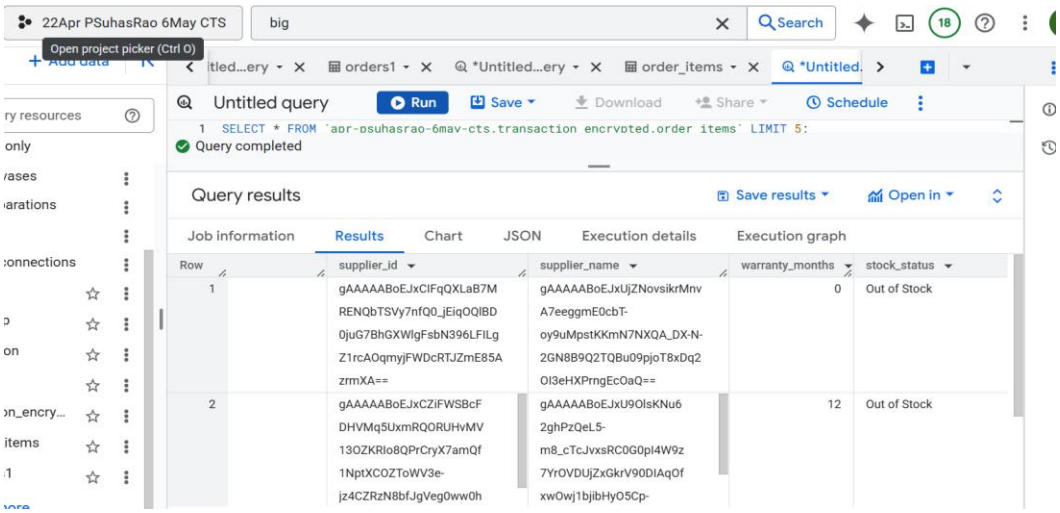
(24) Encrypted Transactional Data in BigQuery



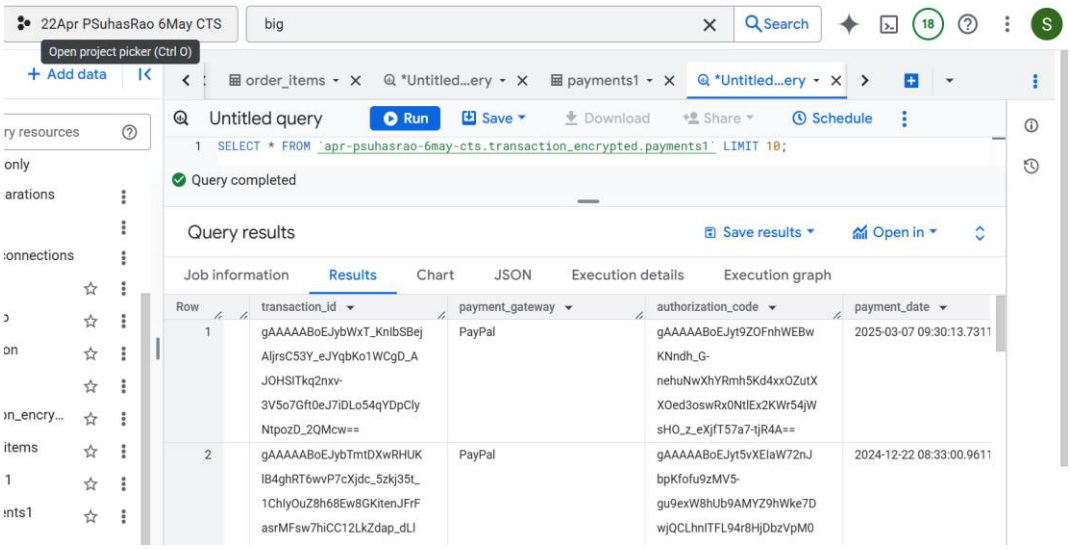
(25) Encrypted Orders Data



(26) Encrypted Orders Data



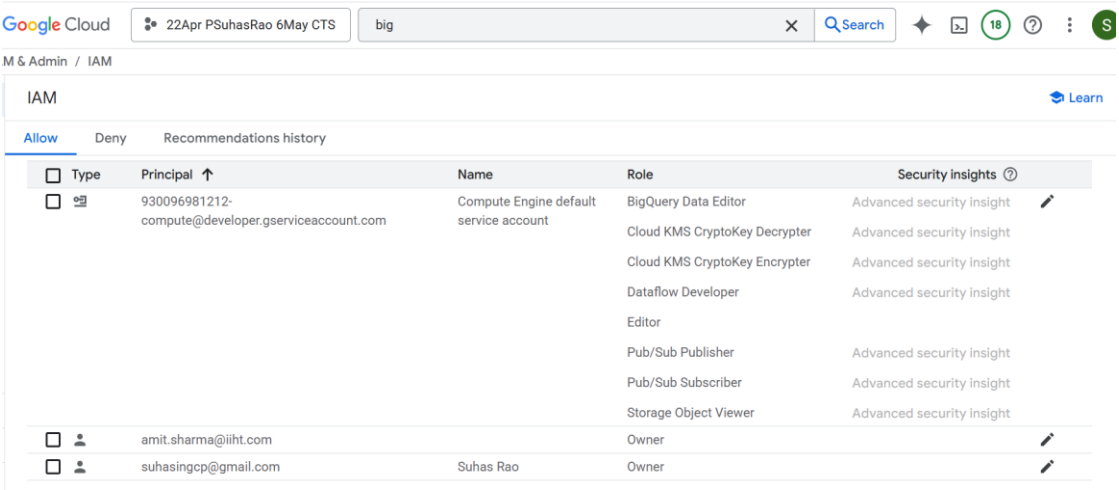
(27) Encrypted Order Items Data



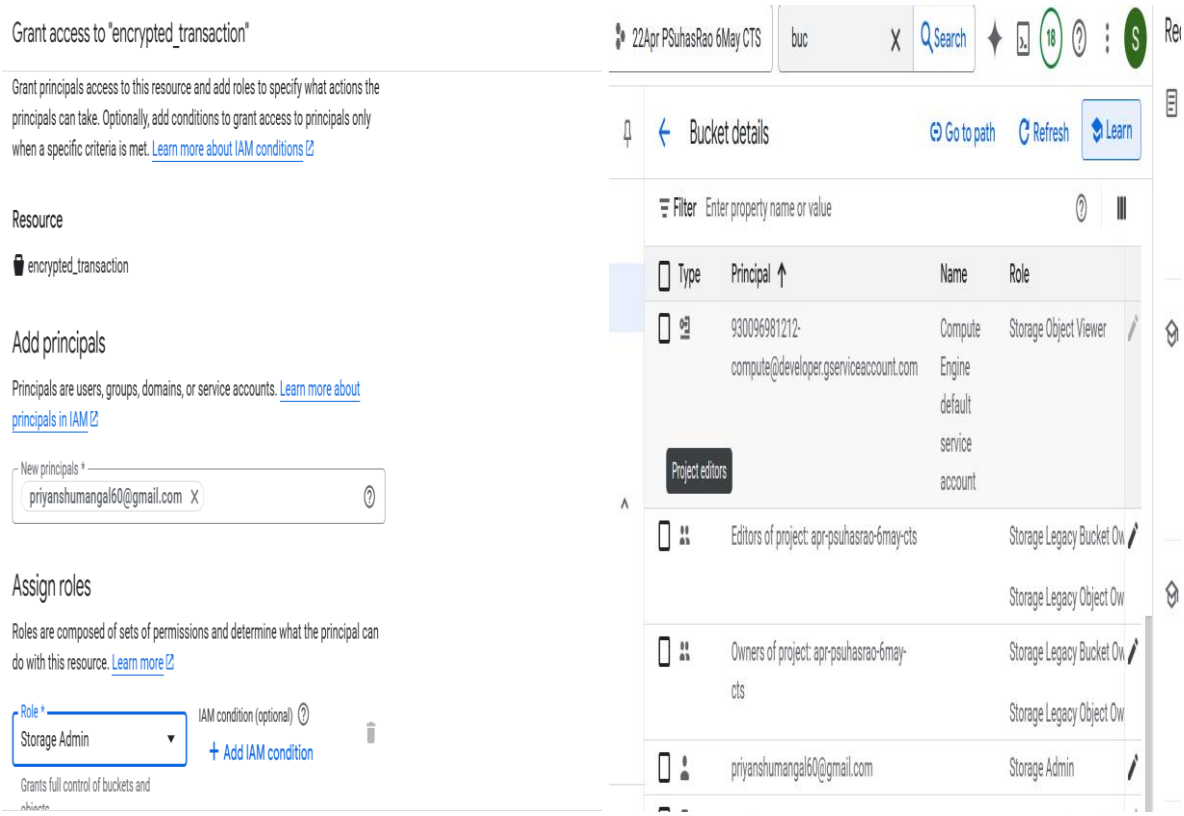
(28) Encrypted Payments Data

IAM and VPC

We were not granted permission to access the VPC at the organizational level.

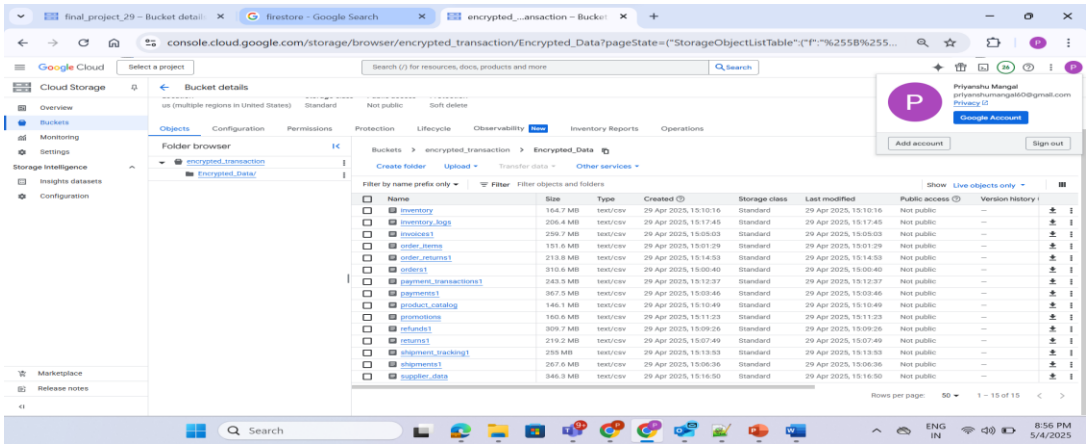


(29) Permissions Provided for Service Account



(30) Storage Admin Role

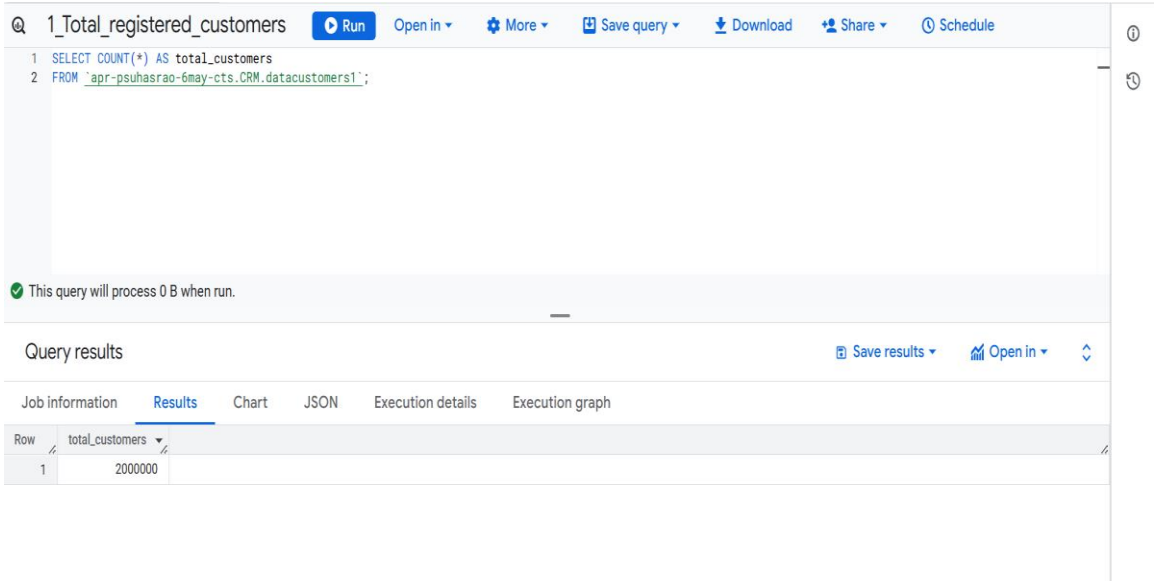
Provided priyanshumangal60@gmail.com a role of storage admin for the bucket encrypted_transaction so that he can access the sensitive data.



(31) Bucket Accessed By priyanshumangal60@gmail.com

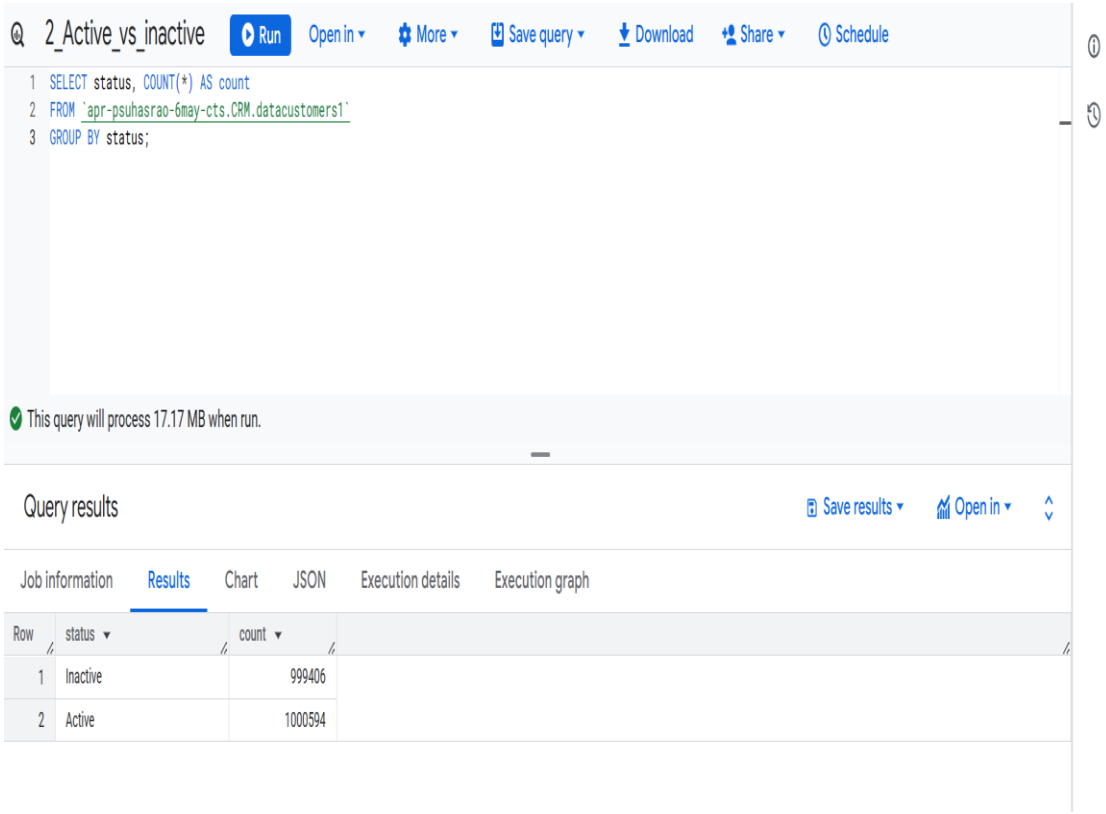
Key Performance Indicators (KPI's)

KPI 1: Total Registered Customers



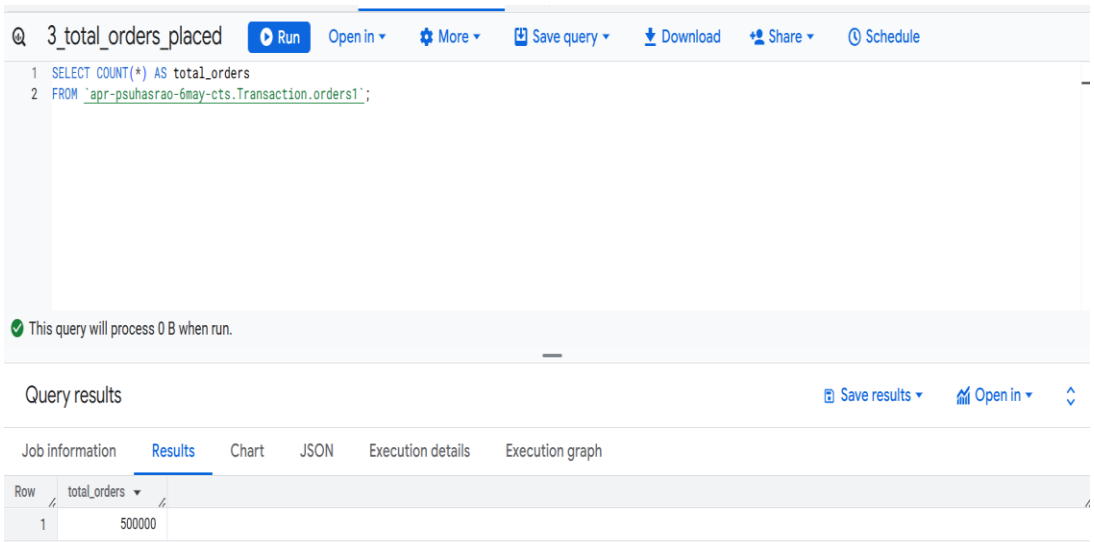
(32) KPI 1

KPI 2: Active vs Inactive Customers



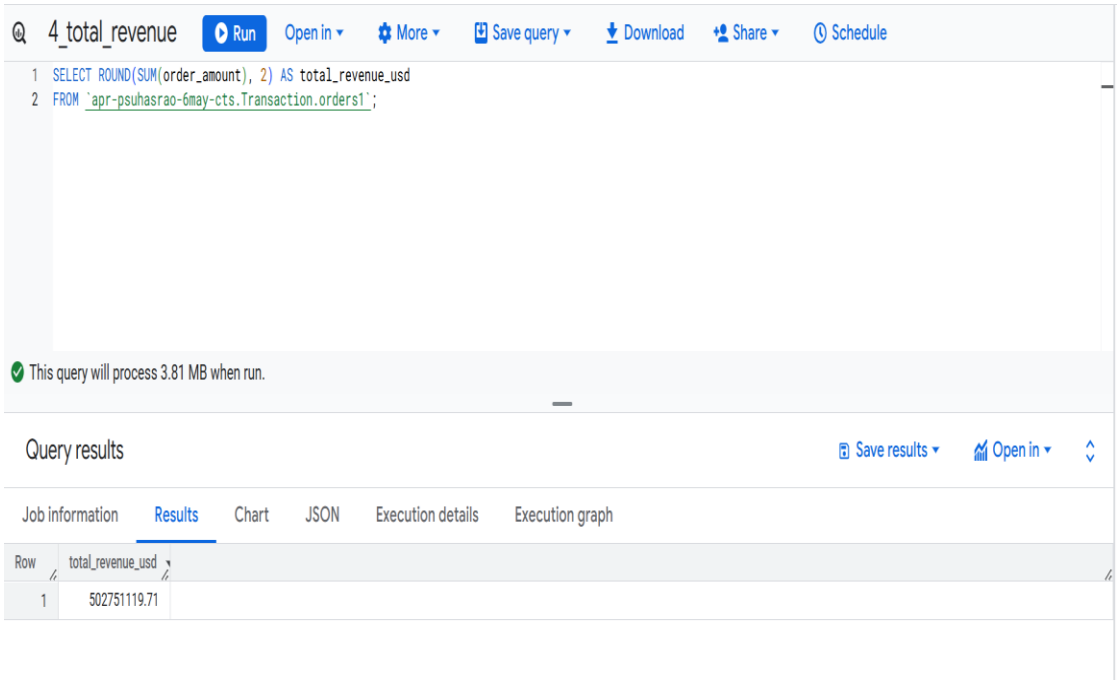
(33) KPI 2

KPI 3: Total Orders Placed



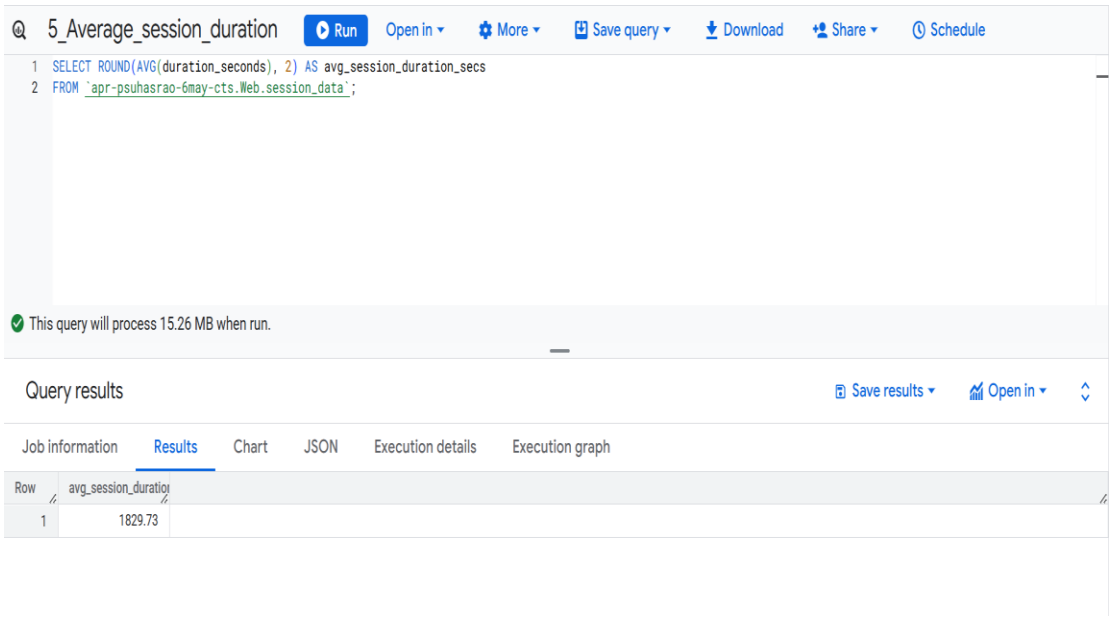
(34) KPI 3

KPI 4: Total Revenue



(35) KPI 4

KPI 5: Average Session Duration

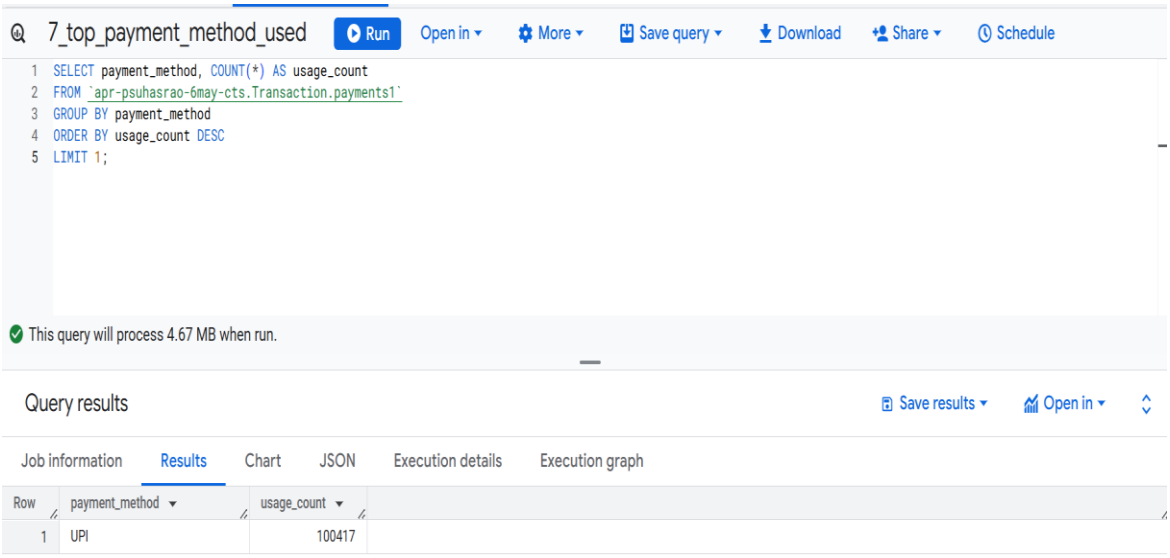


(36) KPI 5

KPI 6: Conversion Rate (Last 30 Days)

(37) KPI 6

KPI 7: Top Payment Method Used



(38) KPI 7

KPI 8: Returning Customer Rate (Last 30 Days)

(39) KPI 8

4 Dashboards

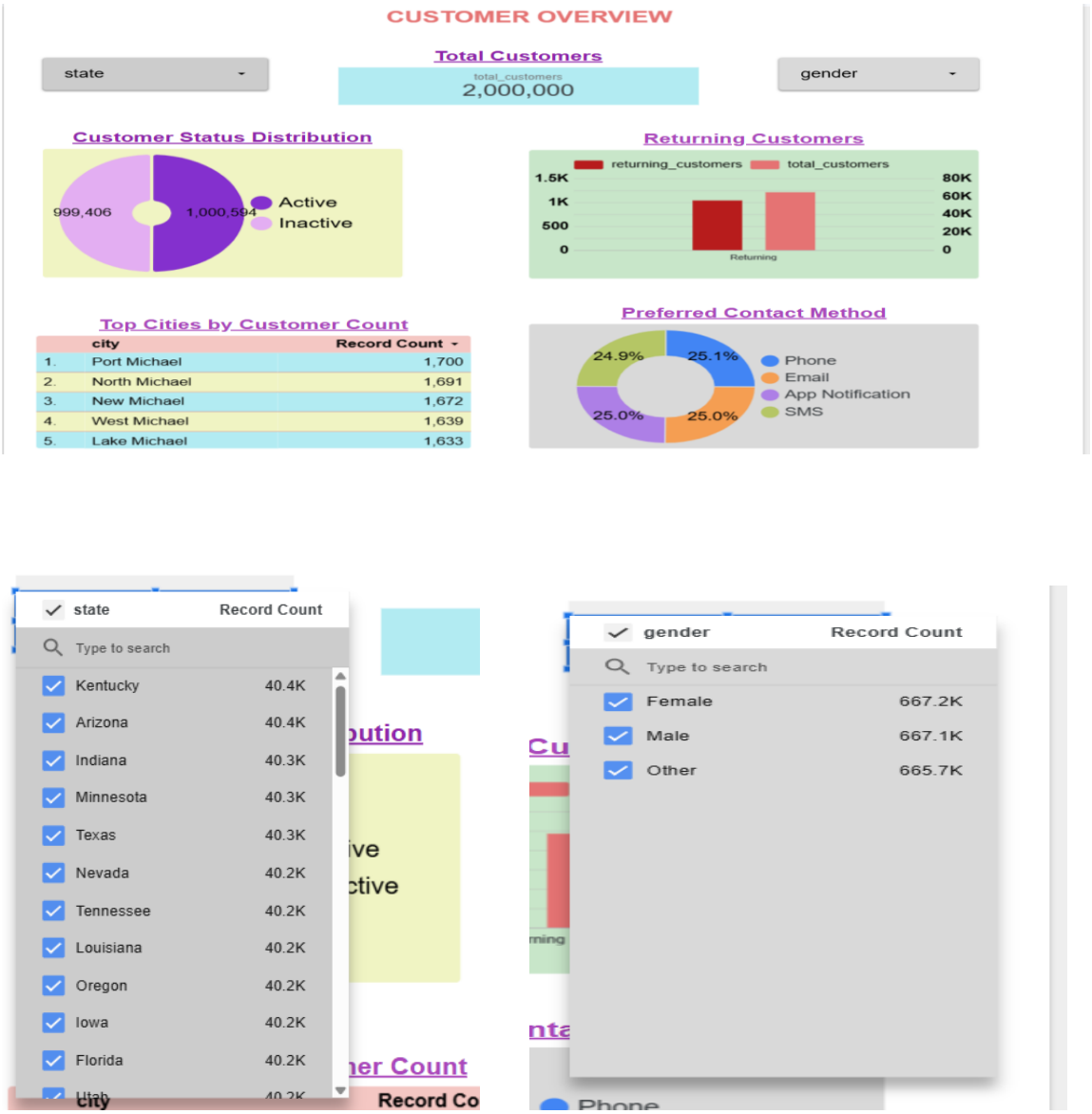
To visualize the processed data effectively, three dashboards were developed:

Customer Overview: Displays key customer metrics including total customers, active users, and customer segmentation.

Web Engagement Analytics: Highlights user interaction patterns such as page views, click events, browsers used etc.

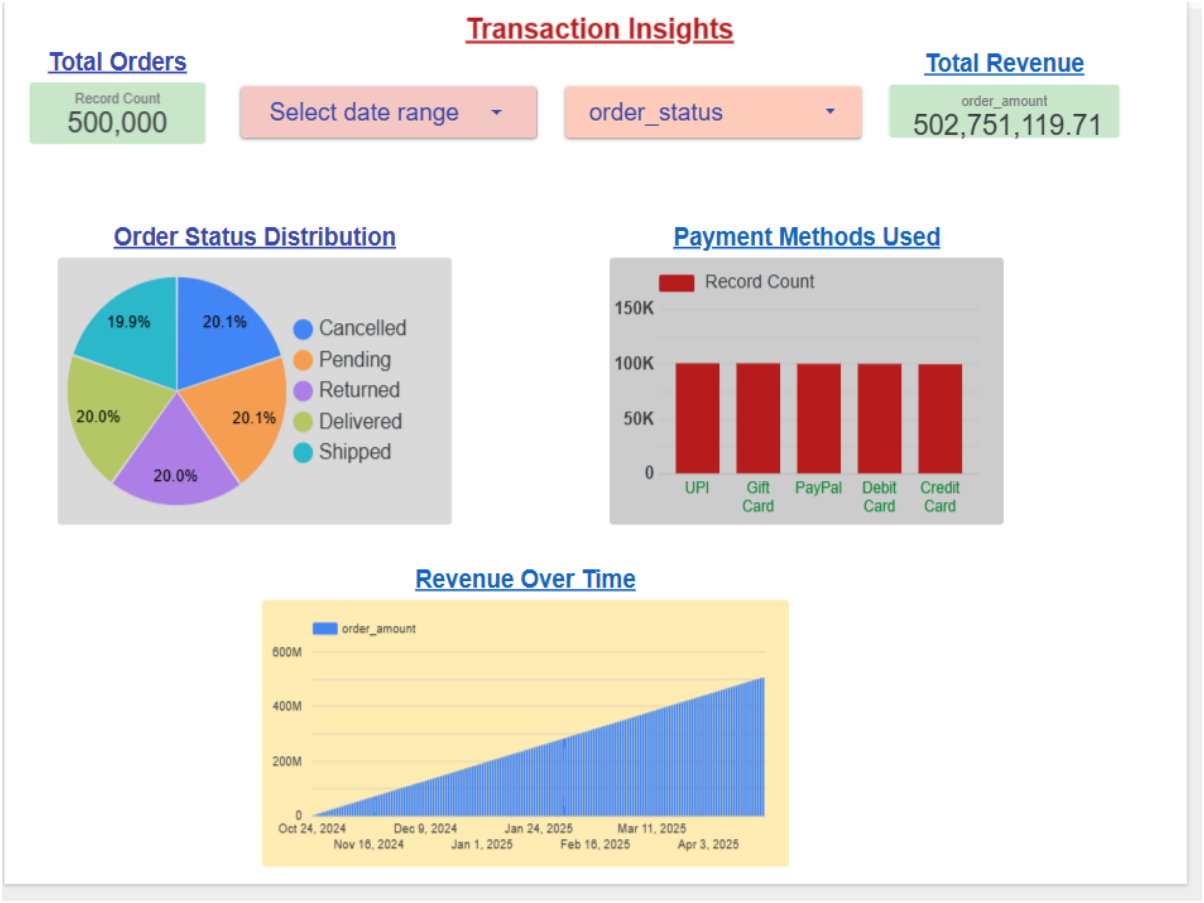
Transaction Insights: Provides visibility into order volumes, payment modes, and revenue trends to support financial analysis.

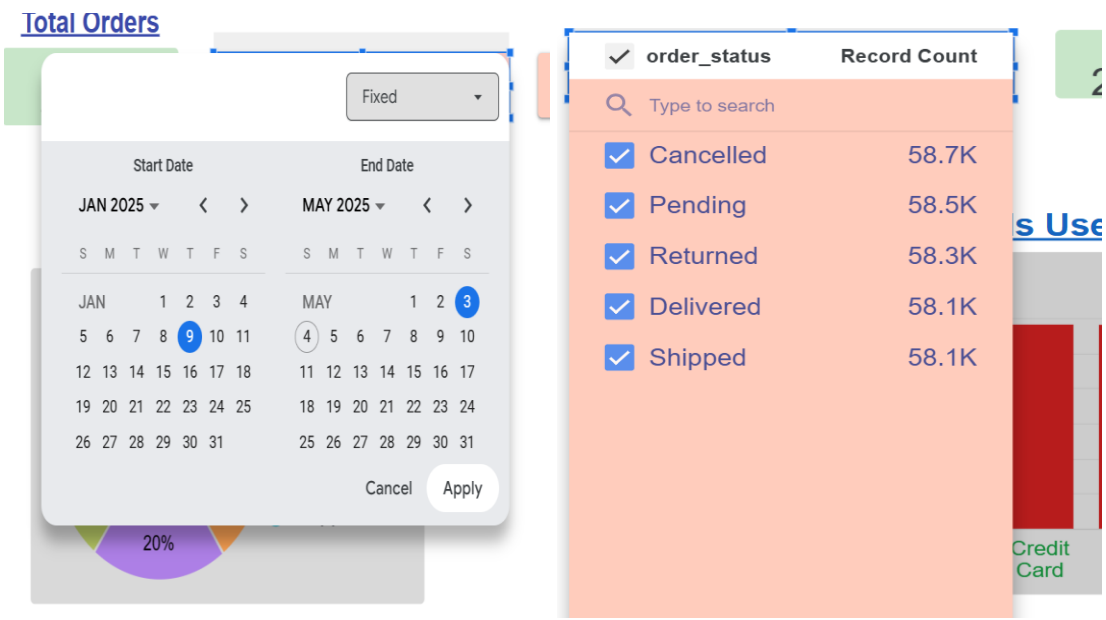
These dashboards help monitor performance and support data-driven decisions.



(40) Customer Overview Dashboard

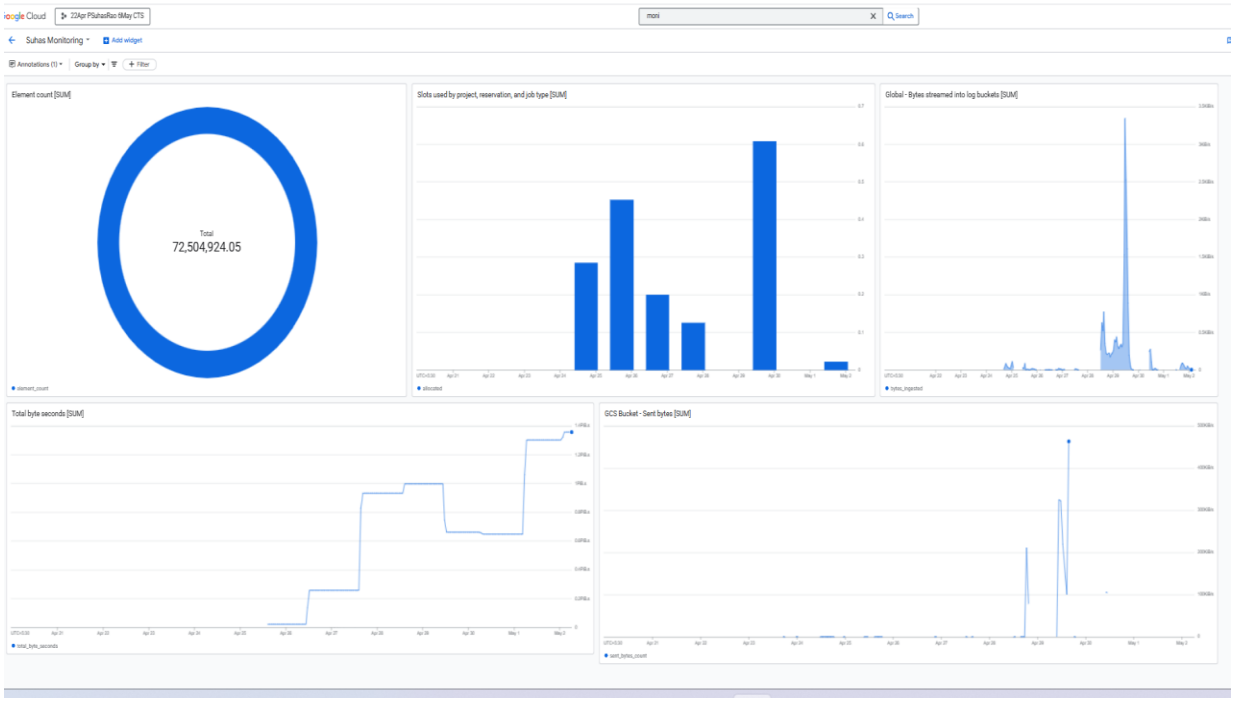
(41) Web Analytics Dashboard





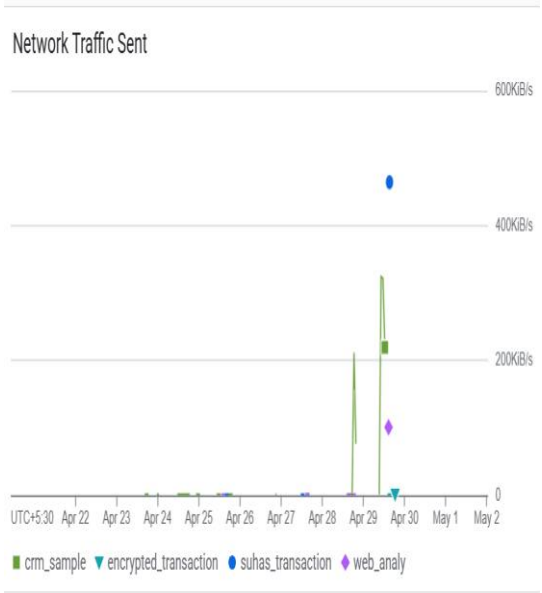
(42) Transaction Insights Dashboard

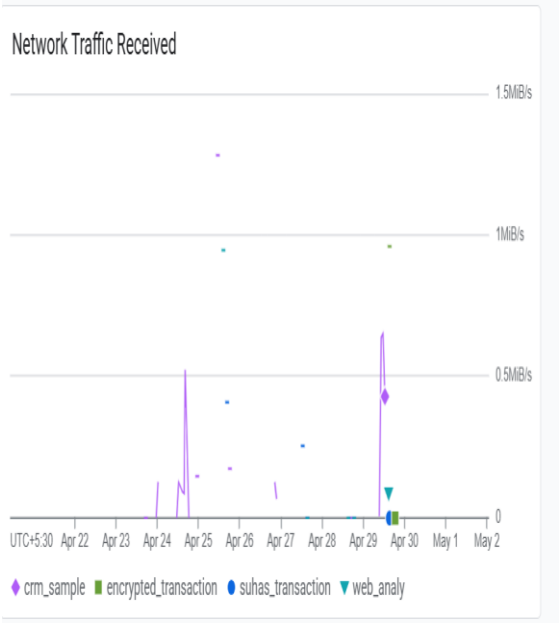
5 Monitoring and Logging

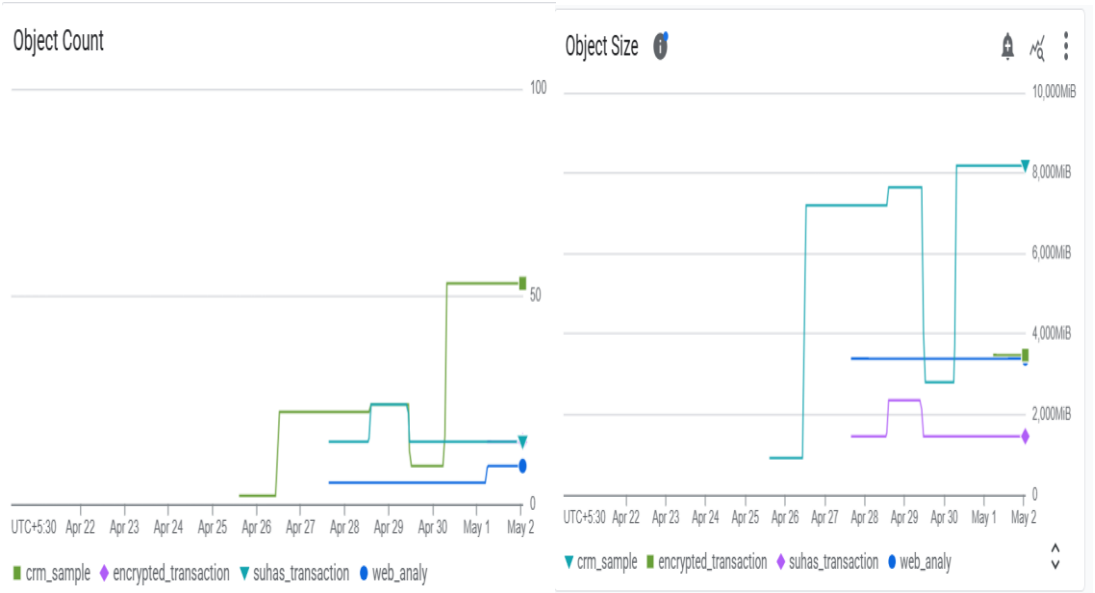


(43) Monitoring Dashboard

Google Cloud Storage Monitoring

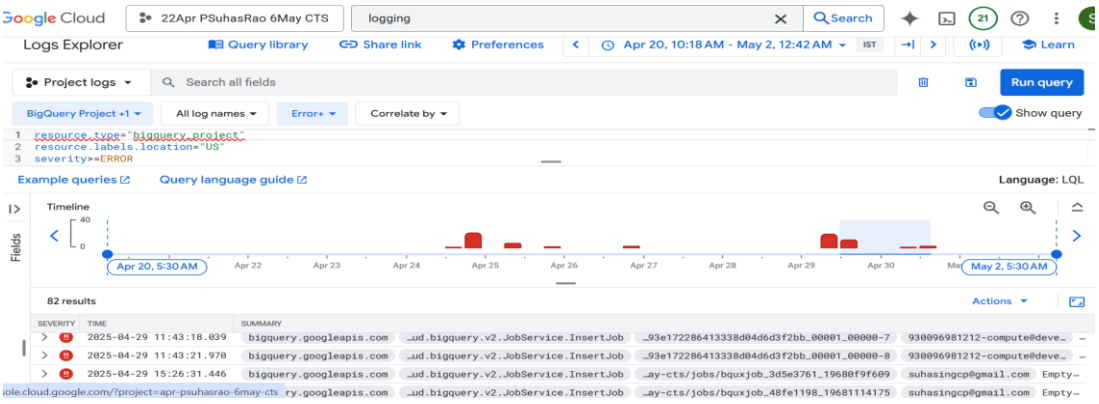


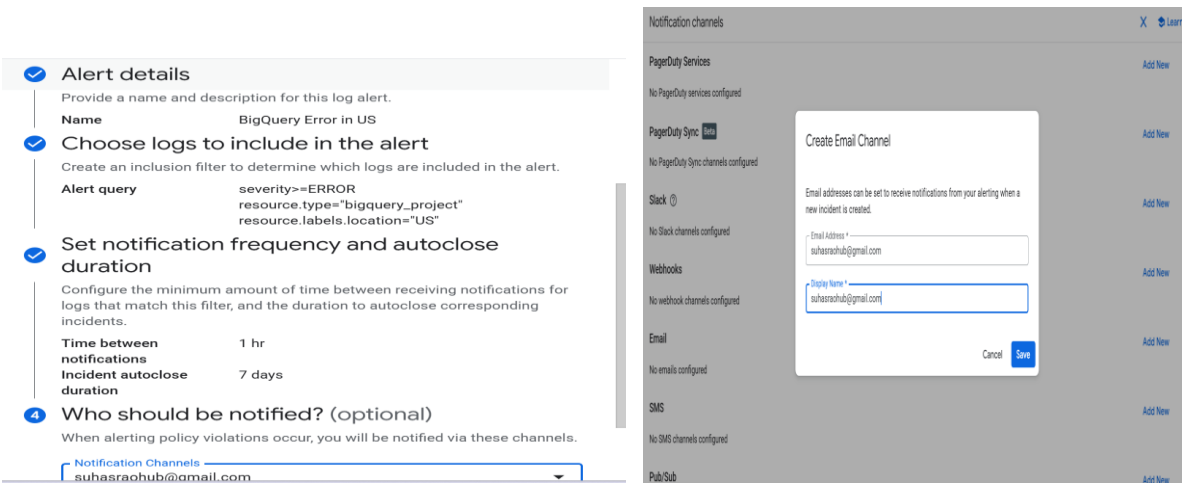




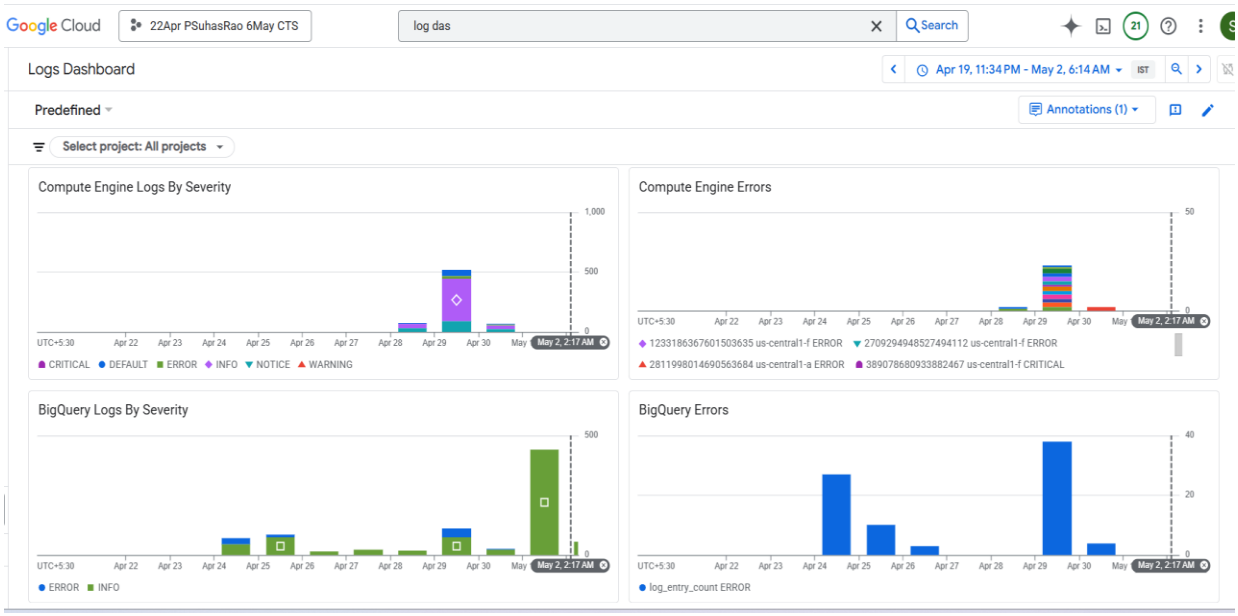
(44) Cloud Storage Monitoring

An alert will be sent to my email address suhasraohub@gmail.com whenever error is triggered in bigquery project.



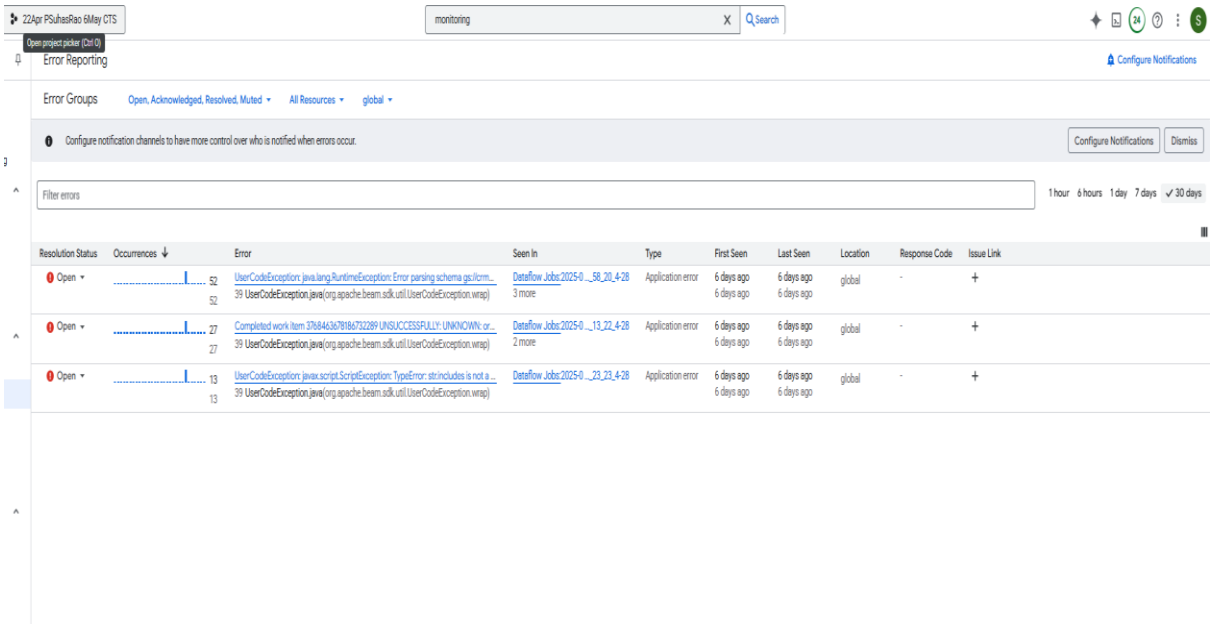


(45) Alerting about BigQuery Project Error



(46) Logging Dashboard

Error Reporting



(47) Error Reporting

6 Changed Log

Version	Date	Description
1.0	2025-05-04	Defined the purpose and scope of the document. Detailed system objectives and boundaries. Documented system scope, perspective, and architecture. Outlined functional requirements and covered implementation including monitoring and logging