Hands-on Task 1

You are building a student management system.

Requirements:

1. Create a Cloud SQL (MySQL or PostgreSQL) instance.
2. Create a database named college_db.
3. Create a table students with columns:

- student_id (Primary Key)
- name
- department
- marks

4. Insert at least 5 records.
5. Write SQL queries to:

- Fetch students with marks > 75
- Count students per department

6. Secure the database by:

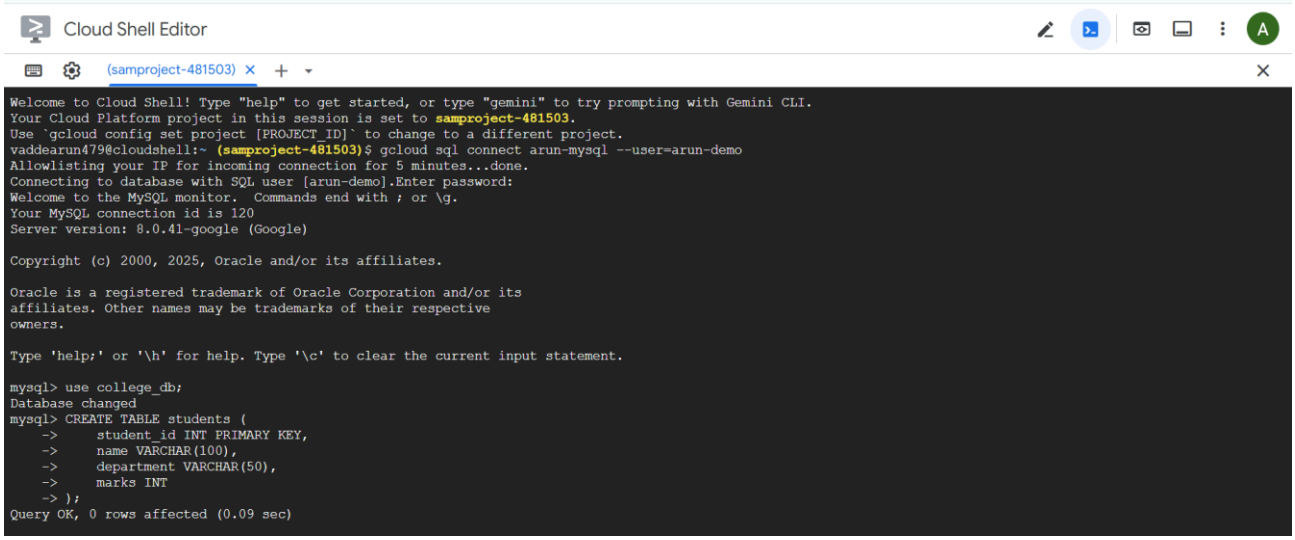
- Creating a read-only user
- Allowing access only from a specific IP

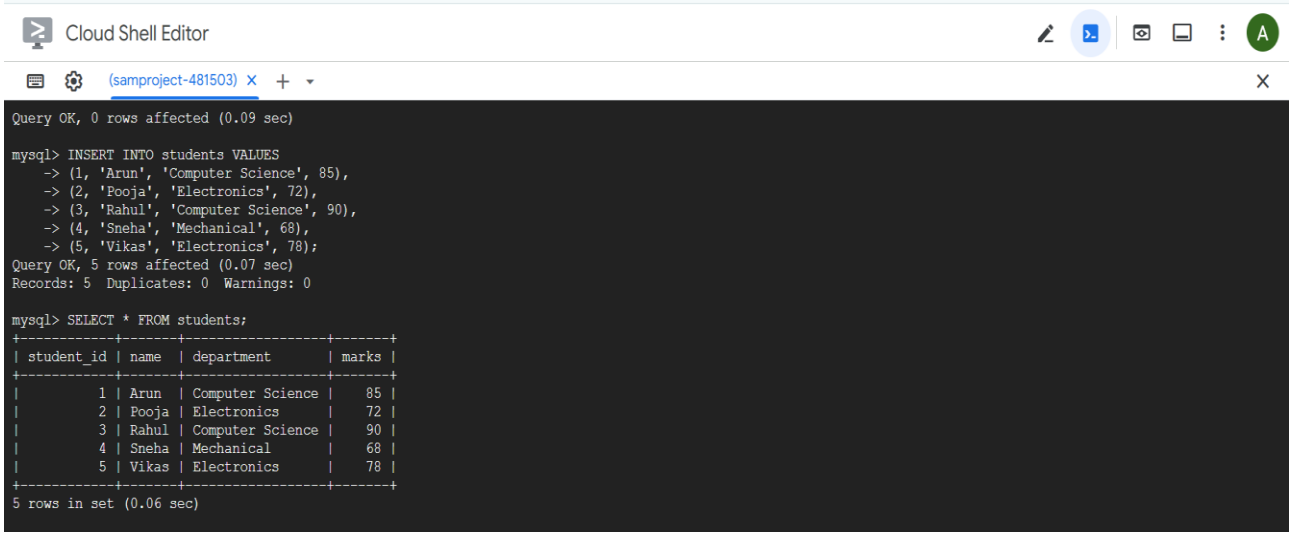
Expected Skills Tested

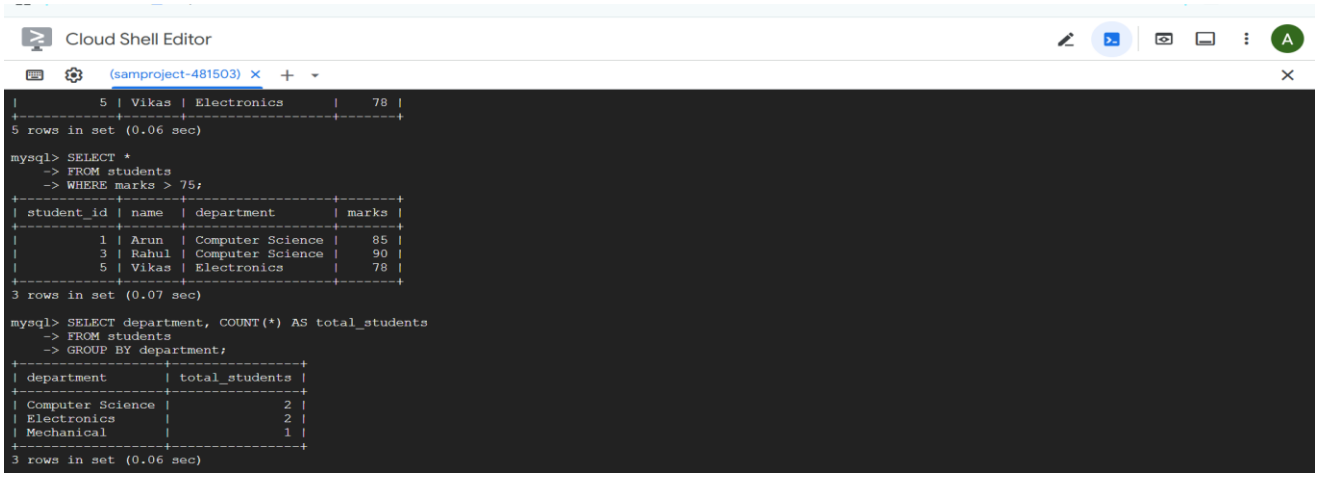
- Cloud SQL setup
- Basic SQL + permissions
- Networking & security basics

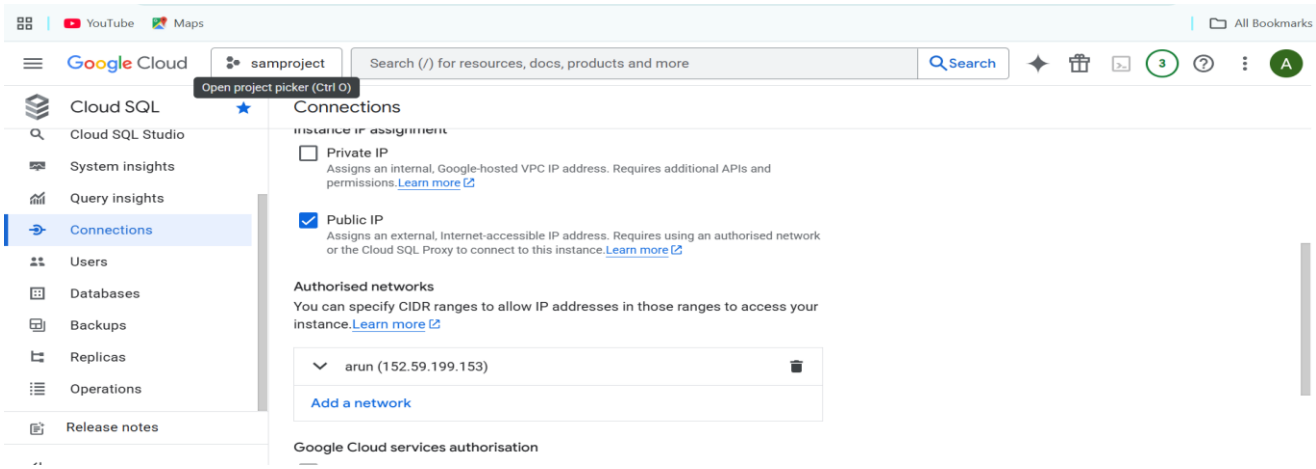
Synopsis – Student Management System using Cloud SQL

I have setup and use of Google Cloud SQL to manage student data securely. A Cloud SQL instance is created with a database named *college_db* and a *students* table to store academic details. Multiple student records are inserted to perform practical SQL operations. SQL queries are used to filter students based on marks and count students by department. Used Database security is implemented by creating a read-only user. And also usedNetwork access is restricted by allowing connections only from a specific IP address .


Cloud Shell Editor
(samproject-481503)
Welcome to Cloud Shell! Type "help" to get started, or type "gemini" to try prompting with Gemini CLI.
Your Cloud Platform project in this session is set to samproject-481503.
Use `gcloud config set project [PROJECT_ID]` to change to a different project.
vaddearun479@cloudshell:~ (samproject-481503)$ gcloud sql connect arun-mysql --user=arun-demo
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [arun-demo].Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 120
Server version: 8.0.41-google (Google)
Copyright (c) 2000, 2025, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> use college_db;
Database changed
mysql> CREATE TABLE students (
-> student_id INT PRIMARY KEY,
-> name VARCHAR(100),
-> department VARCHAR(50),
-> marks INT
->);
Query OK, 0 rows affected (0.09 sec)


Cloud Shell Editor
(samproject-481503)
Query OK, 0 rows affected (0.09 sec)
mysql> INSERT INTO students VALUES
-> (1, 'Arun', 'Computer Science', 85),
-> (2, 'Pooja', 'Electronics', 72),
-> (3, 'Rahul', 'Computer Science', 90),
-> (4, 'Sneha', 'Mechanical', 68),
-> (5, 'Vikas', 'Electronics', 78);
Query OK, 5 rows affected (0.07 sec)
Records: 5 Duplicates: 0 Warnings: 0
mysql> SELECT * FROM students;
+------------+-------+------------------+-------+
| student_id | name | department | marks |
+------------+-------+------------------+-------+
1	Arun	Computer Science	85
2	Pooja	Electronics	72
3	Rahul	Computer Science	90
4	Sneha	Mechanical	68
5	Vikas	Electronics	78
+------------+-------+------------------+-------+
5 rows in set (0.06 sec)


Cloud Shell Editor
(samproject-481503)
| 5 | Vikas | Electronics | 78 |
+------------+-------+------------------+-------+
5 rows in set (0.06 sec)
mysql> SELECT *
-> FROM students
-> WHERE marks > 75;
+------------+-------+------------------+-------+
| student_id | name | department | marks |
+------------+-------+------------------+-------+
1	Arun	Computer Science	85
3	Rahul	Computer Science	90
5	Vikas	Electronics	78
+------------+-------+------------------+-------+
3 rows in set (0.07 sec)
mysql> SELECT department, COUNT(*) AS total_students
-> FROM students
-> GROUP BY department;
+------------------+----------------+
| department | total_students |
+------------------+----------------+
Computer Science	2
Electronics	2
Mechanical	1
+------------------+----------------+
3 rows in set (0.06 sec)



Hands-on Task 2

You are creating a real-time chat application backend.

Requirements:

1. Enable Firestore (Native mode).
2. Create a collection chats.
3. Each document should store:

- sender
- receiver
- message
- timestamp

4. Perform the following:

- Insert at least 10 chat messages
- Query all messages between two users
- Sort messages by timestamp

5. Create a Firestore security rule:

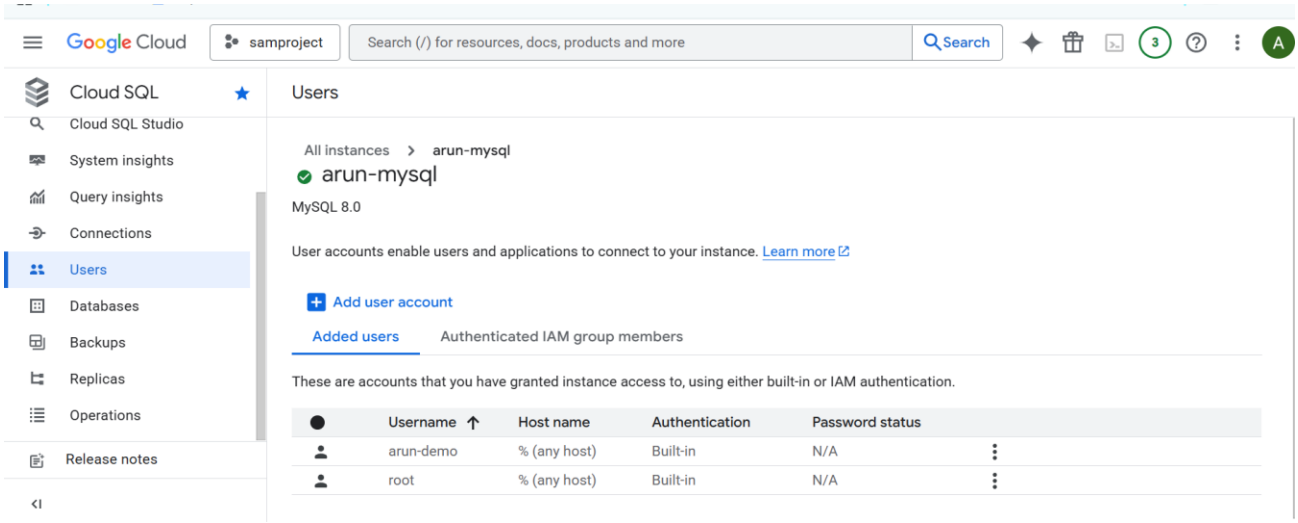
- Only authenticated users can read/write
- Users can read only their own chats

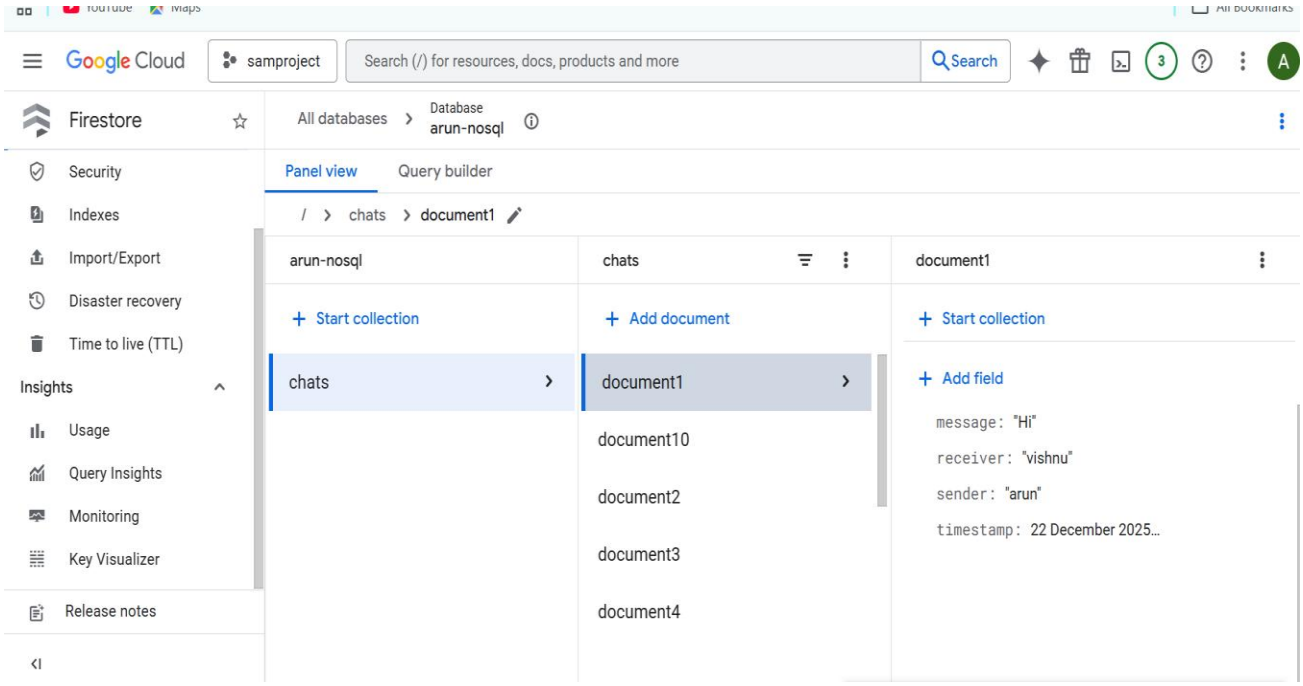
Expected Skills Tested

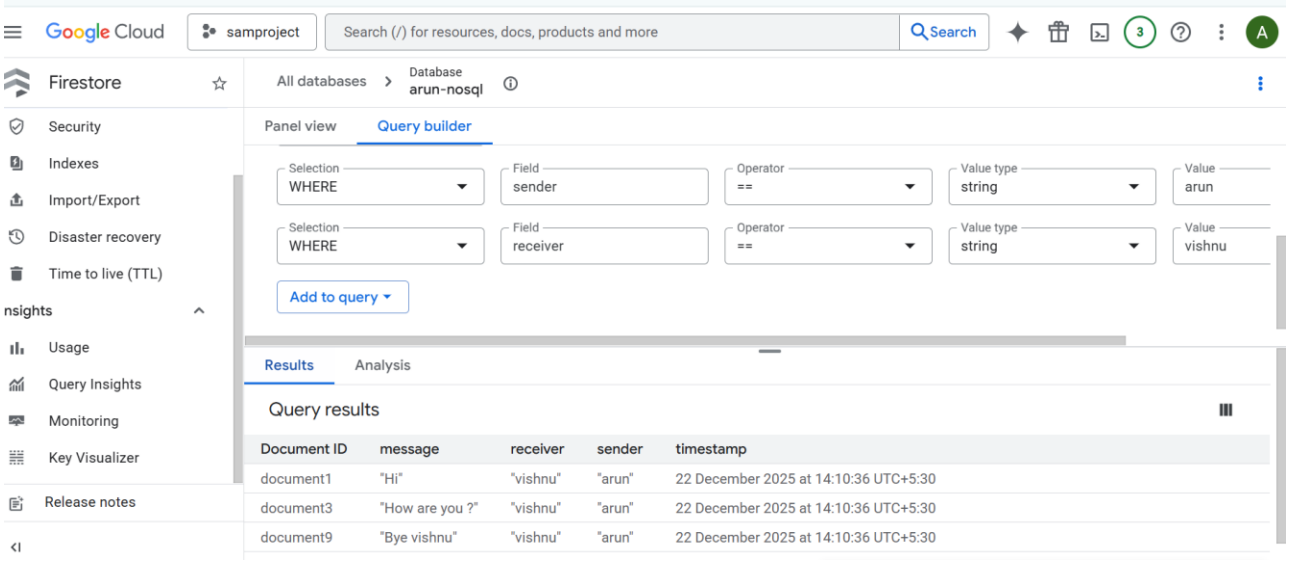
- Firestore data modeling
- NoSQL querying
- Security rules

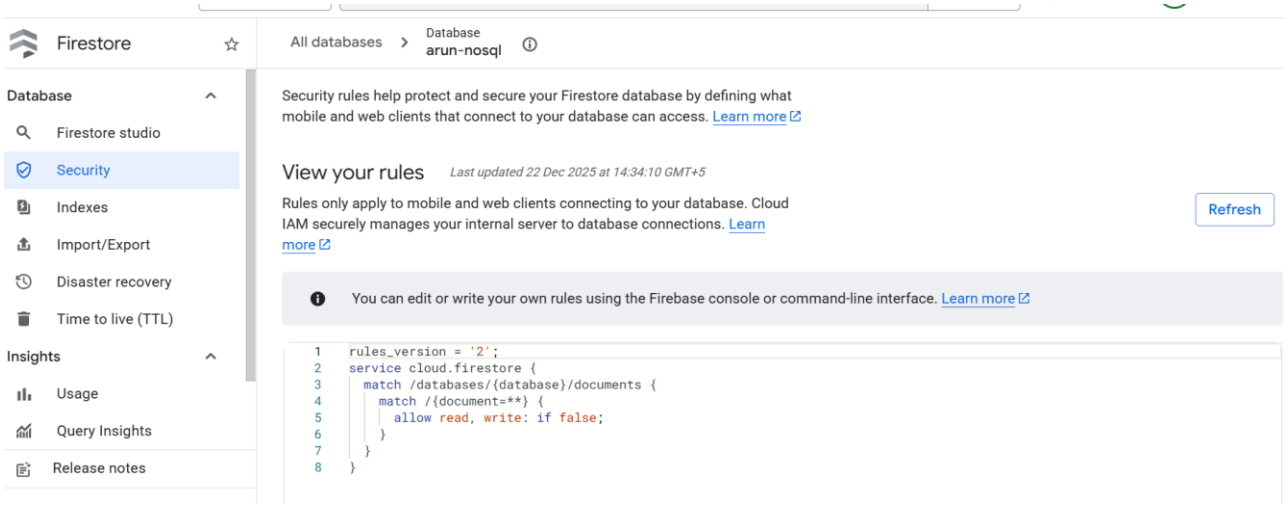
Synopsis – Real-Time Chat Application using Firestore.

I have used in this project backend for a real-time chat application using Firestore in Native mode. A *chats* collection is created to store messages with sender, receiver, message content, and timestamp. Multiple chat documents are inserted to simulate real user communication. Firestore queries are used to retrieve messages between two users and display them in time order. Security rules ensure that only authenticated users can access the database. Users are restricted to reading and writing only their own chat messages, improving data privacy and security.




Google Cloud
samproject
Search (/) for resources, docs, products and more
Search
Firestore
All databases
Database
arun-nosql
Security
Indexes
Import/Export
Disaster recovery
Time to live (TTL)
Insights
Usage
Query Insights
Monitoring
Key Visualizer
Release notes
Panel view
Query builder
/ chats document1
arun-nosql
chats
document1
Start collection
Add document
Start collection
Add field
chats
document1
document10
document2
document3
document4
message: "Hi"
receiver: "vishnu"
sender: "arun"
timestamp: 22 December 2025...


Google Cloud
samproject
Search (/) for resources, docs, products and more
Search
Firestore
All databases
Database
arun-nosql
Security
Indexes
Import/Export
Disaster recovery
Time to live (TTL)
Usage
Query Insights
Monitoring
Key Visualizer
Release notes
Panel view
Query builder
Selection
WHERE
Field
sender
Operator
==
Value type
string
Value
arun
Selection
WHERE
Field
receiver
Operator
==
Value type
string
Value
vishnu
Add to query
Results
Analysis
Query results
Document ID
message
receiver
sender
timestamp
document1
"Hi"
"vishnu"
"arun"
22 December 2025 at 14:10:36 UTC+5:30
document3
"How are you ?"
"vishnu"
"arun"
22 December 2025 at 14:10:36 UTC+5:30
document9
"Bye vishnu"
"vishnu"
"arun"
22 December 2025 at 14:10:36 UTC+5:30



Hands-on Task 3

You are designing an e-commerce platform.

Requirements:

1. Use Cloud SQL to store:

- Orders
- Payments

2. Use Firestore or Bigtable to store:

- User activity logs
- Clickstream data

3. Implement:

- One SQL query to fetch total orders per user
- One NoSQL query to fetch last 50 user activities

4. Explain (practically):

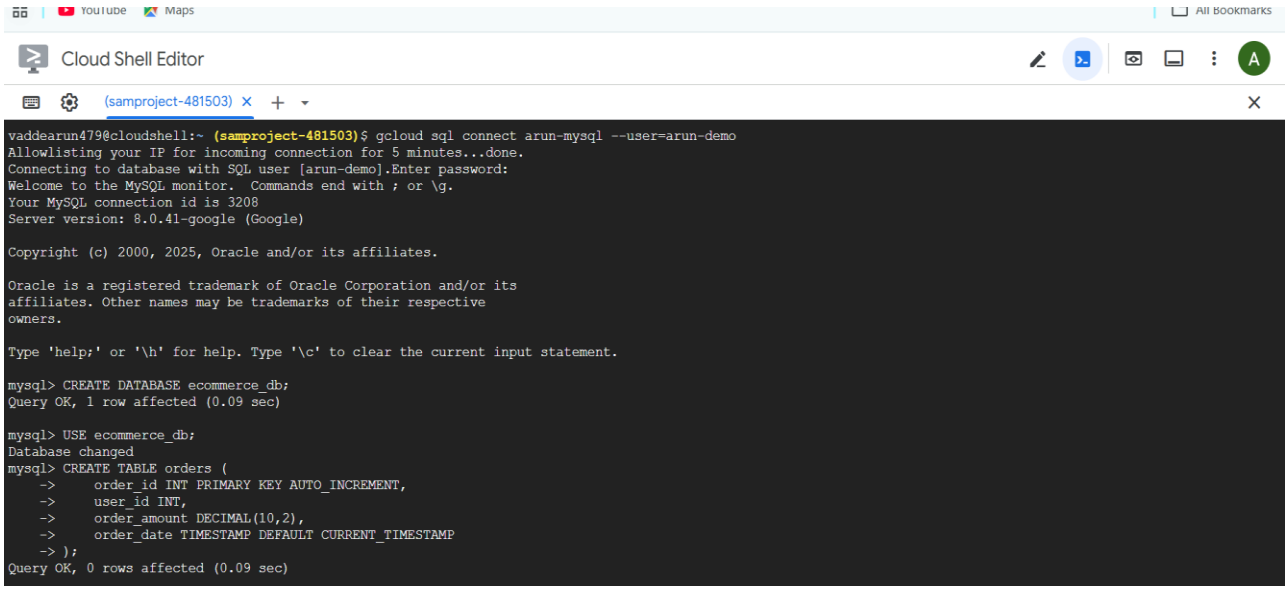
- Why SQL is chosen for transactions
- Why NoSQL is chosen for logs

Expected Skills Tested

- GCP service selection
- Data modeling
- Real-world architecture decisions

Synopsis – E-Commerce Platform Data Architecture

I have used in this project platform using both SQL and NoSQL databases on GCP. Cloud SQL is used to store orders and payment details to ensure data consistency and reliable transactions. SQL queries are applied to calculate the total number of orders placed by each user. Firestore or Bigtable is used to store user activity logs and clickstream data due to high data volume and fast writes. A NoSQL query retrieves the latest 50 user activities efficiently. SQL is chosen for structured transactional data, while NoSQL is used for unstructured and high-frequency logging data.



```
mysql> CREATE DATABASE ecommerce_db;
Query OK, 1 row affected (0.09 sec)

mysql> USE ecommerce_db;
Database changed
mysql> CREATE TABLE orders (
    ->     order_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     user_id INT,
    ->     order_amount DECIMAL(10,2),
    ->     order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO orders (user_id, order_amount) VALUES
    -> (101, 1500),
    -> (101, 2200),
    -> (102, 1800),
    -> (103, 999),
    -> (102, 4500);
Query OK, 5 rows affected (0.07 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

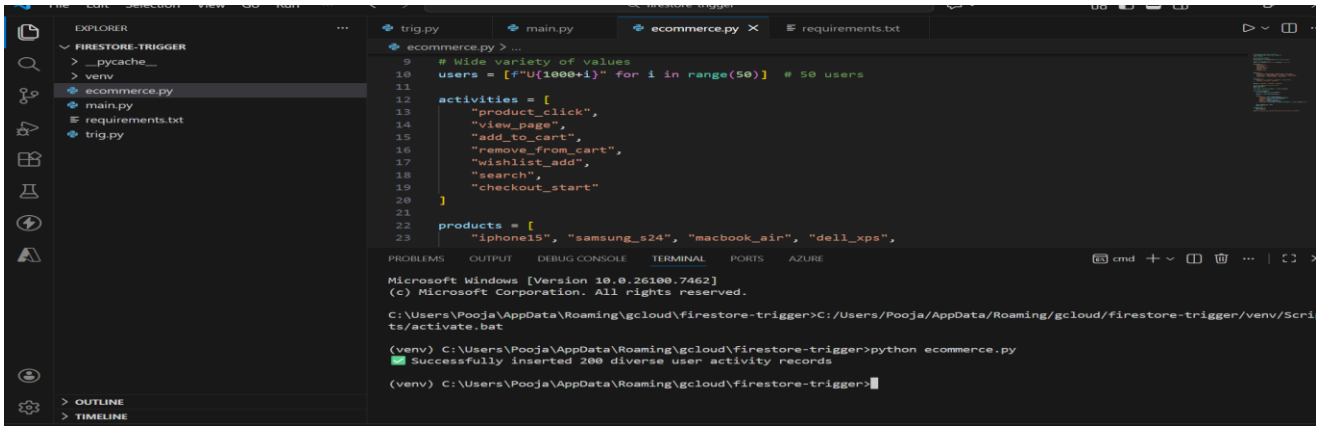
Cloud Shell Editor

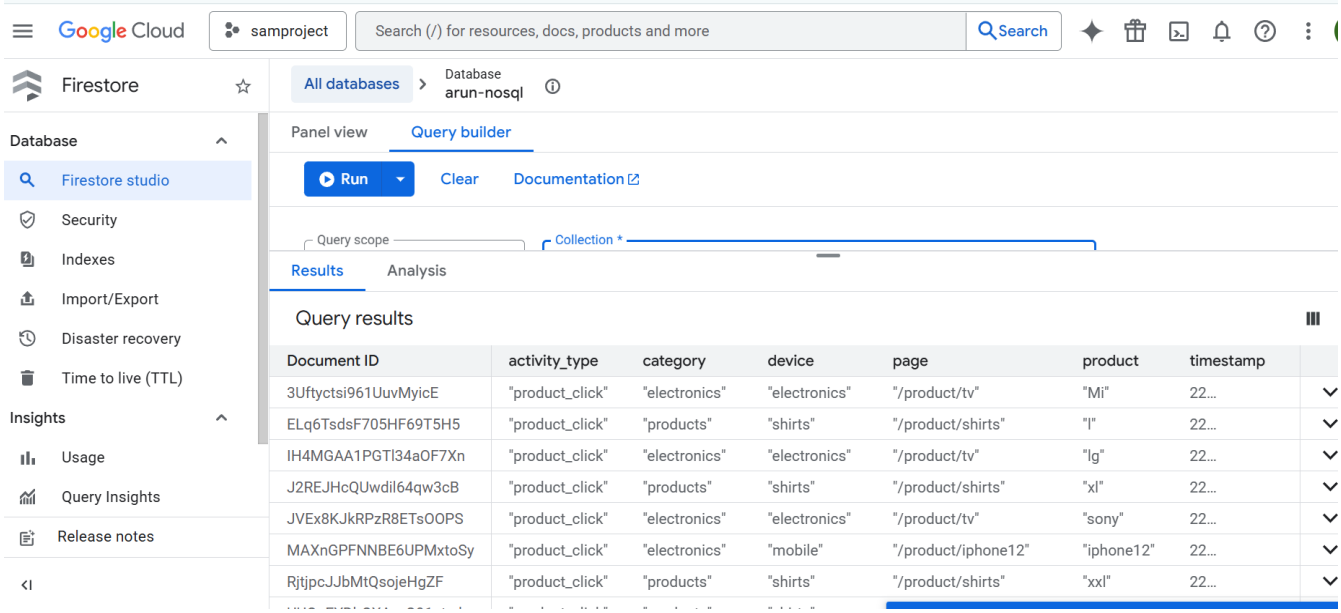
(samproject-481503)

```
mysql> CREATE TABLE payments (
    ->     payment_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     order_id INT,
    ->     payment_status VARCHAR(20),
    ->     payment_method VARCHAR(20),
    ->     payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ->     FOREIGN KEY (order_id) REFERENCES orders(order_id)
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO payments (order_id, payment_status, payment_method) VALUES
    -> (1, 'SUCCESS', 'UPI'),
    -> (2, 'SUCCESS', 'CARD'),
    -> (3, 'FAILED', 'CARD'),
    -> (4, 'SUCCESS', 'NETBANKING'),
    -> (5, 'SUCCESS', 'UPI');
Query OK, 5 rows affected (0.08 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT user_id, COUNT(*) AS total_orders
    -> FROM orders
    -> GROUP BY user_id;
+---------+--------------+
| user_id | total_orders |
+---------+--------------+
|     101 |            2 |
|     102 |            2 |
|     103 |            1 |
+---------+--------------+
```





Explain : • Why SQL is chosen for transactions

• Why NoSQL is chosen for logs

SQL is chosen for transactions because it ensures data accuracy using ACID properties, which are critical for orders and payments.
It supports strong consistency, relationships, and rollback in case a transaction fails.
NoSQL is chosen for logs because log data is high-volume, fast-growing, and does not require complex relationships.
It allows quick writes and horizontal scaling without performance issues.