Student Management System using Google Cloud SQL (MySQL)

Requirements Covered:

1. Cloud SQL instance creation

2. Database creation (college_db / Management_system)

3. Students table creation

4. Data insertion

5. SQL queries

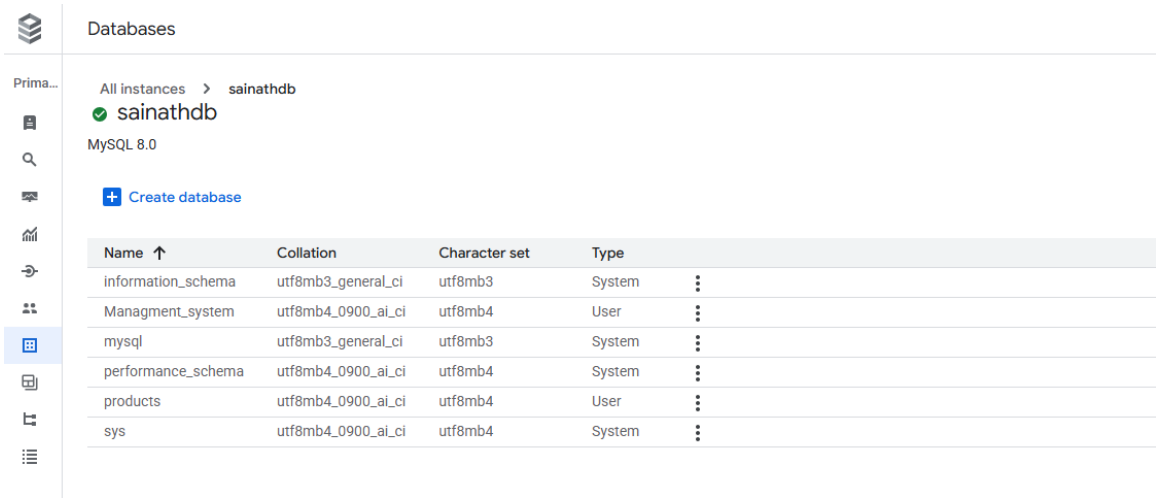
6. Database security (read-only user + IP restriction)

Step 1: Create Cloud SQL MySQL Instance

- Created Cloud SQL instance named 'sainathdb'

- Database version: MySQL 8.0

- Public IP enabled



Step 2: Create Database

Database Name: Management_system

Step 3: Create students Table

Step 4: Insert Records

```
mysql> use Managment_system;
Database changed
mysql> CREATE TABLE students (
    ->     student_id INT PRIMARY KEY,
    ->     name VARCHAR(100),
    ->     department VARCHAR(50),
    ->     marks INT
    -> );
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO students (student_id, name, department, marks) VALUES
    -> (1, 'Rahul', 'Computer Science', 85),
    -> (2, 'Anita', 'Information Technology', 78),
    -> (3, 'Suresh', 'Electronics', 88),
    -> (4, 'Priya', 'Mechanical', 72),
    -> (5, 'Kiran', 'Civil', 69),
    -> (6, 'Neha', 'Computer Science', 91),
    -> (7, 'Arjun', 'Electrical', 76),
    -> (8, 'Pooja', 'Information Technology', 83),
    -> (9, 'Vikram', 'Mechanical', 67),
    -> (10, 'Sneha', 'Electronics', 89);
Query OK, 10 rows affected (0.28 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

Query 1: Students with marks > 75

Query 2: Count students per department

SELECT department, COUNT(*) FROM students GROUP BY department;

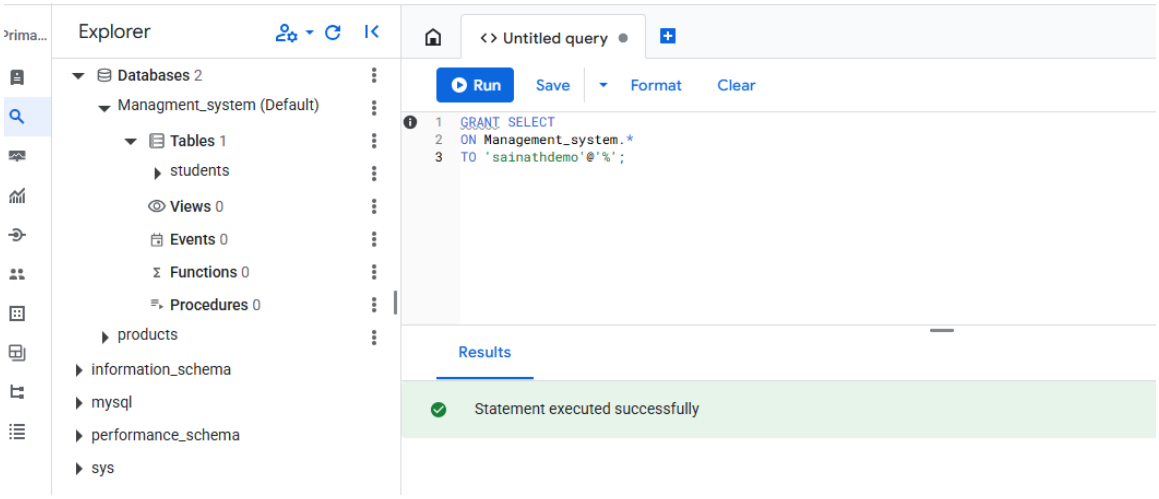
```
(venv) PS C:\Users\reddy\Music\HSBC\gcp> python app.py
Connected Successfully at: 2025-12-22 08:12:37

Students with marks > 75:
(1, 'Rahul', 'Computer Science', 85)
(2, 'Anita', 'Information Technology', 78)
(3, 'Suresh', 'Electronics', 88)
(6, 'Neha', 'Computer Science', 91)
(7, 'Arjun', 'Electrical', 76)
(8, 'Pooja', 'Information Technology', 83)
(10, 'Sneha', 'Electronics', 89)

Count of students per department:
('Computer Science', 2)
('Information Technology', 2)
('Electronics', 2)
('Mechanical', 2)
('Civil', 1)
('Electrical', 1)
(venv) PS C:\Users\reddy\Music\HSBC\gcp>
```

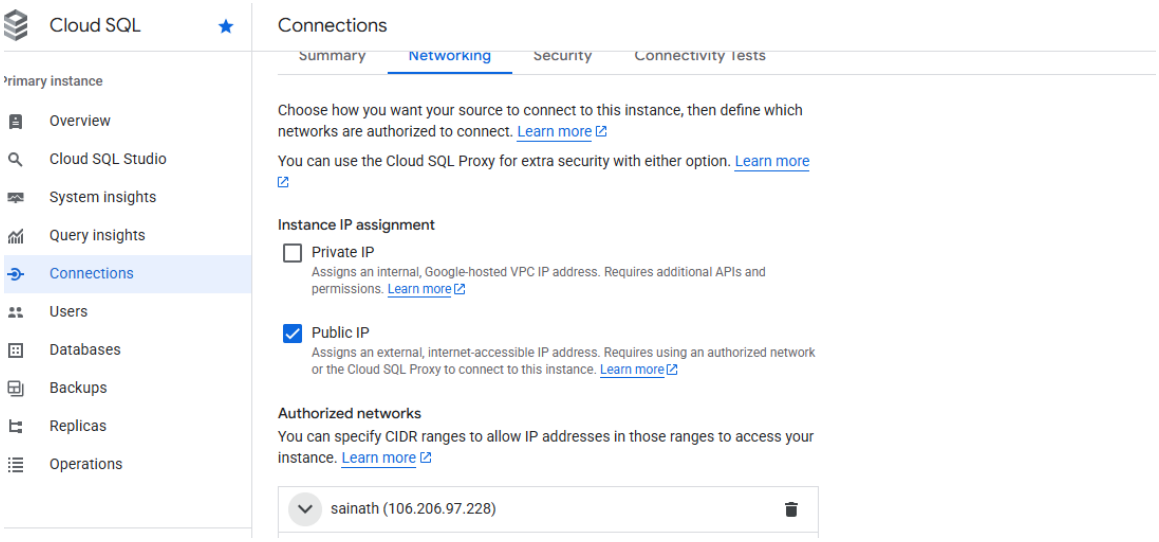
Read-only User:

Username: sainathdemo



IP Restriction:

Authorized Network: 106.206.97.228/32



Real-Time Chat Application Backend using Firestore

Requirements Covered:

1. Enable Firestore (Native mode)

2. Create chats collection

3. Store sender, receiver, message, timestamp

4. Insert chat messages

5. Query chats between users

6. Sort messages by timestamp

7. Secure Firestore using rules

Step 1: Enable Firestore (Native Mode)

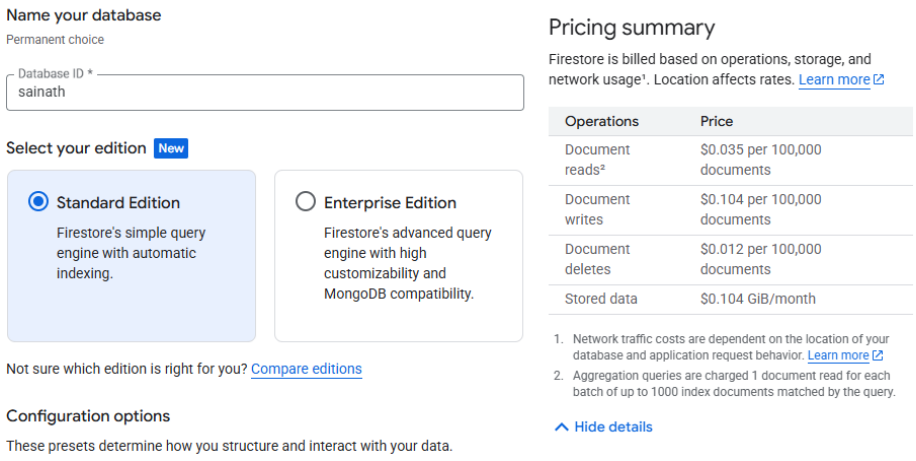
1. Open Google Cloud Console

2. Navigate to Firestore

3. Click Create Database

4. Select Native Mode

5. Choose region and create



Step 2: Create chats Collection

1. Go to Firestore Studio

2. Click Start Collection

3. Collection ID: realtime

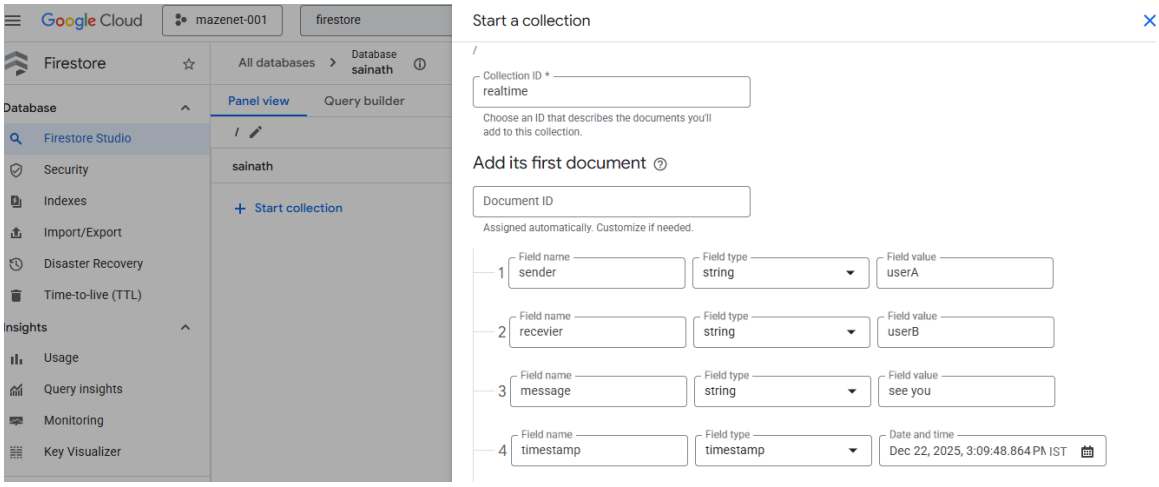
4. Add document with fields:

- sender (string)

- receiver (string)

- message (string)

- timestamp (timestamp)



Step 3: Insert Chat Messages

At least 10 chat messages inserted between users (userA and userB).

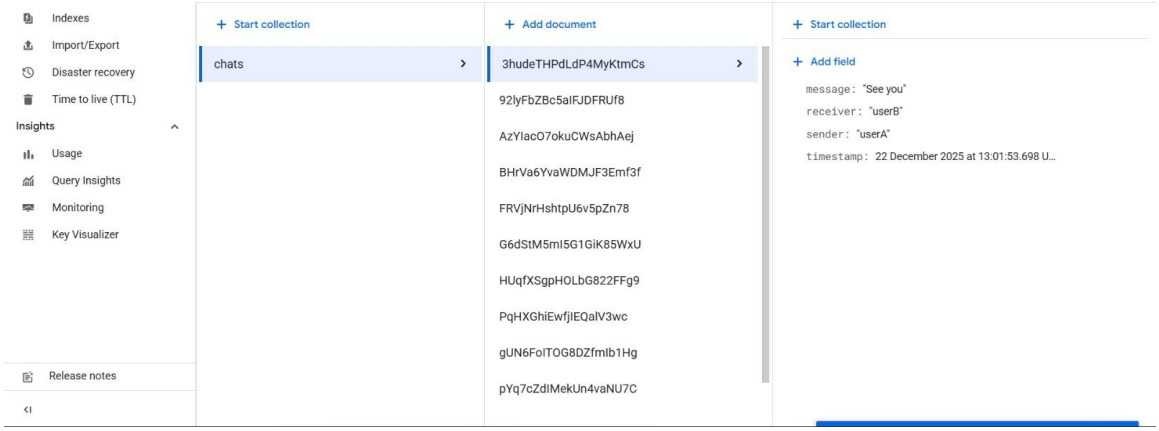
Example fields:

sender: userA / userB

receiver: userB / userA

message: text message

timestamp: current time



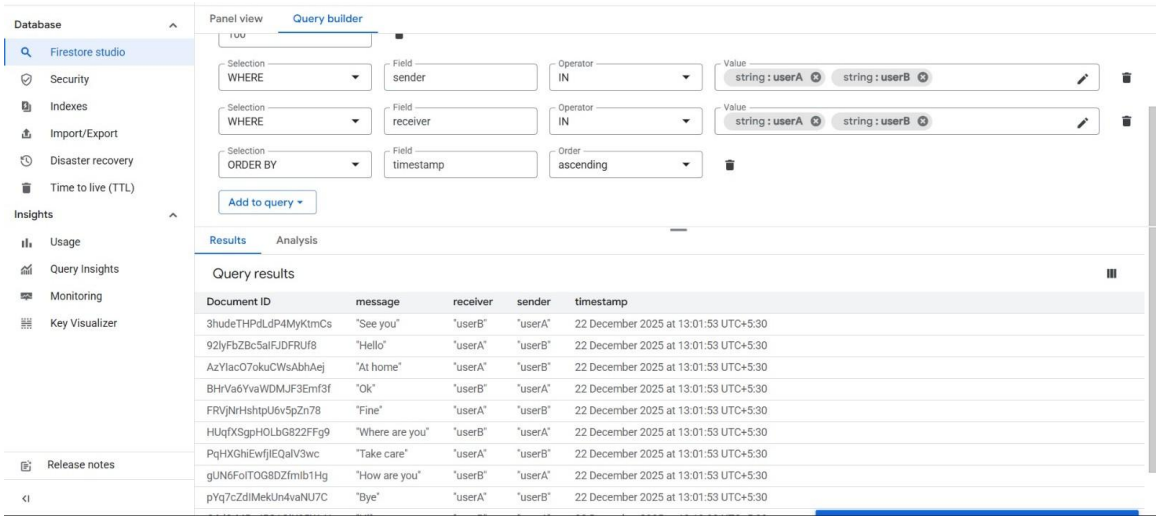
Step 4: Query Messages Between Two Users

Using Firestore Query Builder:

- WHERE sender IN [userA, userB]

- WHERE receiver IN [userA, userB]

- ORDER BY timestamp ASC

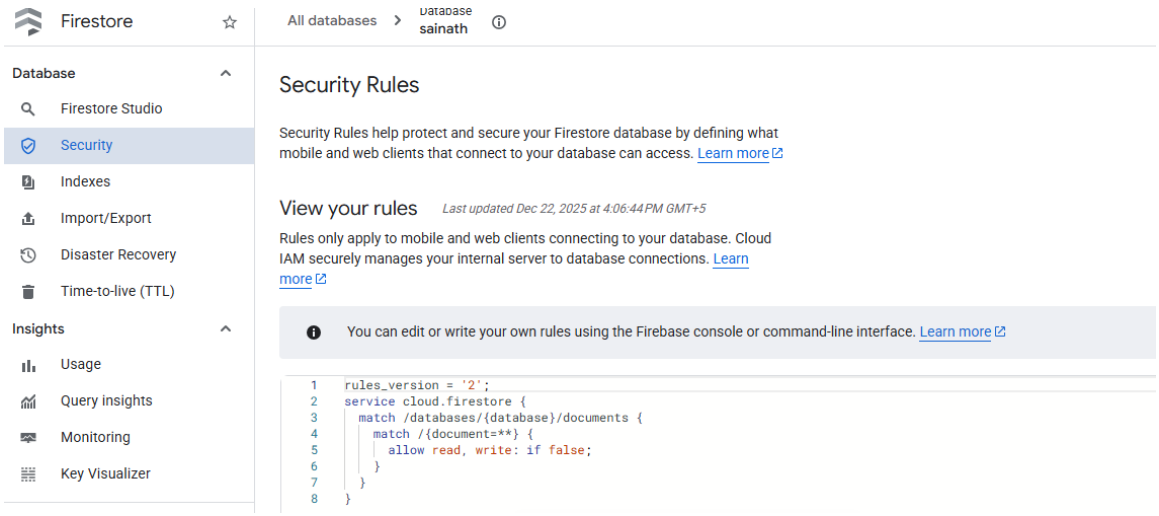


Step 5: Sort Messages by Timestamp

Messages sorted using ORDER BY timestamp (ascending) to display conversation order.

Step 6: Firestore Security Rules

Firestore rules applied:



E-Commerce Platform

Architecture Overview

Cloud SQL (MySQL): Orders, Payments

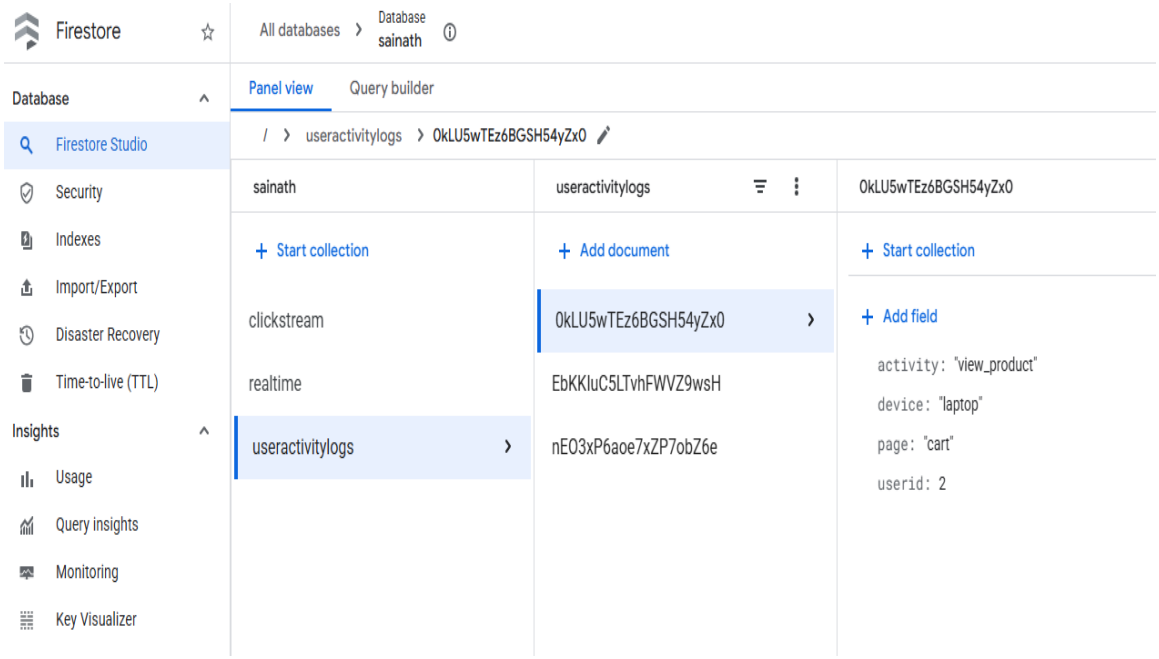
```
mysql> use Ecommerce;
Database changed
mysql> CREATE TABLE Orders (
    ->     order_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     user_id INT NOT NULL,
    ->     order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    ->     status VARCHAR(50) NOT NULL,
    ->     total_amount DECIMAL(10,2) NOT NULL,
    ->     INDEX idx_user_id (user_id)
    -> );
Query OK, 0 rows affected (0.23 sec)

mysql> CREATE TABLE Payments (
    ->     payment_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     order_id INT NOT NULL,
    ->     payment_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    ->     payment_method VARCHAR(50) NOT NULL,
    ->     amount DECIMAL(10,2) NOT NULL,
    ->     status VARCHAR(50) NOT NULL,
    ->     FOREIGN KEY (order_id) REFERENCES Orders(order_id)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

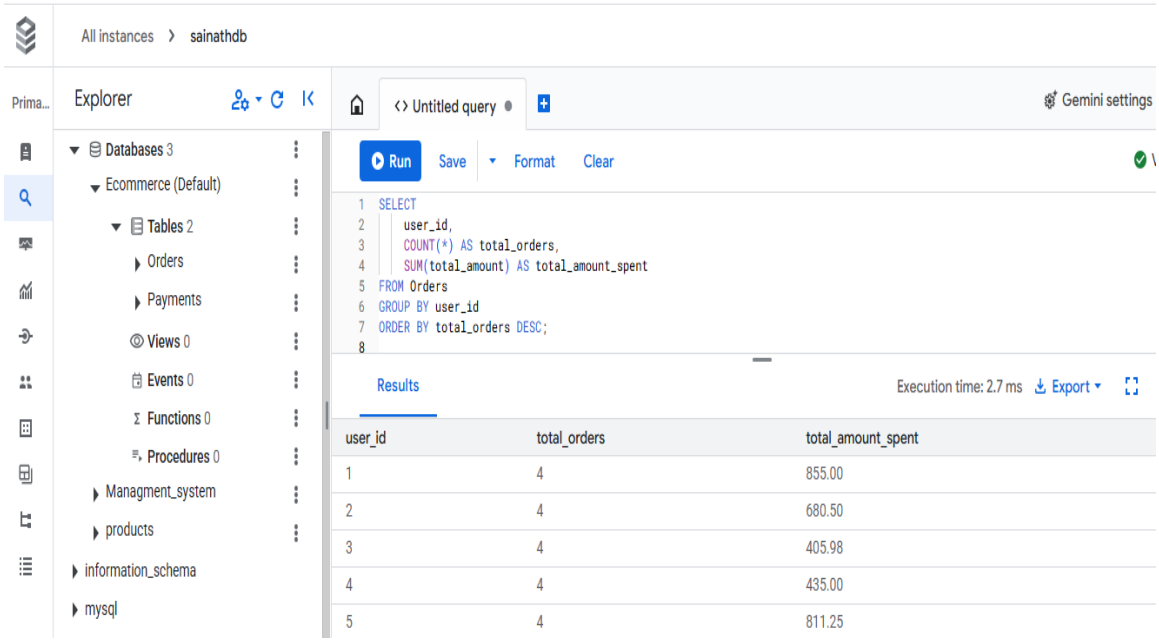
```
mysql> INSERT INTO Orders (user_id, order_date, status, total_amount) VALUES
    -> (1, '2025-12-01 10:00:00', 'delivered', 250.00),
    -> (2, '2025-12-01 11:30:00', 'pending', 120.50),
    -> (3, '2025-12-02 09:15:00', 'shipped', 75.99),
    -> (1, '2025-12-03 14:20:00', 'delivered', 300.00),
    -> (4, '2025-12-04 16:45:00', 'cancelled', 50.00),
    -> (5, '2025-12-05 13:00:00', 'delivered', 180.75),
    -> (2, '2025-12-06 10:30:00', 'shipped', 200.00),
    -> (3, '2025-12-07 12:10:00', 'pending', 99.99),
    -> (4, '2025-12-08 09:50:00', 'delivered', 150.00),
    -> (5, '2025-12-09 17:20:00', 'delivered', 220.50),
    -> (1, '2025-12-10 11:00:00', 'shipped', 125.00),
    -> (2, '2025-12-11 15:30:00', 'delivered', 310.00),
    -> (3, '2025-12-12 14:10:00', 'pending', 90.00),
    -> (4, '2025-12-13 10:00:00', 'delivered', 75.00),
    -> (5, '2025-12-14 16:30:00', 'shipped', 200.00),
    -> (1, '2025-12-15 13:00:00', 'delivered', 180.00),
    -> (2, '2025-12-16 09:45:00', 'cancelled', 50.00),
    -> (3, '2025-12-17 11:20:00', 'delivered', 140.00),
    -> (4, '2025-12-18 12:50:00', 'shipped', 160.00),
    -> (5, '2025-12-19 14:15:00', 'delivered', 210.00);
Query OK, 20 rows affected (0.30 sec)
Records: 20  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Payments (order_id, payment_date, payment_method, amount, status) VALUES
    -> (1, '2025-12-01 10:05:00', 'card', 250.00, 'success'),
    -> (2, '2025-12-01 11:35:00', 'upi', 120.50, 'pending'),
    -> (3, '2025-12-02 09:20:00', 'wallet', 75.99, 'success'),
    -> (4, '2025-12-03 14:25:00', 'card', 300.00, 'success'),
    -> (5, '2025-12-04 16:50:00', 'upi', 50.00, 'failed'),
    -> (6, '2025-12-05 13:05:00', 'wallet', 180.75, 'success'),
    -> (7, '2025-12-06 10:35:00', 'card', 200.00, 'success'),
    -> (8, '2025-12-07 12:15:00', 'upi', 99.99, 'pending'),
    -> (9, '2025-12-08 09:55:00', 'wallet', 150.00, 'success'),
```

Firestore / Bigtable: User activity logs, Clickstream data



SQL Query – Total Orders per User

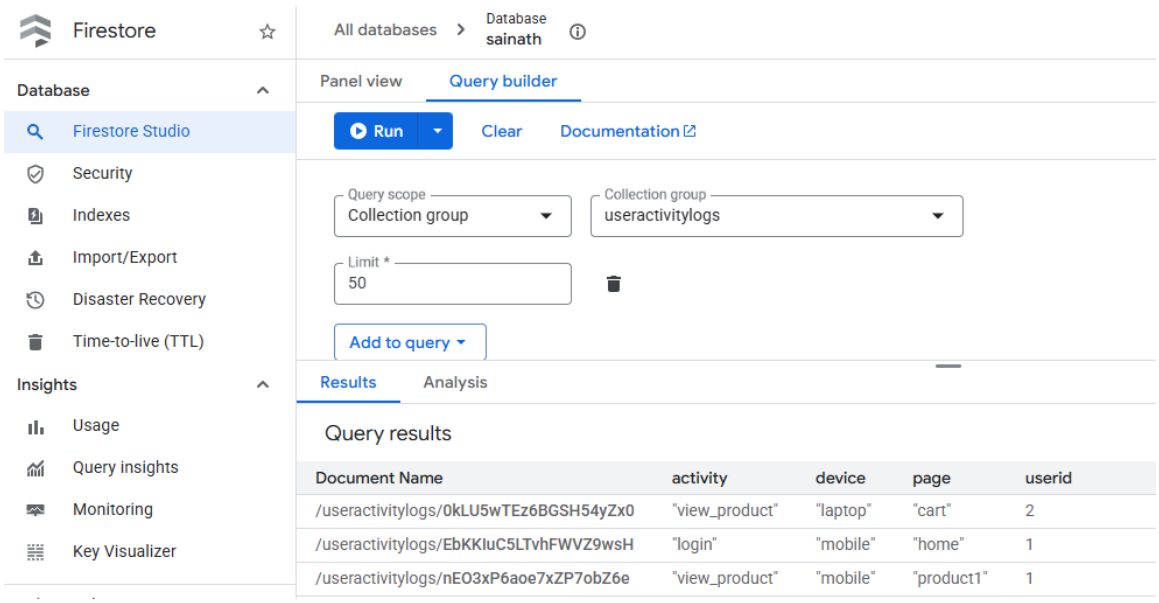


NoSQL – User Activity Logs

Firestore is used to store high-volume user activity and clickstream data.

NoSQL Query – Last 50 User Activities

db.collection("useractivitylogs").orderBy("timestamp","desc").limit(50);



Why SQL for Transactions

Ensures data integrity, consistency, and reliable financial operations.

Why NoSQL for Logs

Provides scalability, flexibility, and fast time-based queries.