

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

The screenshot shows the 'Databases' section of the Google Cloud Platform interface. On the left is a sidebar with icons for Compute Engine, Storage, Functions, BigQuery, Pub/Sub, and Cloud SQL. The 'Cloud SQL' icon is highlighted. The main area shows the 'sainathdb' database under the 'All instances > sainathdb' path. It displays the MySQL 8.0 version and a 'Create database' button. Below is a table listing databases:

Name ↑	Collation	Character set	Type	⋮
information_schema	utf8mb3_general_ci	utf8mb3	System	⋮
Management_system	utf8mb4_0900_ai_ci	utf8mb4	User	⋮
mysql	utf8mb3_general_ci	utf8mb3	System	⋮
performance_schema	utf8mb4_0900_ai_ci	utf8mb4	System	⋮
products	utf8mb4_0900_ai_ci	utf8mb4	User	⋮
sys	utf8mb4_0900_ai_ci	utf8mb4	System	⋮

Step 2: Create Database

Database Name: Management_system

Step 3: Create students Table

Step 4: Insert Records

```

mysql> use Management_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

The screenshot shows the MySQL Workbench interface. On the left is the Explorer pane, which lists databases, tables, and other schema components under the 'Management_system (Default)' database. In the center is the SQL editor pane, containing the following SQL code:

```
1 GRANT SELECT
2 ON Management_system.* 
3 TO 'sainathdemo'@'%';
```

Below the SQL editor is the 'Results' pane, which displays the message: "Statement executed successfully".

IP Restriction:

Authorized Network: 106.206.97.228/32

The screenshot shows the Google Cloud Platform Cloud SQL Connections page for a primary instance. The sidebar on the left includes options like Overview, Cloud SQL Studio, System insights, Query insights, Connections (which is selected), Users, Databases, Backups, Replicas, and Operations. The main content area is titled 'Connections' and has tabs for Summary, Networking (which is selected), Security, and Connectivity tests. Under the 'Networking' tab, there is a section for 'Instance IP assignment' with two options: 'Private IP' (unchecked) and 'Public IP' (checked). Below this is a section for 'Authorized networks' where the IP address 'sainath (106.206.97.228)' is listed. A delete icon is shown next to the IP entry.

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

Name your database
Permanent choice

Database ID * – sainath

Select your edition [New](#)

Operations	Price
Document reads ²	\$0.035 per 100,000 documents
Document writes	\$0.104 per 100,000 documents
Document deletes	\$0.012 per 100,000 documents
Stored data	\$0.104 GiB/month

Pricing summary
Firestore is billed based on operations, storage, and network usage¹. Location affects rates. [Learn more ↗](#)

1. Network traffic costs are dependent on the location of your database and application request behavior. [Learn more ↗](#)
2. Aggregation queries are charged 1 document read for each batch of up to 1000 index documents matched by the query.

[Compare editions](#)

Configuration options
These presets determine how you structure and interact with your data.

[Hide details](#)

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)

The screenshot shows the Google Cloud Firestore interface. On the left, there's a sidebar with options like Firestore, Database, Insights, and Key Visualizer. The main area shows a database named 'sainath'. A modal window titled 'Start a collection' is open, prompting for a 'Collection ID' (set to 'realtime'). Below it, 'Add its first document' is shown with fields for 'Document ID' (auto-generated) and four data entries:

Index	Field name	Field type	Field value
1	sender	string	userA
2	receiver	string	userB
3	message	string	see you
4	timestamp	timestamp	Dec 22, 2025, 3:09:48.864 PM IST

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

The screenshot shows the 'chats' collection in the Firestore interface. It lists several document IDs, each representing a message. The first document is expanded to show its fields:

Field	Value
message	"See you"
receiver	"userB"
sender	"userA"
timestamp	22 December 2025 at 13:01:53.698 U...

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

The screenshot shows the Firestore Query builder interface. The query is set up as follows:

- Selection (WHERE):** sender IN [string : userA, string : userB]
- Selection (WHERE):** receiver IN [string : userA, string : userB]
- Selection (ORDER BY):** timestamp (Order: ascending)

Results:

Document ID	message	receiver	sender	timestamp
3hudeTHPlldP4MyKtmCs	"See you"	"userB"	"userA"	22 December 2025 at 13:01:53 UTC+5:30
92lyFbzBc5alfJDFRUf8	"Hello"	"userA"	"userB"	22 December 2025 at 13:01:53 UTC+5:30
AzYiac07okuCWsaabuAj	"At home"	"userA"	"userB"	22 December 2025 at 13:01:53 UTC+5:30
BHrVa6YvaWDMJF3Emf3f	"Ok"	"userB"	"userA"	22 December 2025 at 13:01:53 UTC+5:30
FtVNrHshlpU6v5pZn78	"Fine"	"userA"	"userB"	22 December 2025 at 13:01:53 UTC+5:30
HUqfxSgpHOLbG822Fg9	"Where are you"	"userB"	"userA"	22 December 2025 at 13:01:53 UTC+5:30
PjXGhiEwfjIEQalV3w0	"Take care"	"userA"	"userB"	22 December 2025 at 13:01:53 UTC+5:30
gJUN6f0TQGDZfmlb1Hg	"How are you"	"userB"	"userA"	22 December 2025 at 13:01:53 UTC+5:30
pyq7cZdlMekUn4vaNU7C	"Bye"	"userA"	"userB"	22 December 2025 at 13:01:53 UTC+5:30

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:

The screenshot shows the Firestore Security Rules page. The rules are defined as follows:

```

rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
  }
}

```

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

The screenshot shows the Firestore Studio interface. On the left, there's a sidebar with options like Security, Indexes, Import/Export, Disaster Recovery, Time-to-live (TTL), Insights (Usage, Query insights, Monitoring, Key Visualizer), and a Database dropdown set to 'Database sainath'. The main area has a 'Panel view' tab selected. The path is / > useractivitylogs > OkLU5wTEz6BGSH54yZx0. The document 'OkLU5wTEz6BGSH54yZx0' is expanded, showing sub-collections 'clickstream' and 'realtime', and a field 'useractivitylogs' with value 'nE03xP6aoe7xZP7obZ6e'. The 'clickstream' collection is selected. Its fields are listed on the right: activity: "view_product", device: "laptop", page: "cart", and userid: 2.

SQL Query – Total Orders per User

The screenshot shows the Bigtable interface. The left sidebar lists databases (Ecommerce (Default)), tables (Orders, Payments), views, events, functions, procedures, and MySQL. The main area shows an 'Untitled query' editor with the following SQL code:

```

1 SELECT
2     user_id,
3     COUNT(*) AS total_orders,
4     SUM(total_amount) AS total_amount_spent
5 FROM Orders
6 GROUP BY user_id
7 ORDER BY total_orders DESC;
8

```

The results table shows the following data:

user_id	total_orders	total_amount_spent
1	4	855.00
2	4	680.50
3	4	405.98
4	4	435.00
5	4	811.25

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);
```

The screenshot shows the Firestore Query builder interface. On the left, there's a sidebar with sections like Database, Security, Indexes, Import/Export, Disaster Recovery, Time-to-live (TTL), Insights, Usage, Query insights, Monitoring, and Key Visualizer. The main area has tabs for Panel view and Query builder, with Query builder being active. It includes fields for Query scope (Collection group set to useractivitylogs), Limit (set to 50), and an Add to query button. Below this, there are Results and Analysis tabs, with Results being active. The results table has columns: Document Name, activity, device, page, and userid. Three documents are listed:

Document Name	activity	device	page	userid
/useractivitylogs/0kLU5wTEz6BGSH54yZx0	"view_product"	"laptop"	"cart"	2
/useractivitylogs/EbKKIuC5LTvhFWVZ9wsH	"login"	"mobile"	"home"	1
/useractivitylogs/nEO3xP6aoe7xZP7obZ6e	"view_product"	"mobile"	"product1"	1

Why SQL for Transactions

Ensures data integrity, consistency, and reliable financial operations.

Why NoSQL for Logs

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