

## **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 used Network access is restricted by allowing connections only from a specific IP address .

Cloud Shell Editor

(samproject-481503) x +

```
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

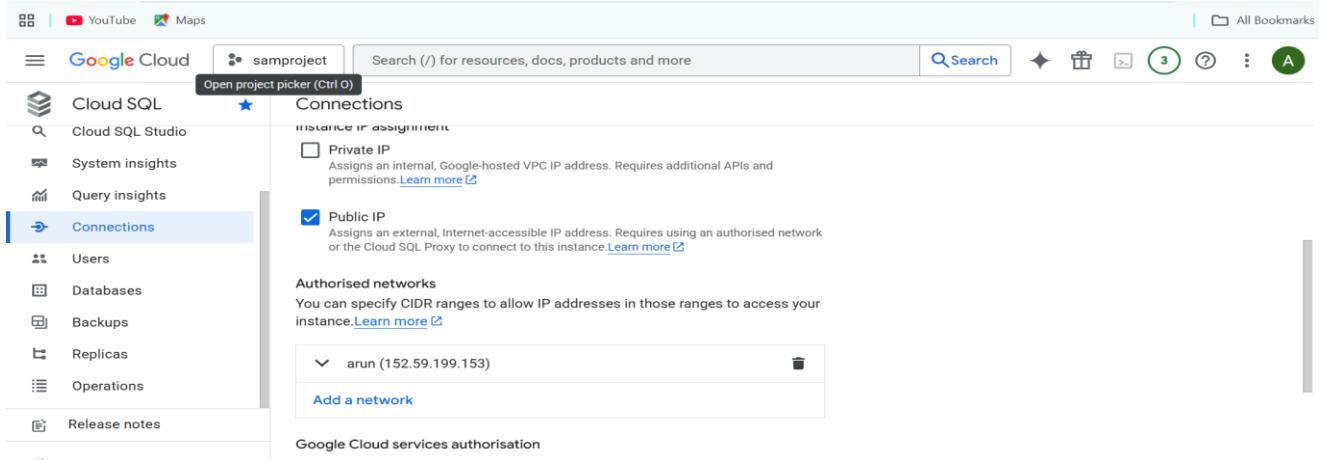
(samproject-481503) x +

```
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) x +

```
| 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.

The screenshot shows the Google Cloud Platform interface for Cloud SQL. The top navigation bar includes 'Google Cloud', a project dropdown ('samproject'), a search bar ('Search (/) for resources, docs, products and more'), and various navigation icons. The main sidebar on the left lists Cloud SQL Studio, System insights, Query insights, Connections, and several tabs: 'Users' (which is selected), 'Databases', 'Backups', 'Replicas', 'Operations', and 'Release notes'. The main content area is titled 'Users' and shows details for the 'arun-mysql' instance. It displays the MySQL 8.0 version and a note about user accounts enabling connections. Below this, there's a section for 'Added users' with a link to 'Authenticated IAM group members'. A table lists three accounts: 'arun-demo' and 'root' (both with host '% (any host)', authentication type 'Built-in', and N/A password status), and a third account (with host '% (any host)', authentication type 'Built-in', and N/A password status). There are also 'Add user account' and 'Edit' buttons.

Google Cloud samproject Search (/) for resources, docs, products and more

Firestore Database arun-nosql

Panel view Query builder

/ > chats > document1

arun-nosql	chats	document1
+ Start collection	+ Add document	+ Start collection
chats >	document1 >	+ Add field
	document10	message: "Hi"
	document2	receiver: "vishnu"
	document3	sender: "arun"
	document4	timestamp: 22 December 2025...

Usage Query Insights Monitoring Key Visualizer Release notes

Google Cloud samproject Search (/) for resources, docs, products and more

Firestore Database arun-nosql

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

The screenshot shows the Firestore security rules configuration page. The left sidebar has 'Security' selected under 'Database'. The main area displays the security rules code:

```
1 rules_version = '2';
2 service cloud.firestore {
3     match /databases/{database}/documents {
4         match /{document=**} {
5             allow read, write: if false;
6         }
7     }
8 }
```

### Hands-on Task 3

You are designing an e-commerce platform.

Requirements:

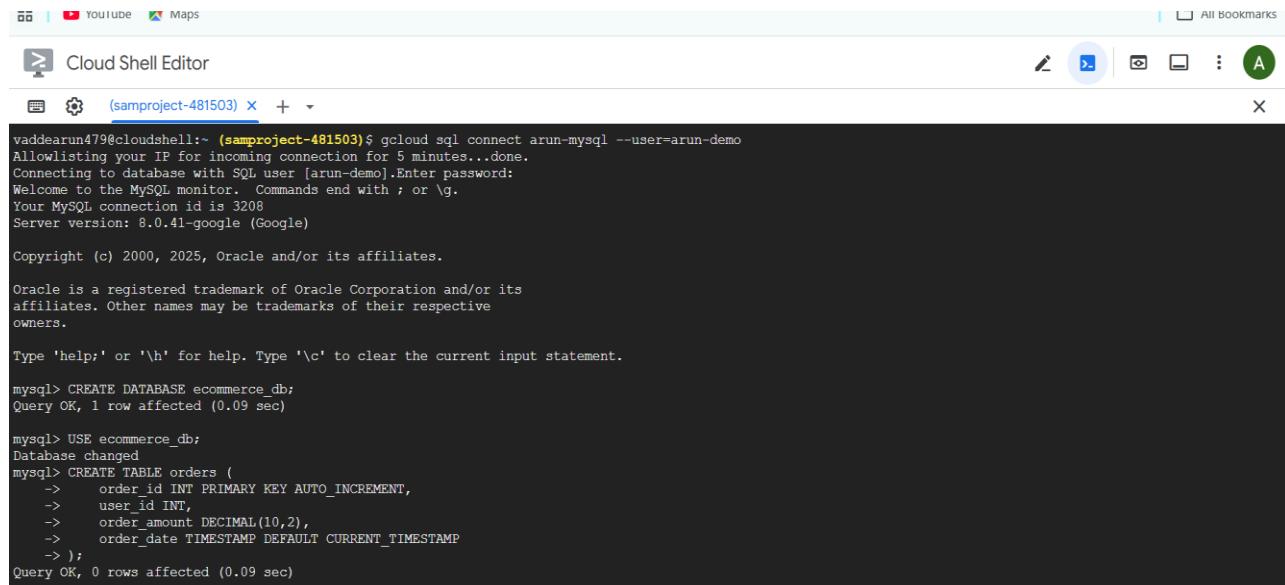
1. Use Cloud SQL to store:
  - Orders
  - Payments
2. Use Firestore or Bigtable to store:
  - User activity logs
  - Clickstream data
3. Implement:
  - One SQL query to fetch total orders per user
  - One NoSQL query to fetch last 50 user activities
4. Explain (practically):
  - Why SQL is chosen for 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.



The screenshot shows a web browser window with a tab labeled "Cloud Shell Editor". The content area displays a terminal session on a "samproject-481503" instance. The terminal output shows the user connecting to a MySQL database using gcloud sql connect, entering their password, and then creating a new database named "ecommerce\_db" and a table named "orders" with specific columns and constraints. The terminal session ends with a "Query OK" message.

```
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 3208
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> 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> 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 |
+-----+-----+

```

File Edit Selection View Go Run

EXPLORER FIRESTORE-TRIGGER ...

trig.py main.py ecommerce.py requirements.txt

ecommerce.py > ...

```

9     # Wide variety of values
10    users = [f"U{i}000+i" for i in range(50)] # 50 users
11
12    activities = [
13        "product_click",
14        "view_page",
15        "add_to_cart",
16        "remove_from_cart",
17        "wishlist_add",
18        "search",
19        "checkout_start"
20    ]
21
22    products = [
23        "iphone15", "samsung_s24", "macbook_air", "dell_xps",

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

Microsoft Windows [Version 10.0.26100.7462]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Pooja\AppData\Roaming\gcloud\firestore-trigger>C:/Users/Pooja/AppData/Roaming/gcloud/firestore-trigger/venv/Scripts/activate.bat

(venv) C:\Users\Pooja\AppData\Roaming\gcloud\firestore-trigger>python ecommerce.py
SuccessFully inserted 200 diverse user activity records

(venv) C:\Users\Pooja\AppData\Roaming\gcloud\firestore-trigger>

> OUTLINE > TIMELINE

The screenshot shows the Google Cloud Firestore interface. The left sidebar has sections for Database (Security, Indexes, Import/Export, Disaster recovery, Time to live (TTL)) and Insights (Usage, Query Insights, Release notes). The main area is titled 'Query builder' with tabs for 'Run' (selected), 'Clear', and 'Documentation'. Below this is a 'Query scope' section with a dropdown for 'Collection \*' set to 'product\_click'. The results tab is selected, showing a table of query results:

Document ID	activity_type	category	device	page	product	timestamp
3Uftyctsi961UuvMyicE	"product_click"	"electronics"	"electronics"	"/product/tv"	"Mi"	22...
ELq6TsdsF705HF69T5H5	"product_click"	"products"	"shirts"	"/product/shirts"	"l"	22...
IH4MGAA1PGTI34aOF7Xn	"product_click"	"electronics"	"electronics"	"/product/tv"	"lg"	22...
J2REJHcQUwdil64qw3cB	"product_click"	"products"	"shirts"	"/product/shirts"	"xl"	22...
JVEx8KJkRPzR8ETsOOPS	"product_click"	"electronics"	"electronics"	"/product/tv"	"sony"	22...
MAXnGPFFNNE6UPMxtoSy	"product_click"	"electronics"	"mobile"	"/product/iphone12"	"iphone12"	22...
RjtjpcJJbMtQsojeHgZF	"product_click"	"products"	"shirts"	"/product/shirts"	"xxl"	22...

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.

