

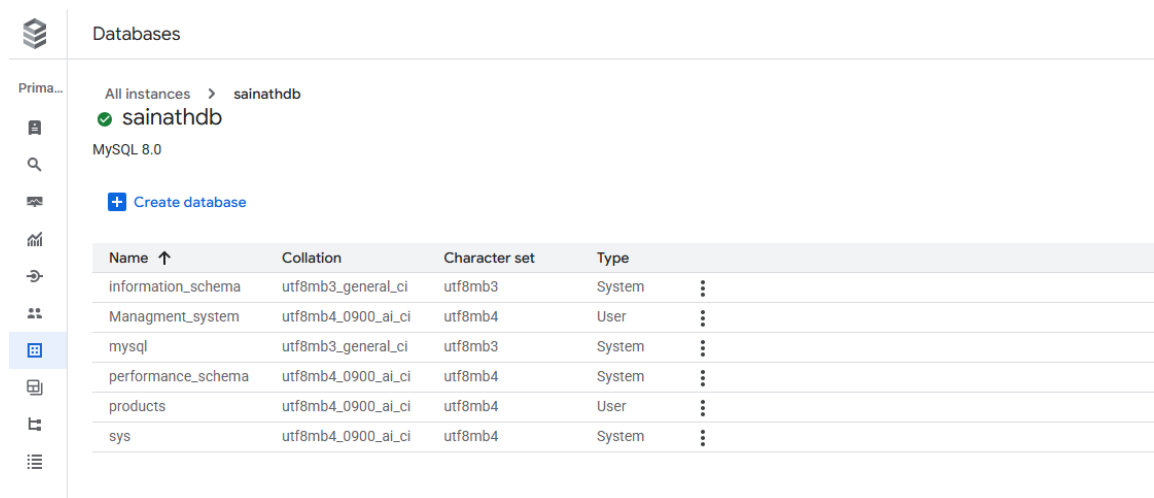
Student Management System using Google Cloud SQL (MySQL)

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 Google Cloud SQL console for instance 'sainathdb'. The 'Databases' tab is selected, displaying a table of existing databases. The table has columns for Name, Collation, Character set, and Type. The databases listed are information_schema, Managment_system, mysql, performance_schema, products, and sys.

Name	Collation	Character set	Type
information_schema	utf8mb3_general_ci	utf8mb3	System
Managment_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 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;

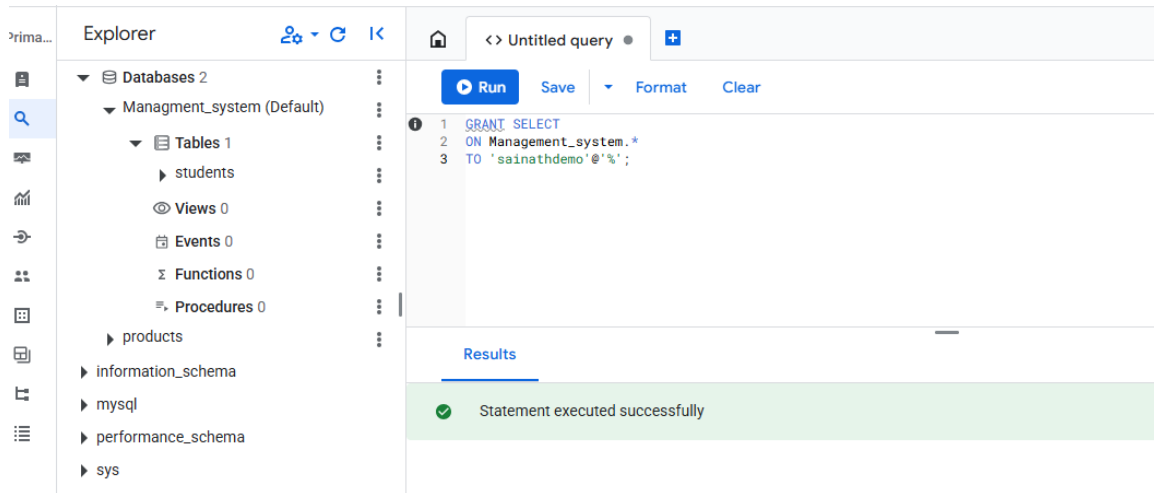
```
(venv) PS C:\Users\reddy\Music\HSBC\gcp> python app.py
Connected Successfully at: 2025-12-22 08:12:37

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> █
```

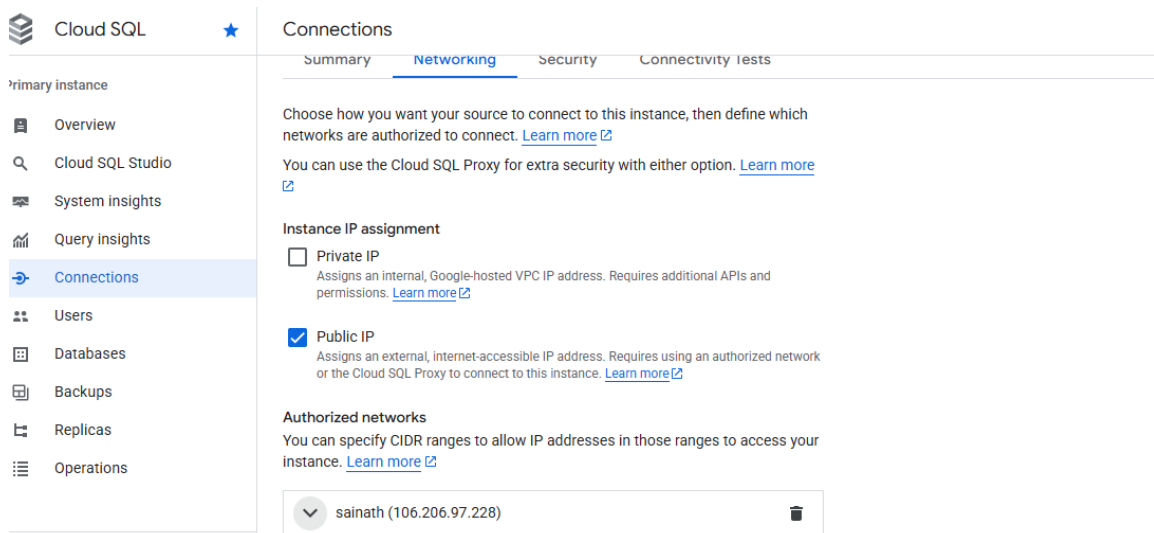
Read-only User:

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

Name your database
Permanent choice

Database ID *
sainath

Select your edition **New**

☒ **Standard Edition**
Firestore's simple query engine with automatic indexing.

☐ **Enterprise Edition**
Firestore's advanced query engine with high customizability and MongoDB compatibility.

Not sure which edition is right for you? [Compare editions](#)

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

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

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

1. Network traffic costs are dependent on the location of your database and application request behavior. [Learn more](#)

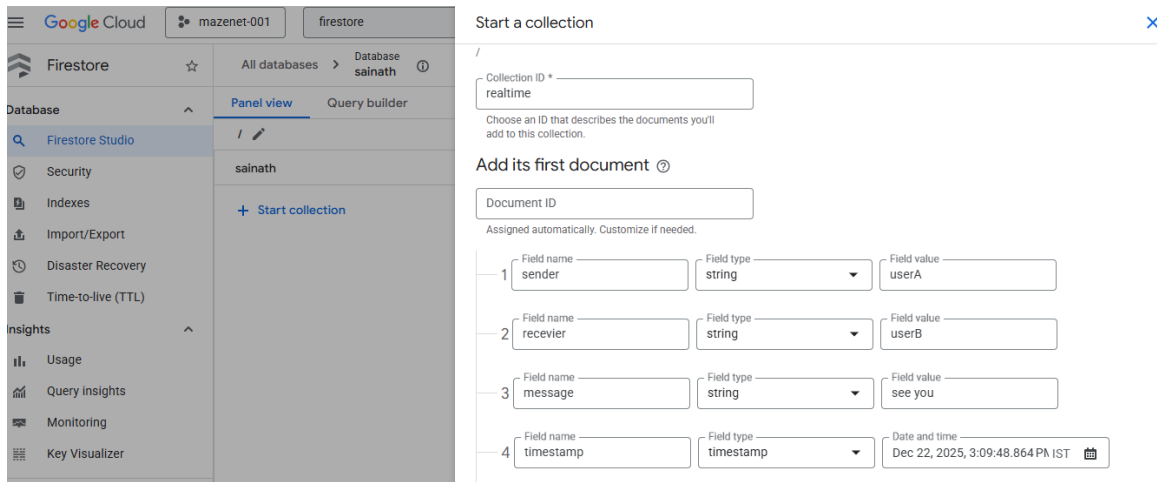
2. Aggregation queries are charged 1 document read for each batch of up to 1000 index documents matched by the query.

[^ 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)



Start a collection

Collection ID

Choose an ID that describes the documents you'll add to this collection.

Add its first document

Document ID

Assigned automatically. Customize if needed.

1	Field name <input type="text" value="sender"/>	Field type <input type="text" value="string"/>	Field value <input type="text" value="userA"/>
2	Field name <input type="text" value="receiver"/>	Field type <input type="text" value="string"/>	Field value <input type="text" value="userB"/>
3	Field name <input type="text" value="message"/>	Field type <input type="text" value="string"/>	Field value <input type="text" value="see you"/>
4	Field name <input type="text" value="timestamp"/>	Field type <input type="text" value="timestamp"/>	Date and time <input type="text" value="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).

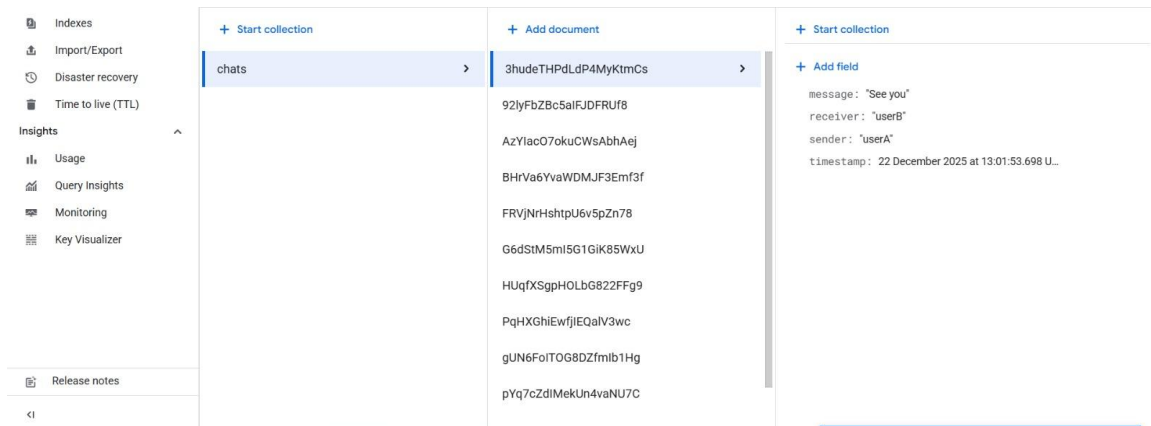
Example fields:

sender: userA / userB

receiver: userB / userA

message: text message

timestamp: current time



Indexes

Import/Export

Disaster recovery

Time to live (TTL)

Insights

Usage

Query Insights

Monitoring

Key Visualizer

Release notes

chats

+ Start collection

+ Add document

3hudeTHPdLdP4MyKtmCs

92lyFbZBc5aIFJDFRUf8

AzYIacO7okuCWsAbhAej

BHrVa6YvaWDMJF3Emf3f

FRVjNrHshpU6v5pZn78

G6dStM5ml5G1GIK85WxU

HUqfXsGpHOLbG822FFg9

PqHXGhiEwfjIEQaIV3wc

gUN6FoITOG8DZfmb1Hg

pYq7cZdlMekUn4vaNU7C

+ Start collection

+ Add field

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 Studio Query Builder interface. The left sidebar contains navigation options: Database, Security, Indexes, Import/Export, Disaster recovery, Time to live (TTL), Insights, Usage, Query Insights, Monitoring, and Key Visualizer. The main panel is titled 'Query builder' and shows a query with three clauses: 'WHERE sender IN [string: userA, string: userB]', 'WHERE receiver IN [string: userA, string: userB]', and 'ORDER BY timestamp ascending'. The 'Query results' section displays a table with columns: Document ID, message, receiver, sender, and timestamp. The table contains 10 rows of data, showing messages sent by 'userA' to 'userB' at various timestamps on December 22, 2025.

Document ID	message	receiver	sender	timestamp
3rhudeTHPdLdP4MyKtmCs	"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
AzYIac07okucWsAbhAej	"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
FRVjNrhshUpU6v5pZn78	"Fine"	"userA"	"userB"	22 December 2025 at 13:01:53 UTC+5:30
HLqfXSgphOLbG822FFg9	"Where are you"	"userB"	"userA"	22 December 2025 at 13:01:53 UTC+5:30
PqHxGhiEwIjEQaIV3wc	"Take care"	"userA"	"userB"	22 December 2025 at 13:01:53 UTC+5:30
gUN6FoITOG8DZfmb1Hg	"How are you"	"userB"	"userA"	22 December 2025 at 13:01:53 UTC+5:30
pYq7cZdlMekJn4vaNU7C	"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 in the Firebase console. The left sidebar contains navigation options: Database, Security, Indexes, Import/Export, Disaster Recovery, Time-to-live (TTL), Insights, Usage, Query insights, Monitoring, and Key Visualizer. The main panel is titled 'Security Rules' and contains the following text: 'Security Rules help protect and secure your Firestore database by defining what mobile and web clients that connect to your database can access. [Learn more](#)'. Below this, there is a section 'View your rules' with the text 'Last updated Dec 22, 2025 at 4:06:44 PM GMT+5'. The rules section contains a single rule with the following 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 }

```

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

Firestore

☆

All databases > Database sainath ⓘ

Database

Panel viewQuery builder

Firestore Studio

/ > useractivitylogs > OkLU5wTEz6BGSH54yZx0 ✎

Security

sainath

useractivitylogs

OkLU5wTEz6BGSH54yZx0

Indexes

+ Start collection

+ Add document

+ Start collection

Import/Export

clickstream

OkLU5wTEz6BGSH54yZx0 >

+ Add field

Disaster Recovery

realtime

EbKKluC5LTvhFWVZ9wsH

activity: "view_product"

Time-to-live (TTL)

useractivitylogs >

nE03xP6aoe7xZP7obZ6e

device: "laptop"

Insights

Usage

Query insights

Monitoring

Key Visualizer

page: "cart"

userid: 2

SQL Query – Total Orders per User

All instances > sainathdb

Prima...

Explorer

<> Untitled query

Gemini settings

Databases 3

Ecommerce (Default)

Tables 2

Orders

Payments

Views 0

Events 0

Functions 0

Procedures 0

Managment_system

products

information_schema

mysql

Run

Save

Format

Clear

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

Results

Execution time: 2.7 ms

Export

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 Google Cloud Firestore Studio interface. On the left is a sidebar with navigation options: Database, Security, Indexes, Import/Export, Disaster Recovery, Time-to-live (TTL), Insights, Usage, Query insights, Monitoring, and Key Visualizer. The main panel is titled 'Query builder' and shows a query for the 'useractivitylogs' collection group, limited to 50 results. The 'Run' button is highlighted. Below the query builder, the 'Results' tab is active, displaying a table of query results.

Document Name	activity	device	page	userid
/useractivitylogs/0kLU5wTEz6BGSH54yZx0	"view_product"	"laptop"	"cart"	2
/useractivitylogs/EbKKIuC5LTvhFWVZ9wsH	"login"	"mobile"	"home"	1
/useractivitylogs/nEQ3xP6aoe7xZP7obZ6e	"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.