HW-5 WRITE UP

Mohit Sai Gutha mohitsai@bu.edu U48519832

Instructions to Run

1. Prerequisites

- Google Cloud SDK installed and authenticated.

2. Steps to Run

a) **Database**
Credentials and instance connector variables :
PROJECT_ID="ds-561-mohitsai"
DB_USER="root"
DB_PASS="CloudComputing"
DB_NAME="hw5"
INSTANCE_CONNECTION_NAME= "ds-561-mohitsai:us-central1:ds561-hw5"
To check the actual records and fields of the db, the teaching staff can access it on the cloud (all members of the staff have owner privileges to the account).

b) If the virtual machines from the previous homework are still existing, then just copy hw5-app1.py into the virtual machine. The second app virtual machine remains the same. The client virtual machine needs a small change in its shell script to run two concurrent clients sending out 50,000 requests each. If the virtual machines do not exist please refer to homework 4 write up to set them up.

c) **Server**
After copying the hw5-app1.py file to the server vm. Install and setup the database client for mysql using the steps provided in Lecture 12 Slides - Slide 9.
Install all the necessary dependencies given in slide 10.
Activate python venv and then run **python3 hw5-app1.py**.
If the app starts serving on port 8081 and prints on command that the database connection was successful then the server is fully setup and ready.

d) **Client**
The virtual machine setup for the client remains the same. we only change the shell script to add the random seed, change the number of requests to 50,000 and then we run two concurrent clients sending these requests.

e) **App2**
App2 virtual machine and code is unchanged.

Design and Implementation (HW-5 Files)

Additional resources used are:
Google Cloud SQL (MySQL): Stores request data and error logs; sqlalchemy; pymysql;

Key Components and Workflow

Configuration and Setup:

Project-specific information (e.g., project ID, database credentials, Pub/Sub topic) is specified as constants.

A Cloud SQL connection pool (pool) is created with a connection function (getconn), which connects to a MySQL instance in Google Cloud SQL.

Database Table Creation:

The function create_tables_if_not_exists() ensures that two tables—requests and failed_requests—exist before the server handles any requests.

requests Table: Stores details of each request, including:

country, client_ip, gender, age_group, and income_group to provide context on the requestor.

is_banned as a Boolean flag to mark requests from restricted regions.

request_time and requested_file to record the time and file requested.

failed_requests Table: Stores unsuccessful requests, including server errors or missing files.

Each entry includes:

request_time, requested_file, and error_code to log the cause of failure.

HTTP GET Request Handling:

Request Parsing:

The handler extracts the bucket name, file directory, and file name from the URL path and additional metadata from headers (X-country, X-client-IP, etc.).

Country Check:

Before proceeding with the request, it checks if the country header matches any in the banned_countries list. If a match is found:The server logs the attempt as a failed request in failed_requests with error code 400 (bad request).

Logs details of the request (with is_banned=True) in the requests table.

Sends a message to Pub/Sub for monitoring purposes.

File Retrieval:

If the country is allowed, the handler attempts to retrieve the specified file from Google Cloud Storage:

File Not Found:

If the file doesn't exist, it logs an error in failed_requests (error code 404) and sends a 404 Not Found response.

Successful File Retrieval:

If the file is found, it sends the file content in a 200 OK response and logs the request details (with is_banned=False) in the requests table.

Database Design:

Using Cloud SQL: A MySQL database is chosen for structured data storage with straightforward querying. The decision to use MySQL via Google Cloud SQL facilitates scalability, durability, and ease of integration with other GCP services.
Connection Pooling with SQLAlchemy: SQLAlchemy's connection pooling simplifies connection management and improves efficiency under heavy traffic.
Logging Requests and Failures: Storing both successful and failed requests serves multiple purposes:
Data Analysis: Collecting demographic data (like gender, age_group, income_group) can be analyzed for insights into usage patterns, which can help guide future development. (HW6)
Tables Design: The requests table has an is_banned flag to differentiate between denied and permitted requests, allowing easy querying of restricted-access logs.
failed_requests is a lightweight table designed for quick lookups and minimal storage requirements, focusing only on key error details (request_time, requested_file, and error_code).
AUTO_INCREMENT Primary Keys: Both tables use auto-incrementing primary keys for unique identifiers, simplifying record retrieval and indexing for frequent reads. This helped me to maintain the db in **2nd NORMAL FORM** easily.
Enums and Boolean Fields: Gender is stored as an ENUM to limit options and reduce errors. is_banned is a BOOLEAN for clarity in banned vs. non-banned requests.
Connection Pooling: Using a connection pool helps maintain efficient resource usage by reusing connections rather than repeatedly opening new ones.

Demonstration of Implementation

1. CURL command to send a legitimate request with all the headers.

   curl -X GET \

   -H "Country: Antigua and Barbuda" \

   -H "Client-IP: 169.93.236.90" \

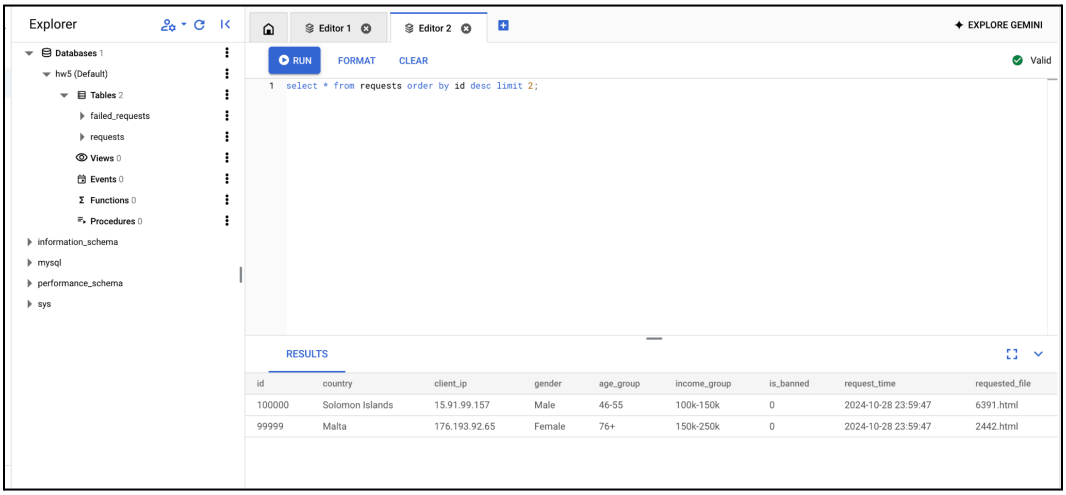
   -H "Gender: Male" \

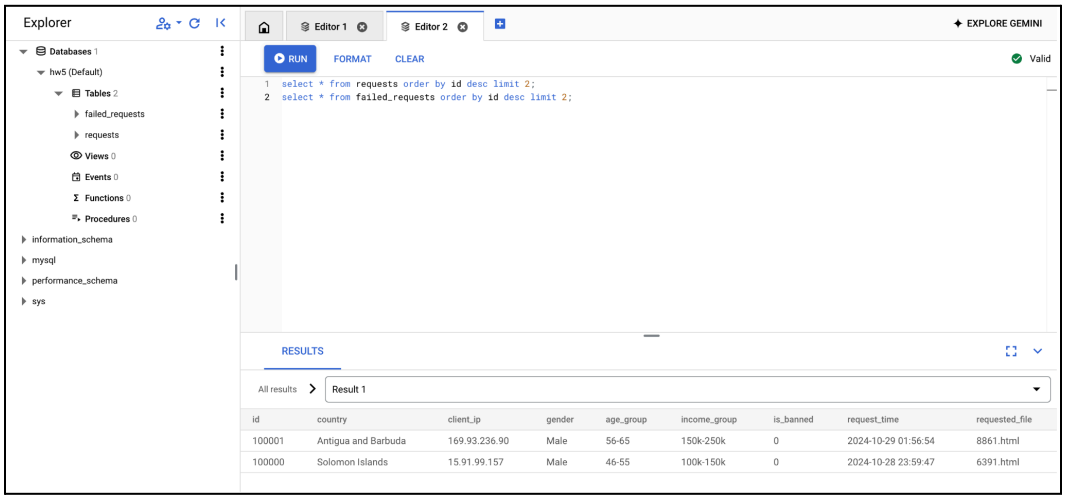
   -H "Age-Group: 56-65" \

   -H "Income-Group: 150k-250k" \

   "http://35.184.82.220:8081/hw2-mohitsai/files/files/8861.html"

   The table before and after running the command :





2. CURL command to send a request from a banned country :

curl -X GET \

-H "Country: Cuba" \

-H "Client-IP: 169.93.236.90" \

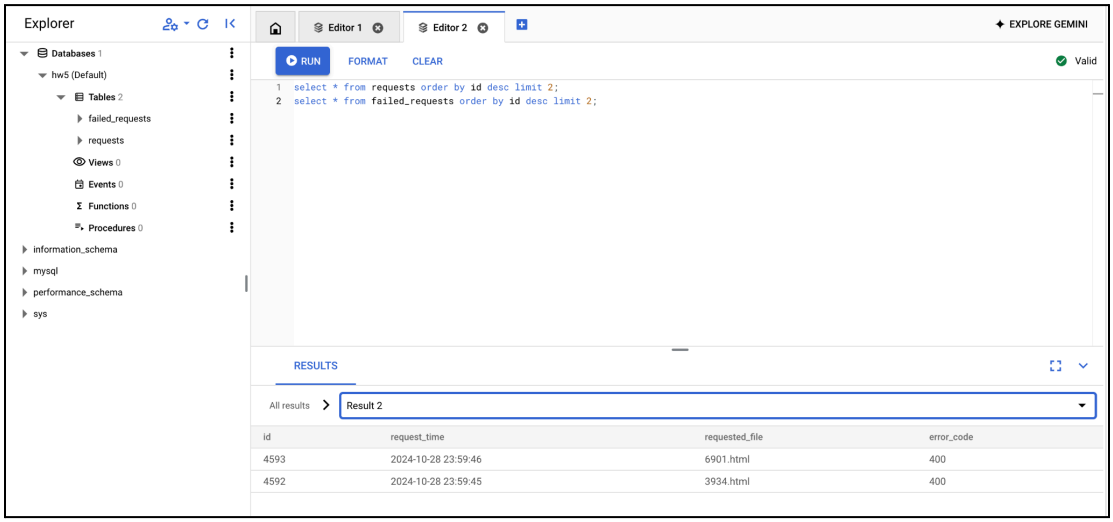
-H "Gender: Male" \

-H "Age-Group: 56-65" \

-H "Income-Group: 150k-250k" \

"http://35.184.82.220:8081/hw2-mohitsai/files/files/1000.html"

The table before and after running the command :



3. CURL command to send a request for a non existent file :

curl -X GET \

-H "Country: Antigua and Barbuda" \

-H "Client-IP: 169.93.236.90" \

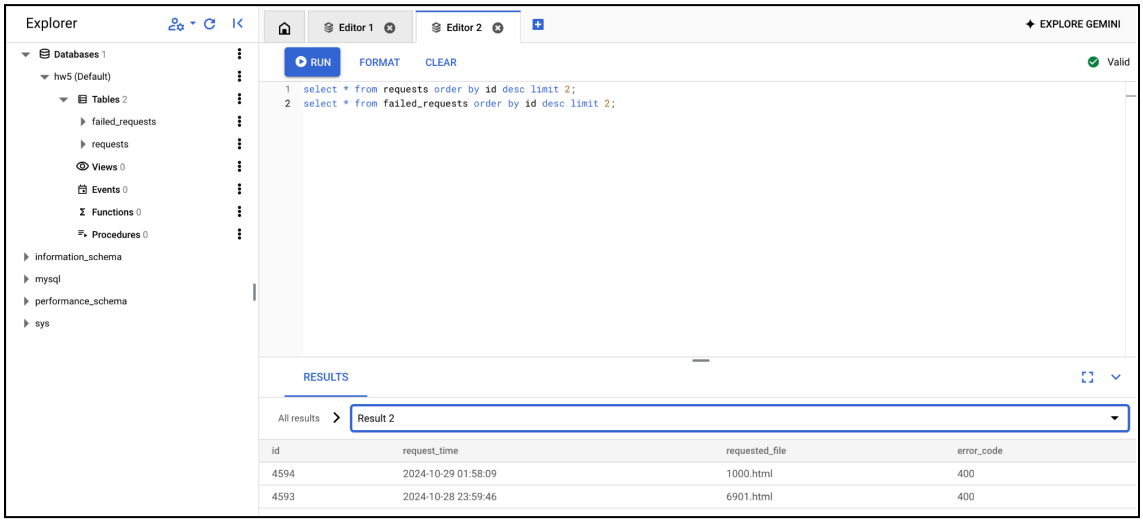
-H "Gender: Male" \

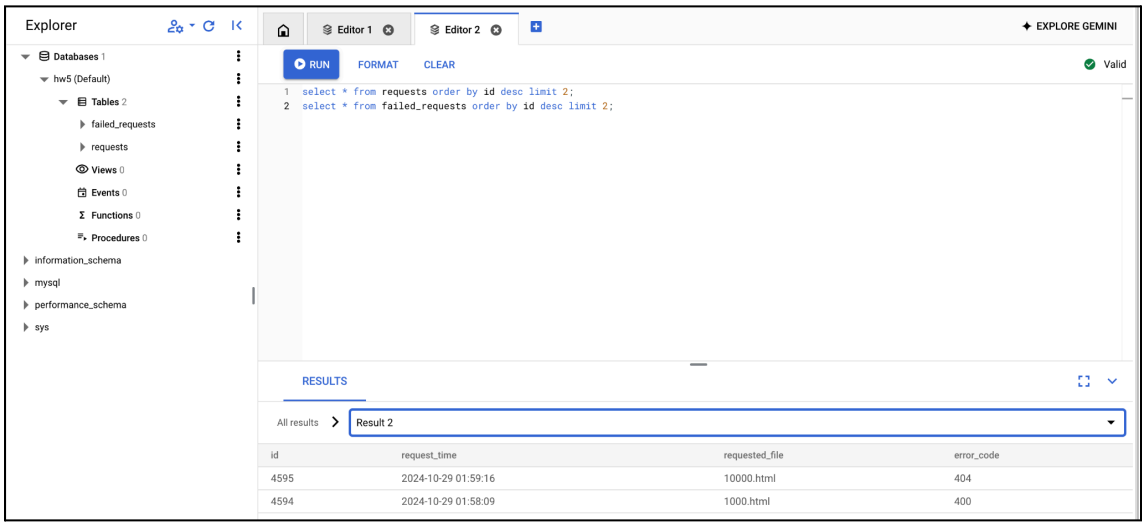
-H "Age-Group: 56-65" \

-H "Income-Group: 150k-250k" \

"http://35.184.82.220:8081/hw2-mohitsai/files/files/10000.html"

The table before and after running the command :





The Statistics computed after all the 10000 requests had been processed and added to the table using SQL :

RUN FORMAT CLEAR

```
2
3
4
5  -- How many requests were you able to process successfully vs unsuccessfully?
6  SELECT
7      (SELECT COUNT(*) FROM requests) AS total_requests,
8      (SELECT COUNT(*) FROM failed_requests) AS total_failed_requests;
9
10
```

RESULTS

total_requests	total_failed_requests
100001	4596

RUN FORMAT CLEAR

```
1
2
3  -- How many requests came from banned countries?
4  SELECT COUNT(*) FROM requests WHERE is_banned = TRUE;
5
```

RESULTS

COUNT(*)
4589

RUN FORMAT CLEAR

```
1
2
3  -- How many requests were made by Male vs Female users?
4  SELECT gender, COUNT(*) FROM requests GROUP BY gender;
5
```

RESULTS

gender	COUNT(*)
Female	50048
Male	49953

RUN FORMAT CLEAR

```
1
2
3  -- What were the top 5 countries sending requests to your server?
4  SELECT country, COUNT(*) FROM requests GROUP BY country ORDER BY COUNT(*) DESC LIMIT 5;
5
6
```

RESULTS

country	COUNT(*)
Cabo Verde	579
Tuvalu	558
Nigeria	557
Gabon	554
Bhutan	554

RUN FORMAT CLEAR

```
1
2
3  -- What age group issued the most requests to your server?
4  SELECT age_group, COUNT(*) FROM requests GROUP BY age_group ORDER BY COUNT(*) DESC LIMIT 1;
5
6
7
```

RESULTS

age_group	COUNT(*)
66-75	12579

RUN FORMAT CLEAR

```
1
2
3
4  -- What income group issued the most requests to your server?
5  SELECT income_group, COUNT(*) FROM requests GROUP BY income_group ORDER BY COUNT(*) DESC LIMIT 1;
6
```

RESULTS

income_group	COUNT(*)
150k-250k	12663

SCHEMA OF DATABASE :

```
1  SELECT * from requests limit 10;
```

RESULTS

id	country	client_ip	gender	age_group	income_group	is_banned	request_time	requested_file
1	Pakistan	154.210.58.90	Female	26-35	20k-40k	0	2024-10-28 20:53:43	1512.html
2	Chile	72.58.174.249	Female	56-65	60k-100k	0	2024-10-28 20:53:43	9085.html
3	Ireland	130.58.136.215	Female	46-55	40k-60k	0	2024-10-28 20:53:43	5223.html
4	Mexico	85.199.155.194	Male	36-45	20k-40k	0	2024-10-28 20:53:43	506.html
5	Cuba	84.12.216.32	Female	66-75	100k-150k	1	2024-10-28 20:53:44	1888.html
6	Syria	55.166.179.24	Male	0-16	0-10k	1	2024-10-28 20:53:44	6398.html
7	Rwanda	72.213.211.205	Male	46-55	40k-60k	0	2024-10-28 20:53:44	2454.html
8	Kyrgyzstan	134.176.104.62	Male	66-75	100k-150k	0	2024-10-28 20:53:44	7977.html
9	Algeria	208.16.179.204	Female	76+	250k+	0	2024-10-28 20:53:44	592.html
10	Oman	116.229.242.187	Male	46-55	60k-100k	0	2024-10-28 20:53:44	8758.html

RUN FORMAT CLEAR Valid

```
1  SELECT * from failed_requests ORDER BY id DESC limit 10;
```

RESULTS

id	request_time	requested_file	error_code
4596	2024-10-29 01:59:24	locale	500
4595	2024-10-29 01:59:16	10000.html	404
4594	2024-10-29 01:58:09	1000.html	400
4593	2024-10-28 23:59:46	6901.html	400
4592	2024-10-28 23:59:45	3934.html	400
4591	2024-10-28 23:59:40	3561.html	400
4590	2024-10-28 23:59:38	7240.html	400
4589	2024-10-28 23:59:36	1407.html	400
4588	2024-10-28 23:59:35	4040.html	400
4587	2024-10-28 23:59:31	9620.html	400

DATABASE - HW5

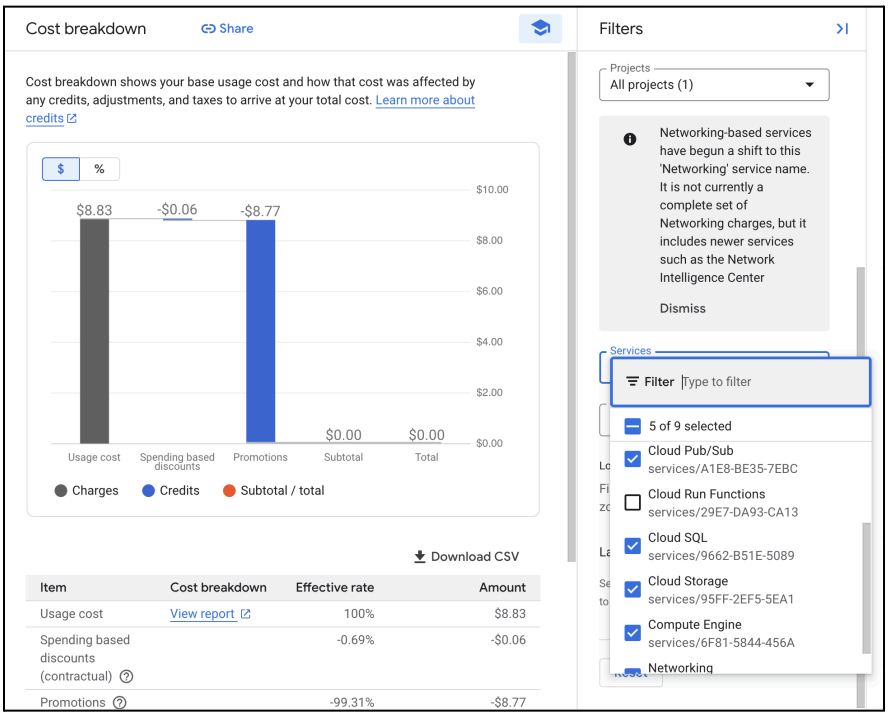
requests	
id (PRIMARY, AUTO_INCREMENT)	INT
country	VARCHAR(255) NOT NULL
client_ip	VARCHAR(45)
gender	ENUM
age_group	VARCHAR(20)
income_group	VARCHAR(20)
is_banned	BOOLEAN
request_time	DATETIME
requested_file	VARCHAR(255)

failed_requests	
id (PRIMARY, AUTO_INCREMENT)	INT
request_time	DATETIME
requested_file	VARCHAR(255)
error_code	INT

Though the two relations of the database exist separately with no relation between them right now, we could possibly relate the requests and failed_requests tables. You could add a foreign key column request_id in the failed_requests table. This request_id would reference the id column in the requests table, establishing a link between a failed request and its original request in requests. This relationship allows you to track failed attempts that are tied to specific requests, making it easier to troubleshoot issues associated with particular requests. Adding this foreign key provides a one-to-many relationship, where a single request might correspond to multiple failed attempts recorded in failed_requests. But for the scope of the task at hand and given that we had already finished loading the database by then, I decided it was not required as of now.

Total Spend on Cloud Resources

Billing report generated for use related to this homework :



Service description	Service ID	Cost ($)	Discounts ($	Promotions a	Unrounded s	Subtotal ($)	Percent change in subtotal compared to previous period
Cloud SQL	9662-B51E-5	5.85	0	-5.85	0.000028	0	0%
Cloud Storage	95FF-2EF5-5	0.15	0	-0.15	0.000001	0	0%
Cloud Run Functions	29E7-DA93-(	0.01	-0.01	0	0	0	0%
Networking	E505-1604-5	0.13	-0.13	0	0	0	0%
Compute Engine	6F81-5844-4	5.57	0	-5.57	-0.000029	0	0%
				Subtotal	0	0	
				Tax	0	0	
				Total	0	0	

For Compute Engine, I incurred a cost of $5.57. All three VMs were the smallest possible VMs and also part of the free tier offering of google cloud, that is E2 Micro. The cost is mainly due to the 10000 requests that were sent and processed and then added to the db. For Networking, I had a cost of $0.13, likely due to the static external ip address assigned to the server vm and all the request traffic.

The following graphs show the Disk Utilization (I/O Cost) and query cost of all the sql operations we performed for the homework :