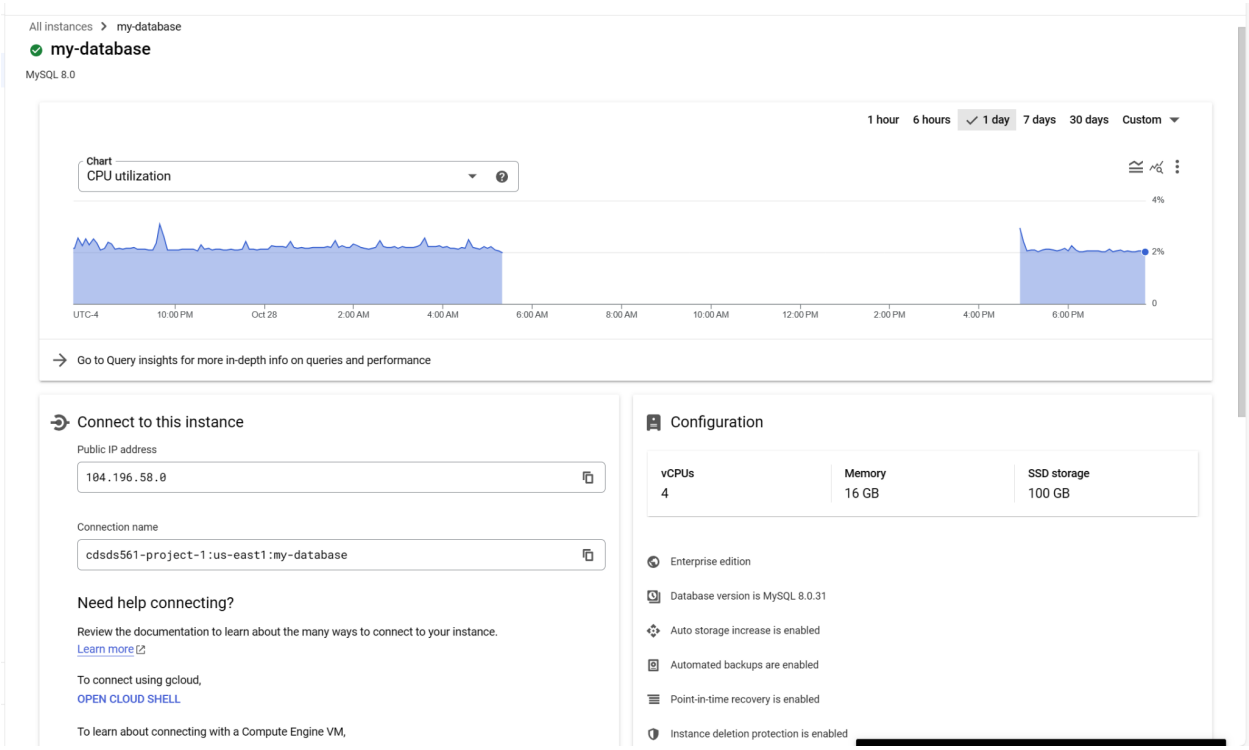
Project id: cdsds561-project-1
Bucket Name: hw2nirbhgsutil
Bucket-2 Name:hw4nirbhgsutil
Directory Name: test-dir
Github Link:CDS-DS-561-hw2/hw4 at master · nirbhay221/CDS-DS-561-hw2 (github.com)
Topic name : projects/cdsds561-project-1/topics/hw3nirbhgsutil
Topic Id: hw3nirbhgsutil
Subscription Id: hw3nirbhgsutil-sub
Service Account email for pub sub:
pubsubserviceacc-hw3-nirbh@cdsds561-project-1.iam.gserviceaccount.com
Service Account for pub sub: pubsubServiceAcc-hw3-nirbh
Service Account for SQL:
cloud-sql-authorize-vm@cdsds561-project-1.iam.gserviceaccount.com
Reserved IP : 34.75.102.252
Sql database instance: my-database
Sql database: First-Trial
Sql Tables : Clients, main_table, error_logs.

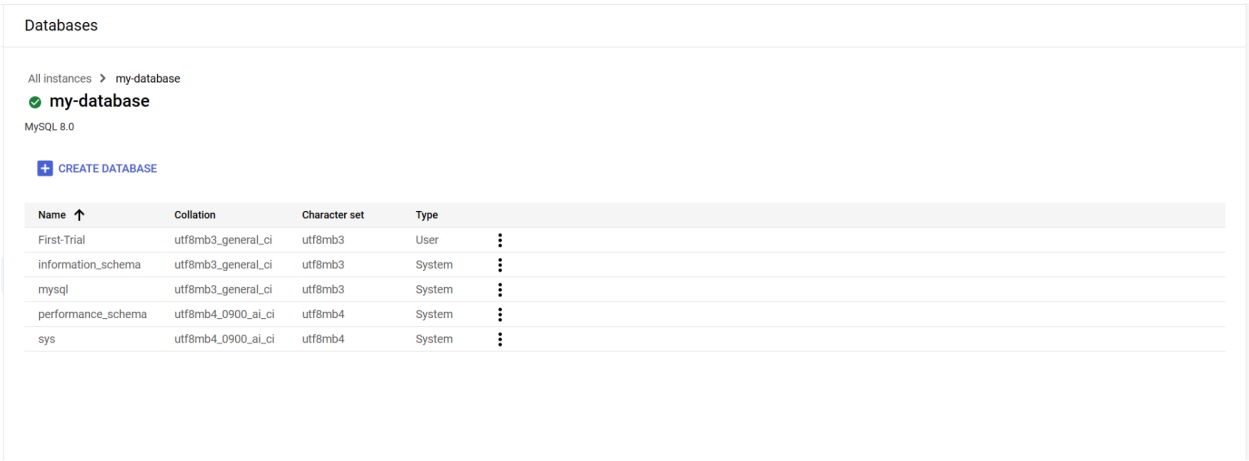
For the following SQL Database, we first create the sql instance, in my case, I create “my-database” as the sql instance in which I will create the databases and connect the web server to the following database. The database instance “my-database” which has a public ip :



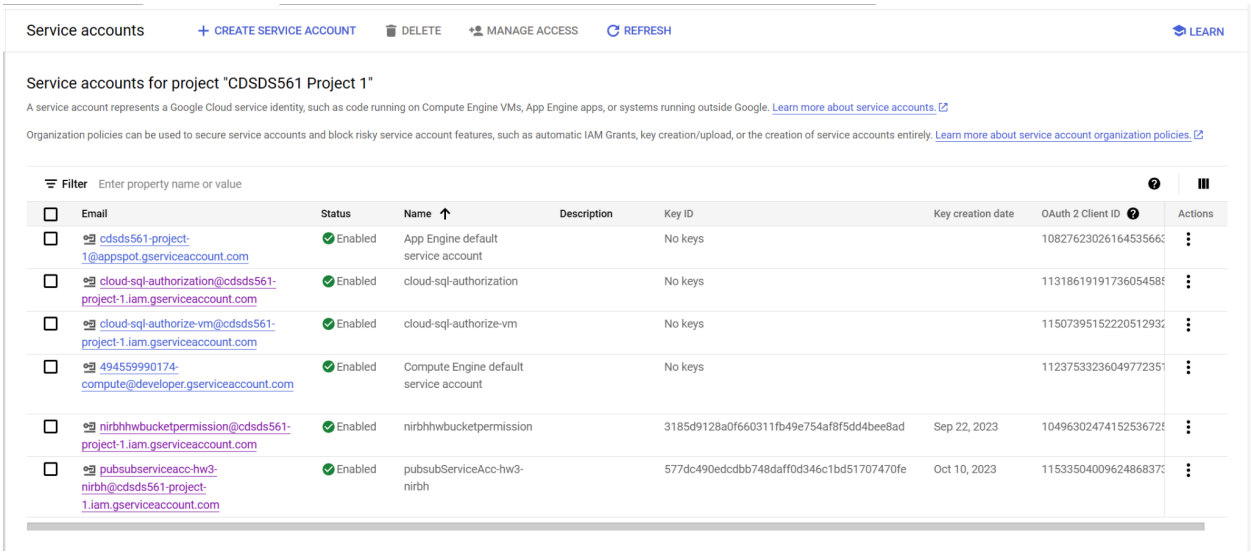
Once the following instance is created we can head to the user section and then change the password of the root (Username) to our liking, in my case the password is “admin” that’s what I have mentioned in the web server python flask application code.

Now we can navigate to the databases section and then generate a new database “First-Trial” in my case. Following Database will be used by our web server to create and update tables like “main_table”, “error_log” and “Clients”.

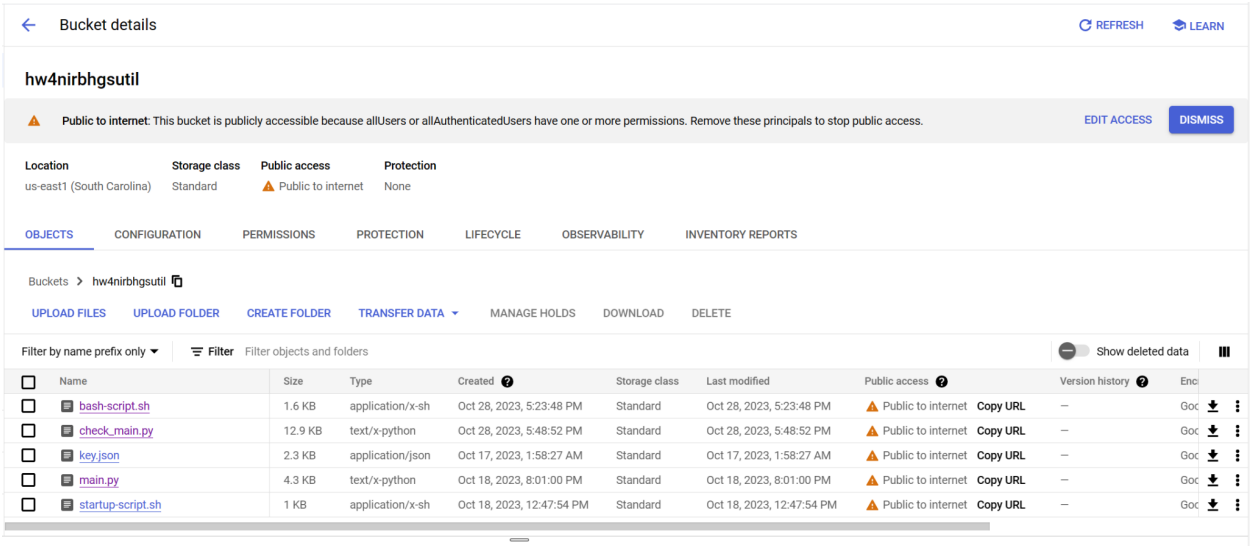
First-Trial database looks like this :



Now, once the following database is created, we can make the service account with the following privileges: Cloud SQL Client and Cloud SQL Admin. I used the following service account (cloud-sql-authorize-vm@cdsds561-project-1.iam.gserviceaccount.com) and gave it the above mentioned privileges.



For the following first application, I created the python flask application that is the webserver running on the instance-1 based on the startup script and I also stored the key for the topic (key.json) , startup script (bash-script.sh) and the check_main.py file in the “hw4nirbhgsutil” bucket:



The following vm instance 1 is created with the following service account : cloud-sql-authorize-vm@cdsds561-project-1.iam.gserviceaccount.com and with the startup script as bash-script and the bash script being the startup script installs the following:

```
/usr/bin/wget
https://dev.mysql.com/get/Downloads/MySQL-8.2/mysql-server_8.2.0-1debian11
_amd64.deb-bundle.tar
/bin/tar -xvf /path/to/mysql-server_8.2.0-1debian11_amd64.deb-bundle.tar
sudo apt-get install libaio1
sudo dpkg-preconfigure mysql-community-server_*.deb
sudo dpkg -i mysql*.deb
sudo apt-get -f install
python3 -m pip install pymysql
python3 -m pip install sqlalchemy
python3 -m pip install "cloud-sql-python-connector[pymysql]"
```

My bash-script looks like the following:

```
C:\Users\nirbh\AppData\Local\Google\Cloud SDK>cat bash-script.sh
#!/bin/bash
mkdir -p ./tmp
cd ./tmp/
echo "Updating package list..."
sudo apt update -y
echo "Installing Python 3..."
sudo apt install -y python3
echo "Installing Python 3 pip..."
sudo apt install -y python3-distutils python3-lib2to3
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
echo "Installing Google Cloud Python libraries..."
python3 -m pip install google-cloud-storage
python3 -m pip install google-cloud-logging
python3 -m pip install google-cloud-pubsub
python3 -m pip install google-auth
python3 -m pip install google-auth-oauthlib
python3 -m pip install google-cloud-storage

python3 -m pip install flask
echo "Installing Google Cloud SDK..."
sudo apt-get install -y google-cloud-sdk


echo "Installing Google Cloud SQL..."

sleep 30
/usr/bin/wget https://dev.mysql.com/get/Downloads/MySQL-8.2/mysql-server_8.2.0-1debian11_amd64.deb-bundle.tar
/bin/tar -xvf /path/to/mysql-server_8.2.0-1debian11_amd64.deb-bundle.tar

sudo apt-get install libaio1


echo "Check - 1 "

sudo dpkg-preconfigure mysql-community-server_*.deb

echo "Check - 2 "

sudo dpkg -i mysql*.deb

echo "Check - 3 "

sudo apt-get -f install


echo "Check - 4 "
python3 -m pip install pymysql

python3 -m pip install sqlalchemy

python3 -m pip install "cloud-sql-python-connector[pymysql]"

GCS_BUCKET="hw4nirbhgsutil"
OBJECT_NAME="key.json"
gsutil cp gs://$GCS_BUCKET/$OBJECT_NAME .
GCS_BUCKET="hw4nirbhgsutil"
OBJECT_NAME="check_main.py"
echo "Copying $OBJECT_NAME from Google Cloud Storage..."
gsutil cp gs://$GCS_BUCKET/$OBJECT_NAME .
echo "Executing check_main.py..."

python3 check_main.py
```

And then we create the vm (f1-micro) using the startup script and using the following command on gcloud cli:

```
gcloud compute instances create instance-1 ^
    --project=cdsds561-project-1 ^
    --zone=us-east1-b ^
    --machine-type=f1-micro ^
    --network-interface=network-tier=PREMIUM,stack-type=IPV4_ONLY,subnet=default ^
    --maintenance-policy=MIGRATE ^
    --provisioning-model=STANDARD ^
    --service-account=cloud-sql-authorize-vm@cdsds561-project-1.iam.gserviceaccount.com ^
    --scopes=https://www.googleapis.com/auth/cloud-platform ^
    --tags=http-server,https-server,lb-health-check ^
    --create-disk=auto-delete=yes,boot=yes,device-name=instance-6,image=projects/debian-cloud/global/images/debian-11-bullseye-v20231010,mode=rw,size=10,type=projects/cdsds561-pro
ject-1/zones/us-central1-a/diskTypes/pd-balanced ^
    --no-shielded-secure-boot ^
    --shielded-vtpm ^
    --shielded-integrity-monitoring ^
    --labels=goog-ec-src=vm_add-gcloud ^
    --reservation-affinity=any ^
    --metadata startup-script-url=gs://hw4nirbhgsutil/bash-script.sh
```

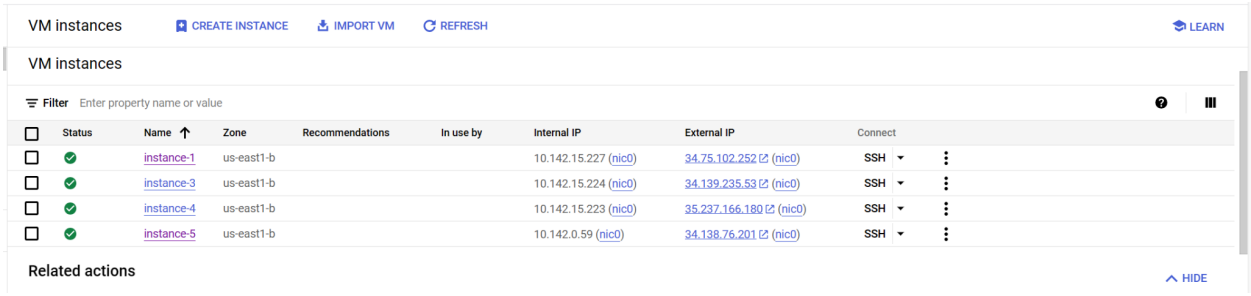
Similar to previous homeworks, I created the bat file for storing the generating vm command and I executed the .bat file in order to generate the vm instance 1 with the service account with sql admin and client privileges. The following generate command also contains the startup script (bash-script) extracted from the bucket to get the server running at the vm generation.

Or you can use an e2-small machine to make it work faster, you can use the following bat file for that :

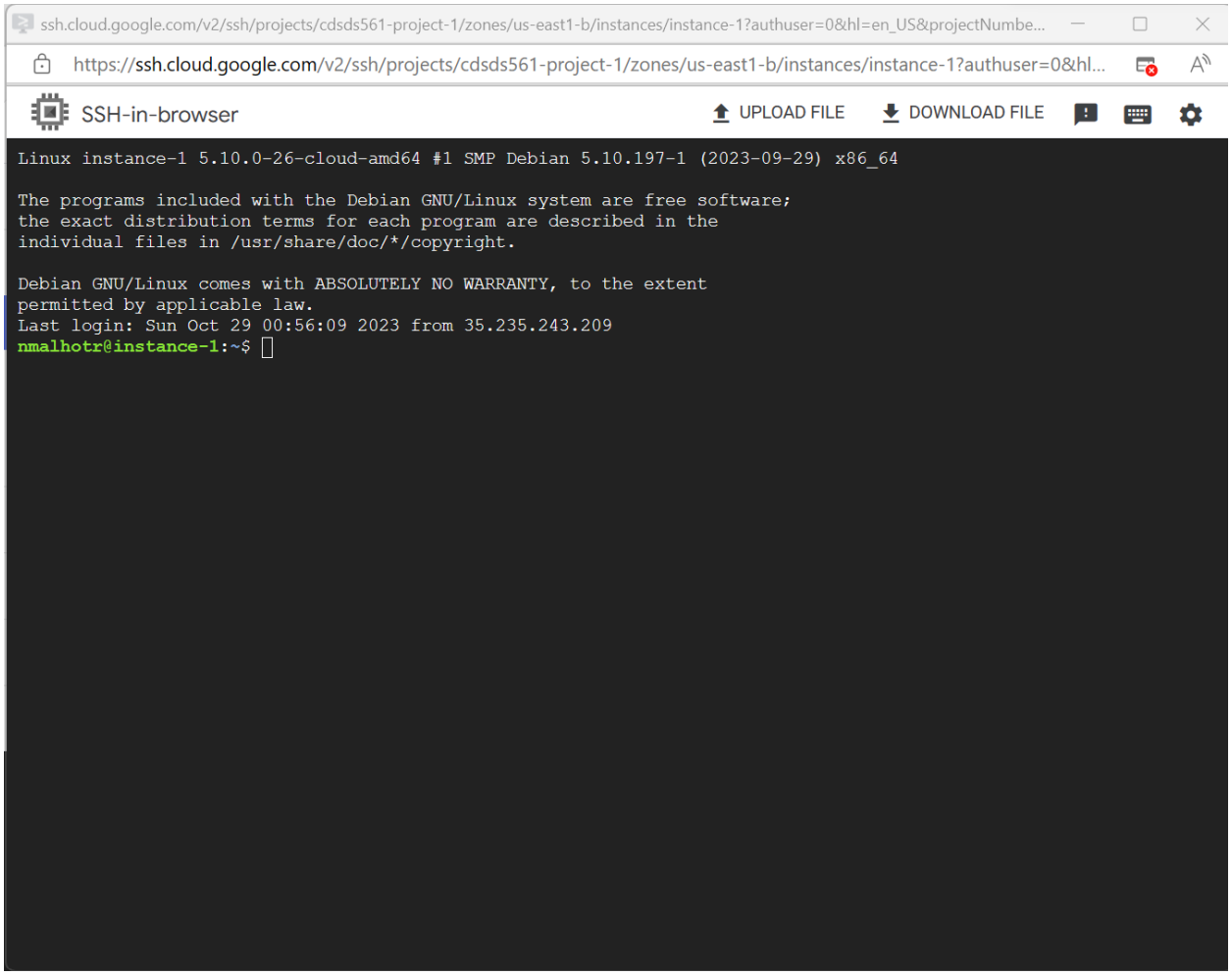
```
C:\Users\nirbh\AppData\Local\Google\Cloud SDK>gcloud compute instances create instance-1     --project=cdsds561-project-1     --zone=us-east1-b     --machine-type=e2-small     --network-interface=network-tier=
PREMIUM,stack-type=IPV4_ONLY,subnet=default     --maintenance-policy=MIGRATE     --provisioning-model=STANDARD     --service-account=cloud-sql-authorize-vm@cdsds561-project-1.iam.gserviceaccount.com     --scop
es=https://www.googleapis.com/auth/cloud-platform     --tags=http-server,https-server,lb-health-check     --create-disk=auto-delete=yes,boot=yes,device-name=instance-6,image=projects/debian-cloud/global/images
/debian-11-bullseye-v20231010,mode=rw,size=10,type=projects/cdsds561-project-1/zones/us-central1-a/diskTypes/pd-balanced     --no-shielded-secure-boot     --shielded-vtpm     --shielded-integrity-monitoring
  --labels=goog-ec-src=vm_add-gcloud     --reservation-affinity=any     --metadata startup-script-url=gs://hw4nirbhgsutil/bash-script.sh
Created [https://www.googleapis.com/compute/v1/projects/cdsds561-project-1/zones/us-east1-b/instances/instance-1].
NAME        ZONE        MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP    EXTERNAL_IP   STATUS
instance-1  us-east1-b  e2-small                   10.142.15.228  34.23.87.202  RUNNING


Updates are available for some Google Cloud CLI components.  To install them,
please run:
  $ gcloud components update
```

Once the vm instance 1 is created, we can ssh the vm instance from the vm instances page:



Using SSH for the instance-1:



Once we ssh into the vm, we can check the database (‘my-database’) to check if the tables gets created in them. So we first use the database using the sql command “USE First-Trial;” and then we can execute the following sql command “SHOW TABLES;” once running this we see the following results:

```
mysql> USE First-Trial
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES
    -> ;
+------------------------+
| Tables_in_First-Trial  |
+------------------------+
| Clients                |
| error_log              |
| main_table             |
+------------------------+
3 rows in set (0.00 sec)
```

Once these tables are created , you can see the architecture of the following tables as the main_table is the one that stores the request's information. The main table follows the Second Normal Form Schema where I disintegrated the main_table into two tables main_table and the Clients to make the main_table follow the 2 NF schema. The main table architecture looks as follows:

```
mysql> desc main_table;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id             | int          | NO   | PRI | NULL    | auto_increment |
| client_id      | int          | NO   | MUL | 1       |                |
| country        | varchar(255) | NO   |     | NULL    |                |
| client_ip      | varchar(255) | NO   |     | NULL    |                |
| is_banned      | tinyint(1)   | NO   |     | NULL    |                |
| time_of_day    | timestamp    | NO   |     | NULL    |                |
| requested_file | varchar(255) | NO   |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Client table architecture looks as follows:

```
mysql> desc Clients;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| client_id | int          | NO   | PRI | NULL    | auto_increment |
| gender    | varchar(255) | NO   |     | NULL    |                |
| age       | varchar(255) | NO   |     | NULL    |                |
| income    | varchar(255) | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

First Normal Form (1NF):

Atomicity:
The columns in the main_table appear to contain atomic values as they are single data elements like integers, strings, and boolean values, so all columns contain atomic values. The single cell of the main_table doesn't hold more than one value as all the columns in the following table contain single indivisible values. So the atomicity condition is met.

Unique Column Names:
The column names in the main_table are unique so this condition is also met.

Order of Data:
In SQL, the order of data in a table does not matter, so this condition is met.

Primary Key:
Main table has "id" as an auto-incremented primary key which allows for identification for each row in the table so the condition of having a primary key for identification is met.

No Duplicated Rows or Columns:
Main table doesn't have duplicated rows as id is an auto incremented primary key and it ensures that each row has a unique identifier. Therefore, there are no duplicated rows and columns.

Each column must have only one value for each row:
Each column in the main_table contains a single value for each row. For example, the country column holds a single country value for each row and the same is true for other columns. Therefore, this condition is also met.

Hence, the main_table satisfies the criteria for 1NF.

Second Normal Form (2 NF):

It's already in 1NF:
The main_table meets the criteria for the First Normal Form schema as it contains atomic values, has a primary key for identification, doesn't have duplicated rows, and each column holds only one value for each row. Therefore, it is in 1 NF.

No Partial Dependency:
The second criteria for 2NF is that there should be no partial dependencies, which means all non-key attributes should be fully dependent on the entire primary key and not just a part of it. In the main_table, the primary key consists of the id and client id is a foreign key that references the client id in the Clients table.

Non-key attributes in the main_table like country, client_ip , is_banned, time_of__day and the request_file are fully dependent on the entire primary key which is the combination of id and

client id which means that each record in the main_table is uniquely identified by this composite key and all other columns are dependent on and there are no partial dependencies because all non-key attributes are fully dependent on the primary key. This adherence to 2NF ensure that the main_table does not suffer from the partial dependencies and has been properly normalized.

And the error_log table architecture looks as follows:

```
mysql> desc error_log;
+-----------------+-----------------+------+-----+---------+----------------+
| Field           | Type            | Null | Key | Default | Extra          |
+-----------------+-----------------+------+-----+---------+----------------+
| id              | bigint unsigned | NO   | PRI | NULL    | auto_increment |
| time_of_request | timestamp       | NO   |     | NULL    |                |
| requested_file  | varchar(255)    | NO   |     | NULL    |                |
| error_code      | int             | NO   |     | NULL    |                |
+-----------------+-----------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

The error_log is following 1 NF:

Atomicity:
Each column in the error_log table contains atomic or indivisible values as id is a primary key , time of request is a timestamp, request file is a string and error_code is an integer. These all columns are atomic data types so the atomicity condition is met.

Primary Key:
The table has a primary key, which is the “id” column which allows for identification of each record, meeting the requirements of having a primary key.

No Duplicated Rows or Columns:
The table should not have duplicated rows and columns. All the column names in the tables are distinct and assuming the “id” column is an auto incremented primary key, each row will have a unique identifier which avoids duplicate rows.

Each column must have only one value for each row:
Each column in the error_log table contains a single value for each row. For instance, the time_of_request column holds a timestamp and the request_file column holds a single file for each row. Therefore, this condition is met.

Therefore, the error_log table is in the First Normal Form (1 NF) as it contains atomic values, has a primary key for identification, doesn’t have duplicated rows and each column holds only one value for each row.

Second Normal Form (2 NF):

Already in 1 NF:

The following error_log table is in 1 NF (First Normal Form) as discussed above as it has atomicity, a primary key , no duplicated rows or columns and each column has only one value for each row.

No Partial Dependencies:

The second normal form (2 NF) criteria requires that there should be no partial dependencies which means that all non-key attributes must be fully dependent on the entire primary key. In the error_log table. “Id” is the primary key and the non key attributes such as “time_of_request” , “requested_file” , and “ error_code” are fully dependent on the entire primary key, which is “id”. This is because each record in the “error_log” table is uniquely identified by the “id” and all other attributes are dependent on the specific value of the “id” for that record. There are no partial dependencies. As a result, the error_log table complies with the Second Normal Form criteria and it is already in 1NF and doesn’t have any partial dependencies as all non key attributes are fully dependent on the primary key “id”.

Once the following tables are created, we can then use the http client to check for the request and the responses. We can use the instance-5 for executing the http client for making the request from the instance-5 and once the request is made, we can check the response as follows:

```
nmalhotr@instance-5:~$ python3 http-client.py -d 34.75.102.252 -b /hw2nirbhgsutil -w test-dir -n 1 -i 1000 -v
Requesting  /hw2nirbhgsutil/test-dir/892.html  from  34.75.102.252 80
200 OK
Server: Werkzeug/3.0.1 Python/3.9.2
Date: Sun, 29 Oct 2023 01:54:25 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 46131
Connection: close


b'<!DOCTYPE html>\r\n<html>\r\n<body>\r\nLorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmo
d tempor incididunt ut labore et dolore magna aliqua. Ut enim ad\r\nminim veniam, quis nostrud exercitation ull
amco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
 esse\r\ncillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.\r\n<p>\r\n<a HREF="4450.html"> This is a link </a>\r\n<p>\r\nL
orem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore ma
gna aliqua. Ut enim ad\r\nminim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
 consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse\r\ncillum dolore eu fugiat nulla par
iatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est la
borum.\r\n<p>\r\n<a HREF="370.html"> This is a link </a>\r\n<p>\r\nLorem ipsum dolor sit amet, consectetur adip
iscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad\r\nminim veniam, qui
s nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in repreh
enderit in voluptate velit esse\r\ncillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat no
n proident, sunt in culpa qui officia deserunt mollit anim id est laborum.\r\n<p>\r\n<a HREF="6659.html"> This
is a link </a>\r\n<p>\r\nLorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incidid
unt ut labore et dolore magna aliqua. Ut enim ad\r\nminim veniam, quis nostrud exercitation ullamco laboris nis
i ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse\r\ncillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia dese
runt mollit anim id est laborum.\r\n<p>\r\n<a HREF="1691.html"> This is a link </a>\r\n<p>\r\nLorem ipsum dolor
 sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut e
nim ad\r\nminim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis
 aute irure dolor in reprehenderit in voluptate velit esse\r\ncillum dolore eu fugiat nulla pariatur. Excepteur
 sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.\r\n<p>\r\
n<a HREF="588.html"> This is a link </a>\r\n<p>\r\nLorem ipsum dolor sit amet, consectetur adipiscing elit, sed
 do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad\r\nminim veniam, quis nostrud exerci
tation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in volup
```

Now we can check the sql tables for the entry created for the following request:

Main Table Entry:

```
mysql> SELECT * FROM main_table;
+----+-----------+--------------------+----------------+-----------+---------------------+----------------+
| id | client_id | country            | client_ip      | is_banned | time_of_day         | requested_file |
+----+-----------+--------------------+----------------+-----------+---------------------+----------------+
|  1 |         1 | Dominican Republic | 134.210.37.218 |         0 | 2023-10-29 00:00:00 | 892.html       |
+----+-----------+--------------------+----------------+-----------+---------------------+----------------+
1 row in set (0.00 sec)
```

Clients Table Entry:

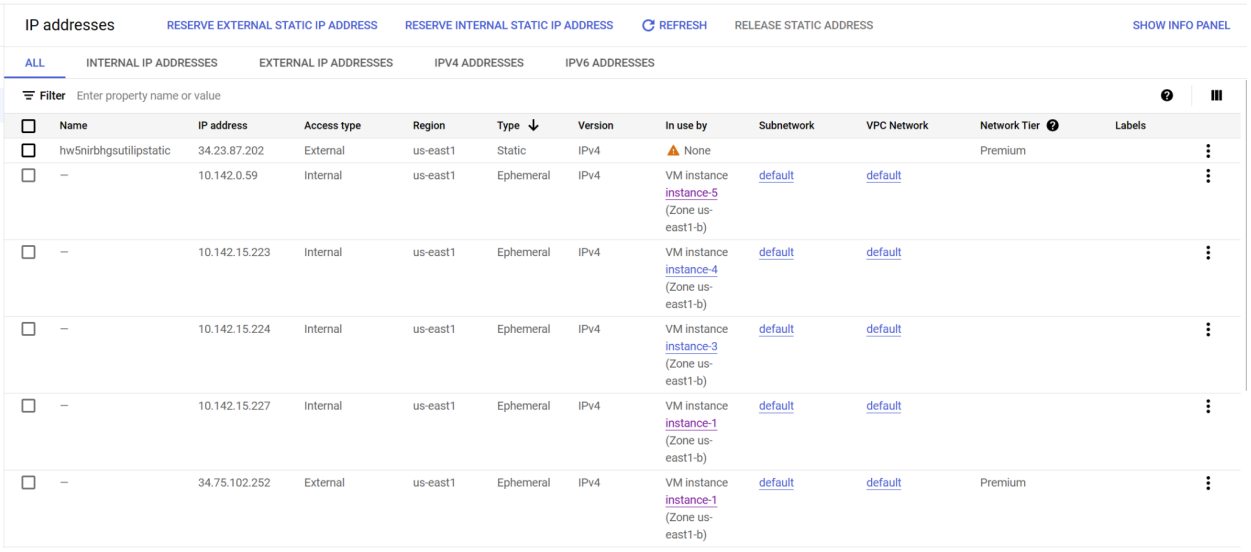
```
mysql> SELECT * FROM Clients;
+-----------+--------+-------+---------+
| client_id | gender | age   | income  |
+-----------+--------+-------+---------+
|         1 | Male   | 56-65 | 10k-20k |
+-----------+--------+-------+---------+
1 row in set (0.00 sec)
```

Error_log Entry:

```
mysql> SELECT * FROM error_log;
Empty set (0.00 sec)
```

Error_log table is empty because there was no error here.

Now, we can reserve the ip address of the vm instance 1, for the following we can navigate to the VPC network and then head to the IP addresses where you can head to the reserve external static IP address and then provide a name of the following and then choose the vm instance as instance -1. Reserved External IP address will look as follows:



Once, the external ip address of the vm instance 1 is reserved then we can check for the curl commands and the erroneous requests.

Curl Commands:

CURL GET REQUEST:

We can use instance 5 for the curl get request and get the response as follows:

```
nmalhotr@instance-5:~$ curl -X GET http://34.75.102.252/hw2nirbhgsutil/test-dir/102.html
<!DOCTYPE html>
<html>
<body>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="2729.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="8430.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="3505.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="5355.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="5096.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="523.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="4045.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
<p>
<a HREF="8559.html"> This is a link </a>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
```

SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+----------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
1 row in set (0.01 sec)
```

Clients Table:

```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
1 row in set (0.01 sec)
```

Error_log Table:

```
mysql>
mysql> SELECT * FROM error_log;
Empty set (0.00 sec)

mysql>
```

CURL FILE NOT FOUND REQUEST:

Now we can use curl for sending a get request for the file that is not available in the bucket. We will get the following result for the following curl command:

```
</html>
nmalhotr@instance-5:~$ curl -X GET http://34.75.102.252/hw2nirbhgsutil/test-dir/100002.html
File Not Foundnmalhotr@instance-5:~$
```

We will get the File Not Found error for the following file request.

SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+----------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
1 row in set (0.00 sec)
```

Clients Table:

```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
1 row in set (0.00 sec)
```

Error_log Table:

```
mysql> SELECT * FROM error_log;
+----+---------------------+----------------+------------+
| id | time_of_request     | requested_file | error_code |
+----+---------------------+----------------+------------+
|  1 | 2023-10-29 03:17:52 | 100002.html    |        404 |
+----+---------------------+----------------+------------+
1 row in set (0.00 sec)
```

CURL Method Not Implemented Request:
Now we can use curl for sending a delete request for the file available in the bucket. We will get the following result for the following curl command:

```
nmalhotr@instance-5:~$ curl -X GET http://34.75.102.252/hw2nirbhgsutil/test-dir/100002.html
File Not Foundnmalhotr@curl -X DELETE http://34.75.102.252/hw2nirbhgsutil/test-dir/102.html
Method Not Implementednmalhotr@instance-5:~$
```

We will get the Method Not Implemented error for the following file request.

SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+----------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
1 row in set (0.01 sec)
```

Clients Table:

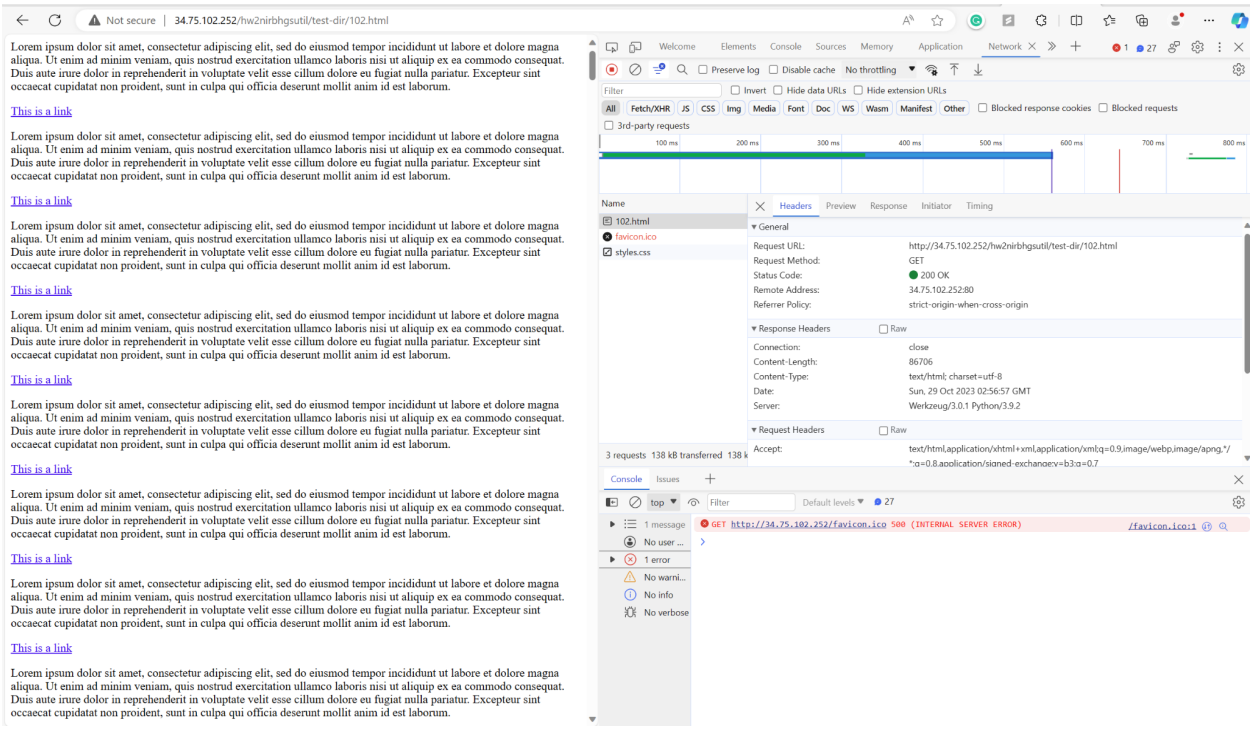
```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
1 row in set (0.00 sec)
```

Error_log Table:

```
mysql> SELECT * FROM error_log;
+----+---------------------+----------------+------------+
| id | time_of_request     | requested_file | error_code |
+----+---------------------+----------------+------------+
|  1 | 2023-10-29 03:17:52 | 100002.html    |        404 |
|  2 | 2023-10-29 03:24:26 | 102.html       |        501 |
+----+---------------------+----------------+------------+
2 rows in set (0.00 sec)
```

BROWSER GET REQUEST:

We can browse for the file name from the bucket using the browser using the external ip of the vm instance-1. The browser get request will look as follows:



SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+----------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
|  2 |         2 | Unknown | Unknown   |         0 | 2023-10-29 03:30:45 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
2 rows in set (0.00 sec)
```

Clients Table:

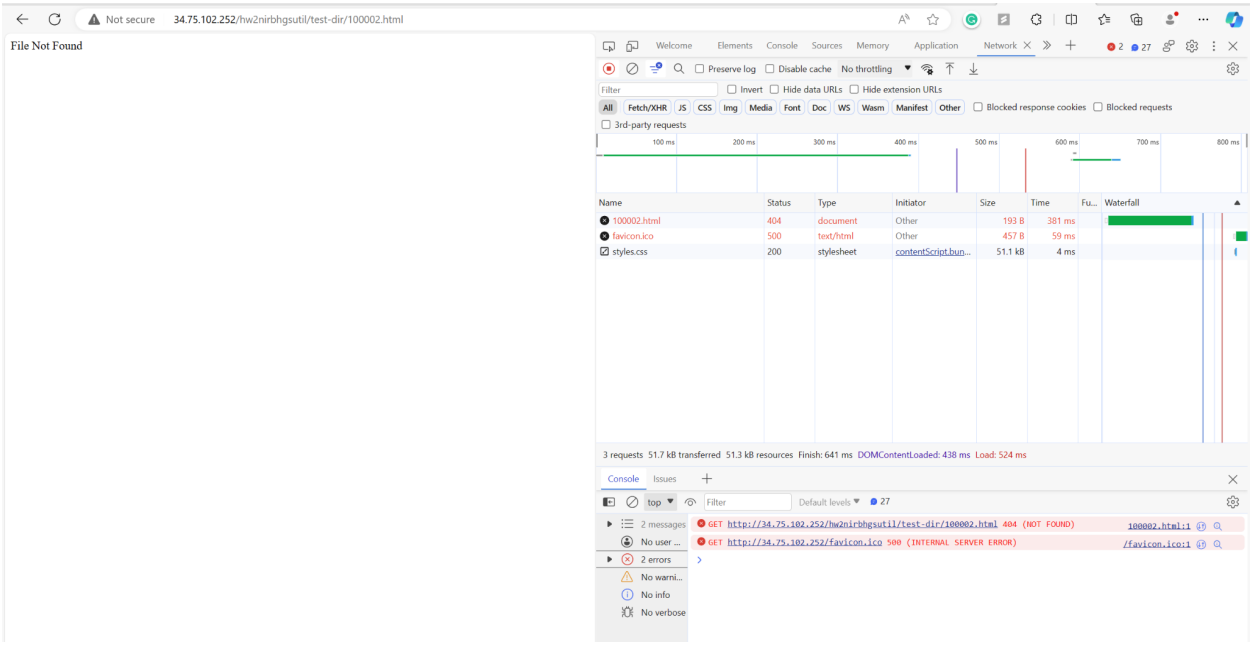
```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
|         2 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
2 rows in set (0.00 sec)
```

Error_log Table:

```
mysql> SELECT * FROM error_log;
+----+---------------------+----------------+------------+
| id | time_of_request     | requested_file | error_code |
+----+---------------------+----------------+------------+
|  1 | 2023-10-29 03:17:52 | 100002.html    |        404 |
|  2 | 2023-10-29 03:24:26 | 102.html       |        501 |
+----+---------------------+----------------+------------+
2 rows in set (0.00 sec)
```

BROWSER FILE NOT FOUND REQUEST:

We can browse for the file name not available in the bucket using the browser using the external ip of the vm instance-1. The browser get request will look as follows:



SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+----------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
|  2 |         2 | Unknown | Unknown   |         0 | 2023-10-29 03:30:45 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
2 rows in set (0.00 sec)
```

Clients Table:

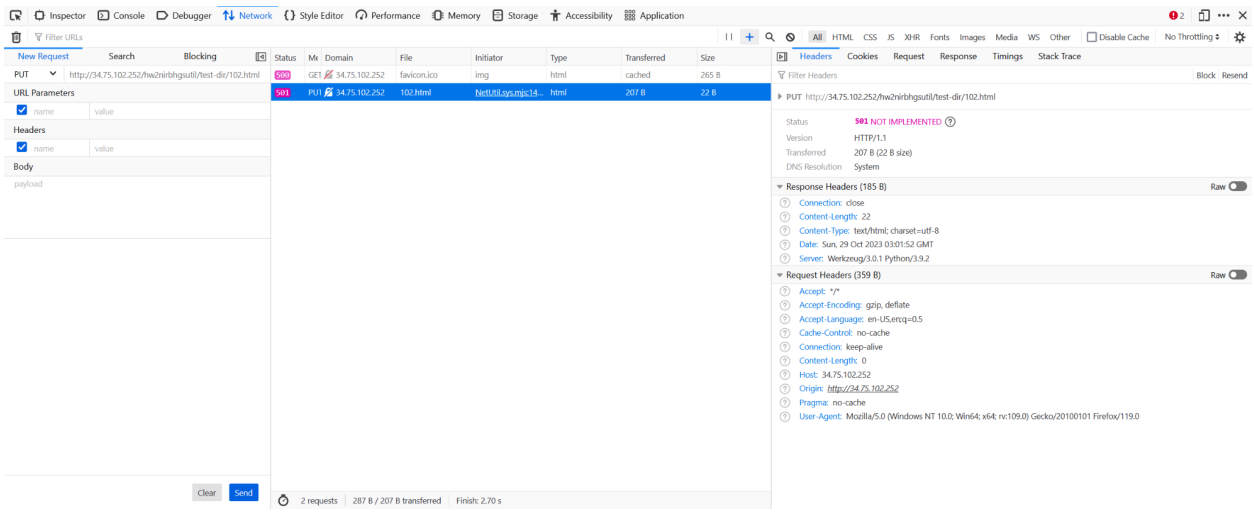
```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
|         2 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
2 rows in set (0.01 sec)
```

Error_log Table:

```
mysql> SELECT * FROM error_log;
+----+---------------------+----------------+------------+
| id | time_of_request     | requested_file | error_code |
+----+---------------------+----------------+------------+
|  1 | 2023-10-29 03:17:52 | 100002.html    |        404 |
|  2 | 2023-10-29 03:24:26 | 102.html       |        501 |
|  3 | 2023-10-29 03:39:17 | 100002.html    |        404 |
+----+---------------------+----------------+------------+
3 rows in set (0.00 sec)
```

BROWSER METHOD NOT IMPLEMENTED REQUEST:

We can browse for the file name available in the bucket using the browser using the external ip of the vm instance-1. The browser PUT request will look as follows:



SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+---------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+---------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
|  2 |         2 | Unknown | Unknown   |         0 | 2023-10-29 03:30:45 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+---------------+
2 rows in set (0.00 sec)
```

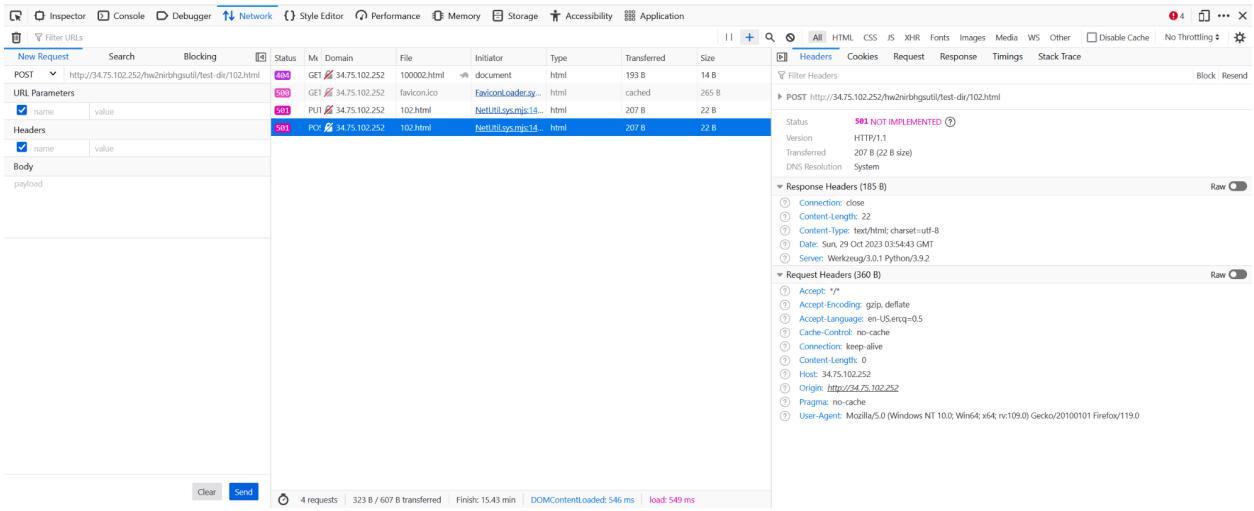
Clients Table:

```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
|         2 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
2 rows in set (0.00 sec)
```

Error_log Table:

```
mysql> SELECT * FROM error_log;
+----+---------------------+----------------+------------+
| id | time_of_request     | requested_file | error_code |
+----+---------------------+----------------+------------+
|  1 | 2023-10-29 03:17:52 | 100002.html    |        404 |
|  2 | 2023-10-29 03:24:26 | 102.html       |        501 |
|  3 | 2023-10-29 03:39:17 | 100002.html    |        404 |
|  4 | 2023-10-29 03:49:24 | 102.html       |        501 |
+----+---------------------+----------------+------------+
4 rows in set (0.00 sec)
```

We can browse for the file name available in the bucket using the browser using the external ip of the vm instance-1. The browser POST request will look as follows:



SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+----------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
|  2 |         2 | Unknown | Unknown   |         0 | 2023-10-29 03:30:45 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
2 rows in set (0.01 sec)
```

Client Table:

```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
|         2 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
2 rows in set (0.00 sec)
```

Error_Log Table:

```
mysql> SELECT * FROM error_log;
+----+---------------------+----------------+------------+
| id | time_of_request     | requested_file | error_code |
+----+---------------------+----------------+------------+
|  1 | 2023-10-29 03:17:52 | 100002.html    |        404 |
|  2 | 2023-10-29 03:24:26 | 102.html       |        501 |
|  3 | 2023-10-29 03:39:17 | 100002.html    |        404 |
|  4 | 2023-10-29 03:49:24 | 102.html       |        501 |
|  5 | 2023-10-29 03:54:43 | 102.html       |        501 |
+----+---------------------+----------------+------------+
5 rows in set (0.00 sec)
```

CURL Method Not Implemented Request:

Now we can use curl for sending a TRACE request for the file available in the bucket. We will get the following result for the following curl command:

```
</html>
nmalhotr@instance-5:~$ curl -X GET http://34.75.102.252/hw2nirbhgsutil/test-dir/100002.html
File Not Foundnmalhotr@curl -X DELETE http://34.75.102.252/hw2nirbhgsutil/test-dir/102.html
nmalhotr@instance-5:~$ curl -X TRACE http://34.75.102.252/hw2nirbhgsutil/test-dir/102.html
Method Not Implementednmalhotr@instance-5:~$
```

We will get the Method Not Implemented error for the following file request.

SQL Tables after Following Actions:

Main_Table:

```
mysql> SELECT * FROM main_table;
+----+-----------+---------+-----------+-----------+---------------------+----------------+
| id | client_id | country | client_ip | is_banned | time_of_day         | requested_file |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
|  1 |         1 | Unknown | Unknown   |         0 | 2023-10-29 03:13:03 | 102.html       |
|  2 |         2 | Unknown | Unknown   |         0 | 2023-10-29 03:30:45 | 102.html       |
+----+-----------+---------+-----------+-----------+---------------------+----------------+
2 rows in set (0.01 sec)
```

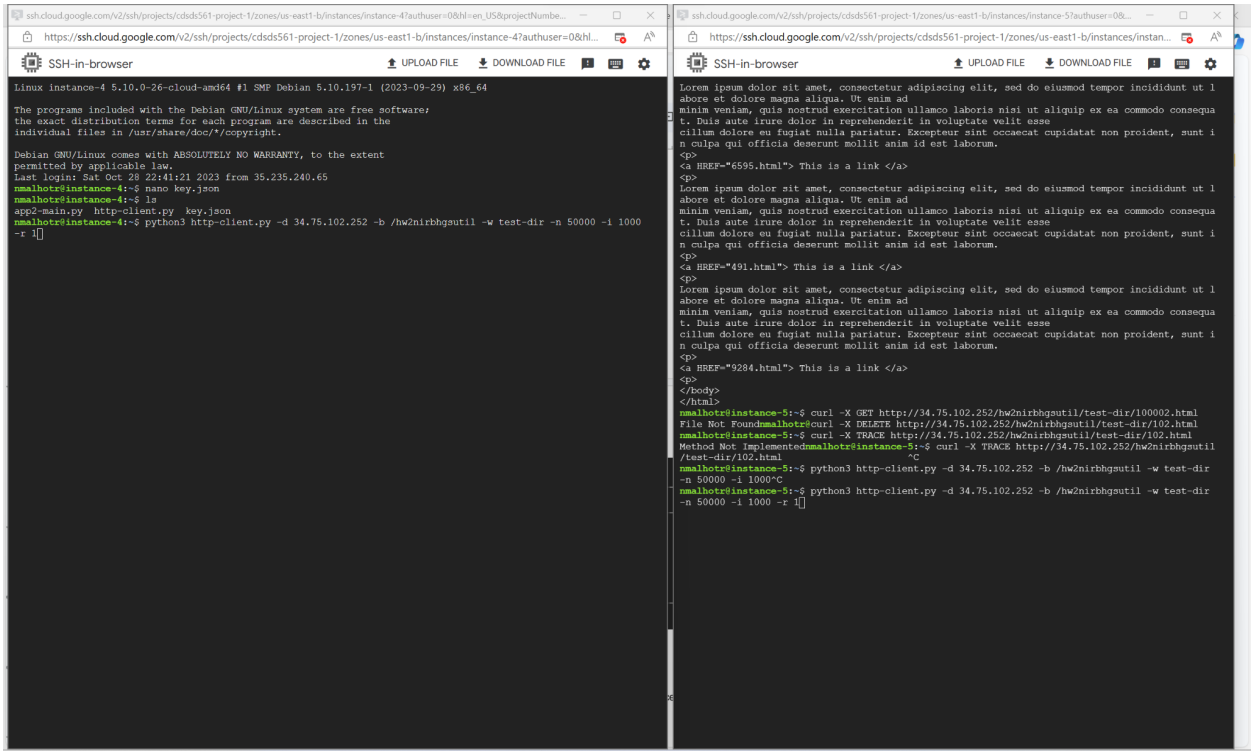
Client Table:

```
mysql> SELECT * FROM Clients;
+-----------+---------+---------+---------+
| client_id | gender  | age     | income  |
+-----------+---------+---------+---------+
|         1 | Unknown | Unknown | Unknown |
|         2 | Unknown | Unknown | Unknown |
+-----------+---------+---------+---------+
2 rows in set (0.01 sec)
```

Error_Log Table:

```
mysql> SELECT * FROM error_log;
+----+---------------------+---------------+------------+
| id | time_of_request     | requested_file | error_code |
+----+---------------------+---------------+------------+
|  1 | 2023-10-29 03:17:52 | 100002.html    |        404 |
|  2 | 2023-10-29 03:24:26 | 102.html       |        501 |
|  3 | 2023-10-29 03:39:17 | 100002.html    |        404 |
|  4 | 2023-10-29 03:49:24 | 102.html       |        501 |
|  5 | 2023-10-29 03:54:43 | 102.html       |        501 |
|  6 | 2023-10-29 03:57:36 | 102.html       |        501 |
+----+---------------------+---------------+------------+
6 rows in set (0.00 sec)
```

For the concurrent client requests, I used instance 4 and instance 5 for executing the http client and sending the 50,000 requests from both of the instances.



Using the '-r' arguments for the random seed to be the same for both of the http client requests.

Once the following http-client requests are made, we can check the sql database tables for verifying if 100,000 requests were made:

We can first check the total entries in the main_table:

```
mysql> USE First-Trial
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT COUNT(*) FROM main_table;
+----------+
| COUNT(*) |
+----------+
|    95376 |
+----------+
1 row in set (0.01 sec)
```

The total entries for the main_table is 95376 files requested.

And then we can check the total entries in the error_log table:

```
mysql> SELECT COUNT(*) FROM error_log WHERE requested_file LIKE '%.html';
+----------+
| COUNT(*) |
+----------+
|     4624 |
+----------+
1 row in set (0.00 sec)
```

The total entries for the error_log table is 4624 files requested.

Now the total files requested will be entries from main_table + entries from error_log table which will be 95376 + 4624 = 100,000 files.

Now we can check the following statistics from my requests:

Total Requests that were made successfully vs unsuccessfully:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT 'Successful' AS request_type, COUNT(*) AS count
    -> FROM main_table
    -> UNION
    -> SELECT 'Unsuccessful' AS request_type, COUNT(*) AS count
    -> FROM error_log
    -> WHERE requested_file LIKE '%.html';
+--------------+-------+
| request_type | count |
+--------------+-------+
| Successful   | 95376 |
| Unsuccessful |  4624 |
+--------------+-------+
2 rows in set (0.01 sec)
```

Total Requests that came from banned countries:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT COUNT(*) AS request_count
    -> FROM error_log
    -> WHERE error_code = 403
    -> AND requested_file LIKE '%.html'
    -> ORDER BY request_count;
+---------------+
| request_count |
+---------------+
|          4624 |
+---------------+
1 row in set (0.00 sec)
```

We got the following number as I made a new database for the concurrent http client requests.So we won't see any 501 method not implemented for the following case.

Total Requests made by Male and Female Users:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT gender, COUNT(*) as request_count
    -> FROM Clients
    -> GROUP BY gender;
+--------+---------------+
| gender | request_count |
+--------+---------------+
| Male   |         47762 |
| Female |         47614 |
+--------+---------------+
2 rows in set (0.08 sec)
```

For the following query result, we can see that since the gender column is in the Clients table, we query the Clients table for the following, and then we can add the male and female entries to determine the total entries of the client table which will be equal to the main_table entries.

Male Entries + Female Entries = Clients Table Entries = Main_Table Entries = 47762 + 47614 = 95376.

Top 5 countries that sent requests to your server:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql>
mysql> SELECT country
    -> FROM main_table
    -> GROUP BY country
    -> ORDER BY COUNT(*) DESC
    -> LIMIT 5;
+------------+
| country    |
+------------+
| Paraguay   |
| Zambia     |
| Oman       |
| Cyprus     |
| Micronesia |
+------------+
5 rows in set (0.08 sec)
```

Querying the main_table provides the top 5 countries and then we get Paraguay with the highest number of requests to the web server. We can also display the request count with another column as follows:

```
mysql> SELECT country,COUNT(*) as request_count FROM main_table GROUP BY country ORDER BY request_count DESC LIMIT 5;
+------------+---------------+
| country    | request_count |
+------------+---------------+
| Paraguay   |           576 |
| Zambia     |           576 |
| Oman       |           572 |
| Cyprus     |           566 |
| Micronesia |           566 |
+------------+---------------+
5 rows in set (0.09 sec)
```

Age Group that issued the most requests to the server:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT age
    -> FROM Clients
    -> GROUP BY age
    -> ORDER BY COUNT(*) DESC
    -> LIMIT 1;
+-----+
| age |
+-----+
| 76+ |
+-----+
1 row in set (0.08 sec)
```

We can also check the request count made by the following age group as follows:

```
mysql> SELECT age,COUNT(*) AS request_count FROM Clients GROUP BY age ORDER BY COUNT(*) DESC LIMIT 1;
+-----+---------------+
| age | request_count |
+-----+---------------+
| 76+ |         12180 |
+-----+---------------+
1 row in set (0.07 sec)
```

We can also check the request count of other age groups too as follows:

```
mysql> SELECT age,COUNT(*) AS request_count FROM Clients GROUP BY age ORDER BY COUNT(*) DESC LIMIT 5;
+-------+---------------+
| age   | request_count |
+-------+---------------+
| 76+   |         12180 |
| 36-45 |         12092 |
| 26-35 |         11964 |
| 46-55 |         11956 |
| 0-16  |         11920 |
+-------+---------------+
5 rows in set (0.08 sec)
```

Income Group that issued the most requests to the web server :

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT income
    -> FROM Clients
    -> GROUP BY income
    -> ORDER BY COUNT(*) DESC
    -> LIMIT 1;
+---------+
| income  |
+---------+
| 20k-40k |
+---------+
1 row in set (0.08 sec)
```

We can also check the request count made by the following income group as follows:

```
mysql> SELECT income, COUNT(*) AS request_count FROM Clients GROUP BY income ORDER BY COUNT(*) DESC LIMIT 1;
+---------+---------------+
| income  | request_count |
+---------+---------------+
| 20k-40k |         12062 |
+---------+---------------+
1 row in set (0.08 sec)
```

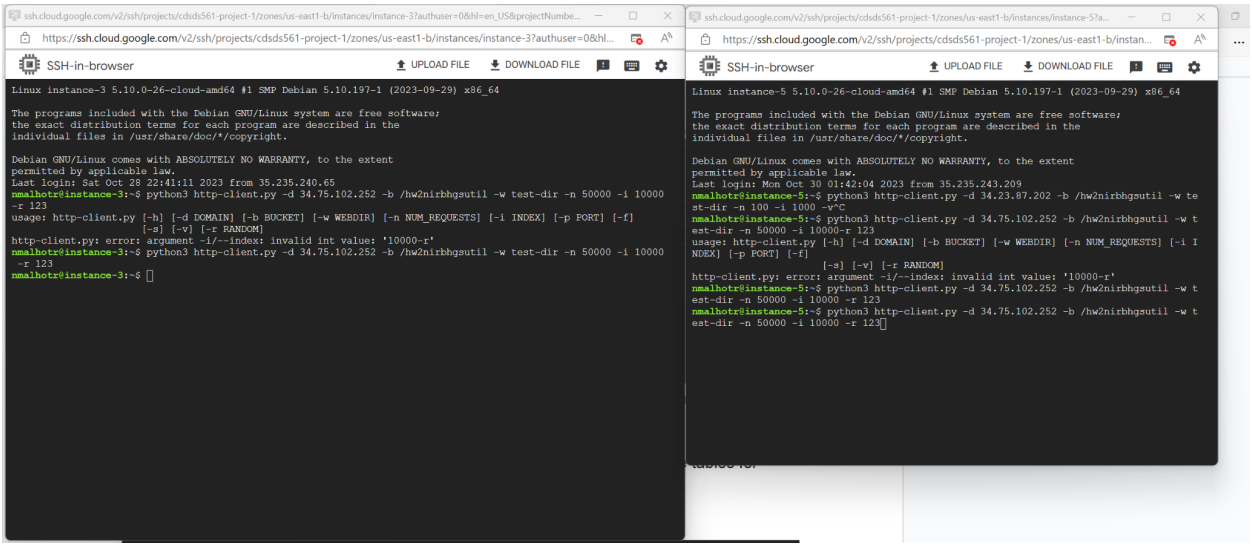
We can also check the request count of other income groups too as follows:

```
mysql> SELECT income, COUNT(*) AS request_count FROM Clients GROUP BY income ORDER BY COUNT(*) DESC LIMIT 5;
+-----------+---------------+
| income    | request_count |
+-----------+---------------+
| 20k-40k   |         12062 |
| 150k-250k |         12058 |
| 0-10k     |         12038 |
| 10k-20k   |         12014 |
| 40k-60k   |         11904 |
+-----------+---------------+
5 rows in set (0.08 sec)
```

For checking the functionality of the vm you can restart the vm instance-1 which means the web server will be running. Then you can run your http client on instance 4, 5 , or 3 and can run app 2 on instance - 4 for checking the banned countries.

Other than this I also added tried for the i being 10,000 the above one was just for 1000. Now using the following I get this:

For the concurrent client requests, I used instance 4 and instance 5 for executing the http client and sending the 50,000 requests from both of the instances.



Using the '-r' arguments for the random seed to be the same for both of the http client requests. I used the r as 123 for both instances.

Once the following http-client requests are made, we can check the sql database tables for verifying if 100,000 requests were made:

We can first check the total entries in the main_table:

```
mysql> SELECT COUNT(*) from main_table where requested_file like '%.html';
+----------+
| COUNT(*) |
+----------+
|    95520 |
+----------+
1 row in set (0.04 sec)
```

The total entries for the main_table is 95520 files requested.

And then we can check the total entries in the error_log table:

```
mysql> SELECT COUNT(*) as requested_file FROM error_log WHERE requested_file LIKE '%.html';
+----------------+
| requested_file |
+----------------+
|           4480 |
+----------------+
1 row in set (0.00 sec)
```

The total entries for the error_log table is 4480 files requested.

Now the total files requested will be entries from main_table + entries from error_log table which will be 95520 + 4480 = 100,000 files.

Now we can check the following statistics from my requests:

Total Requests that were made successfully vs unsuccessfully:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT 'Successful' AS request_type, COUNT(*) AS count
    -> FROM main_table
    -> UNION
    -> SELECT 'Unsuccessful' AS request_type, COUNT(*) AS count
    -> FROM error_log
    -> WHERE requested_file LIKE '%.html';
+--------------+-------+
| request_type | count |
+--------------+-------+
| Successful   | 95520 |
| Unsuccessful |  4480 |
+--------------+-------+
2 rows in set (0.01 sec)
```

Total Requests that came from banned countries:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT COUNT(*) AS request_count
    -> FROM error_log
    -> WHERE error_code = 403
    -> AND requested_file LIKE '%.html'
    -> ORDER BY request_count;
+---------------+
| request_count |
+---------------+
|          4480 |
+---------------+
1 row in set (0.01 sec)
```

We got the following number as I made a new database for the concurrent http client requests.So we won't see any 501 method not implemented for the following case.

Total Requests made by Male and Female Users:

We can use SQL queries for querying the database to get the following statistics as follows:

```
1 row in set (0.01 sec)

mysql> SELECT gender, COUNT(*) as request_count
    -> FROM Clients
    -> GROUP BY gender;
+--------+---------------+
| gender | request_count |
+--------+---------------+
| Male   |         47972 |
| Female |         47548 |
+--------+---------------+
2 rows in set (0.07 sec)
```

For the following query result, we can see that since the gender column is in the Clients table, we query the Clients table for the following, and then we can add the male and female entries to determine the total entries of the client table which will be equal to the main_table entries.

Male Entries + Female Entries = Clients Table Entries = Main_Table Entries = 47972 + 47548 = 95520.

Top 5 countries that sent requests to your server:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT country
    -> FROM main_table
    -> GROUP BY country
    -> ORDER BY COUNT(*) DESC
    -> LIMIT 5;
+----------+
| country  |
+----------+
| Austria  |
| Mongolia |
| Armenia  |
| Andorra  |
| Ukraine  |
+----------+
5 rows in set (0.08 sec)
```

Querying the main_table provides the top 5 countries and then we get Paraguay with the highest number of requests to the web server. We can also display the request count with another column as follows:

```
mysql> SELECT country, COUNT(*) AS request_count FROM main_table GROUP BY country ORDER BY request_count DESC LIMIT 5;
+----------+---------------+
| country  | request_count |
+----------+---------------+
| Austria  |           580 |
| Mongolia |           576 |
| Armenia  |           576 |
| Andorra  |           576 |
| Ukraine  |           576 |
+----------+---------------+
5 rows in set (0.08 sec)
```

Instance reset

Age Group that issued the most requests to the server:

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT age
    -> FROM Clients
    -> GROUP BY age
    -> ORDER BY COUNT(*) DESC
    -> LIMIT 1;
+-----+
| age |
+-----+
| 76+ |
+-----+
1 row in set (0.07 sec)

mysql>
```

We can also check the request count made by the following age group as follows:

```
mysql> SELECT age,count(*) as request_count FROM Clients GROUP BY age ORDER BY request_count DESC LIMIT 1;
+-----+---------------+
| age | request_count |
+-----+---------------+
| 76+ |         12196 |
+-----+---------------+
1 row in set (0.07 sec)
```

We can also check the request count of other age groups too as follows:

```
mysql> SELECT age,count(*) as request_count FROM Clients GROUP BY age ORDER BY request_count DESC LIMIT 5;
+-------+---------------+
| age   | request_count |
+-------+---------------+
| 76+   |         12196 |
| 0-16  |         12166 |
| 17-25 |         11976 |
| 46-55 |         11900 |
| 26-35 |         11892 |
+-------+---------------+
5 rows in set (0.07 sec)
```

Income Group that issued the most requests to the web server :

We can use SQL queries for querying the database to get the following statistics as follows:

```
mysql> SELECT income
    -> FROM Clients
    -> GROUP BY income
    -> ORDER BY COUNT(*) DESC
    -> LIMIT 1;
+---------+
| income  |
+---------+
| 10k-20k |
+---------+
1 row in set (0.08 sec)
```

We can also check the request count made by the following income group as follows:

```
mysql> SELECT income,count(*) as request_count FROM Clients GROUP BY income ORDER BY request_count DESC LIMIT 1;
+---------+---------------+
| income  | request_count |
+---------+---------------+
| 10k-20k |         12122 |
+---------+---------------+
1 row in set (0.08 sec)
```

We can also check the request count of other income groups too as follows:

```
mysql> SELECT income,count(*) as request_count FROM Clients GROUP BY income ORDER BY request_count DESC LIMIT 5;
+-----------+---------------+
| income    | request_count |
+-----------+---------------+
| 10k-20k   |         12122 |
| 20k-40k   |         12116 |
| 150k-250k |         12032 |
| 100k-150k |         11986 |
| 40k-60k   |         11956 |
+-----------+---------------+
5 rows in set (0.08 sec)
```

For checking the functionality of the vm you can restart the vm instance-1 which means the web server will be running. Then you can run your http client on instance 4, 5 , or 3 and can run app 2 on instance - 4 for checking the banned countries.

I also started app - 2 just to check if it is running fine with the following web server so I checked the last rows of file requests from banned countries from error_logs table, and the files are as follows:

```
| 4462 | 2023-10-30 07:22:17 | 9784.html      |        403 |
| 4463 | 2023-10-30 07:22:20 | 4164.html      |        403 |
| 4464 | 2023-10-30 07:22:20 | 4164.html      |        403 |
| 4465 | 2023-10-30 07:22:24 | 6871.html      |        403 |
| 4466 | 2023-10-30 07:22:24 | 6871.html      |        403 |
| 4467 | 2023-10-30 07:22:25 | 7332.html      |        403 |
| 4468 | 2023-10-30 07:22:25 | 7332.html      |        403 |
| 4469 | 2023-10-30 07:22:28 | 628.html       |        403 |
| 4470 | 2023-10-30 07:22:28 | 628.html       |        403 |
| 4471 | 2023-10-30 07:22:44 | 1482.html      |        403 |
| 4472 | 2023-10-30 07:22:45 | 1482.html      |        403 |
| 4473 | 2023-10-30 07:23:22 | 3238.html      |        403 |
| 4474 | 2023-10-30 07:23:23 | 3238.html      |        403 |
| 4475 | 2023-10-30 07:23:26 | 5017.html      |        403 |
| 4476 | 2023-10-30 07:23:27 | 4994.html      |        403 |
| 4477 | 2023-10-30 07:23:28 | 5017.html      |        403 |
| 4478 | 2023-10-30 07:23:28 | 4994.html      |        403 |
| 4479 | 2023-10-30 07:23:37 | 5960.html      |        403 |
| 4480 | 2023-10-30 07:23:38 | 5960.html      |        403 |
| 4481 | 2023-10-30 07:23:48 | 5260.html      |        403 |
| 4482 | 2023-10-30 07:23:48 | 5260.html      |        403 |
+------+---------------------+----------------+------------+
4480 rows in set (0.01 sec)
```

Now I started the app-2 to check for the same :

```
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
Received forbidden request from country: Cuba
Error: Access is prohibited. Error 403
Received forbidden request from country: Cuba
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
Received forbidden request from country: Zimbabwe
Error: Access is prohibited. Error 403
Received forbidden request from country: Zimbabwe
Error: Access is prohibited. Error 403
Received forbidden request from country: Libya
Error: Access is prohibited. Error 403
Received forbidden request from country: Libya
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
Received forbidden request from country: Sudan
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
Received forbidden request from country: Sudan
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
Received forbidden request from country: Zimbabwe
Error: Access is prohibited. Error 403
Received forbidden request from country: Zimbabwe
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
Received forbidden request from country: Syria
Error: Access is prohibited. Error 403
```

Also for running the app2-main.py you can use the key.json as previously mentioned in other assignments too.