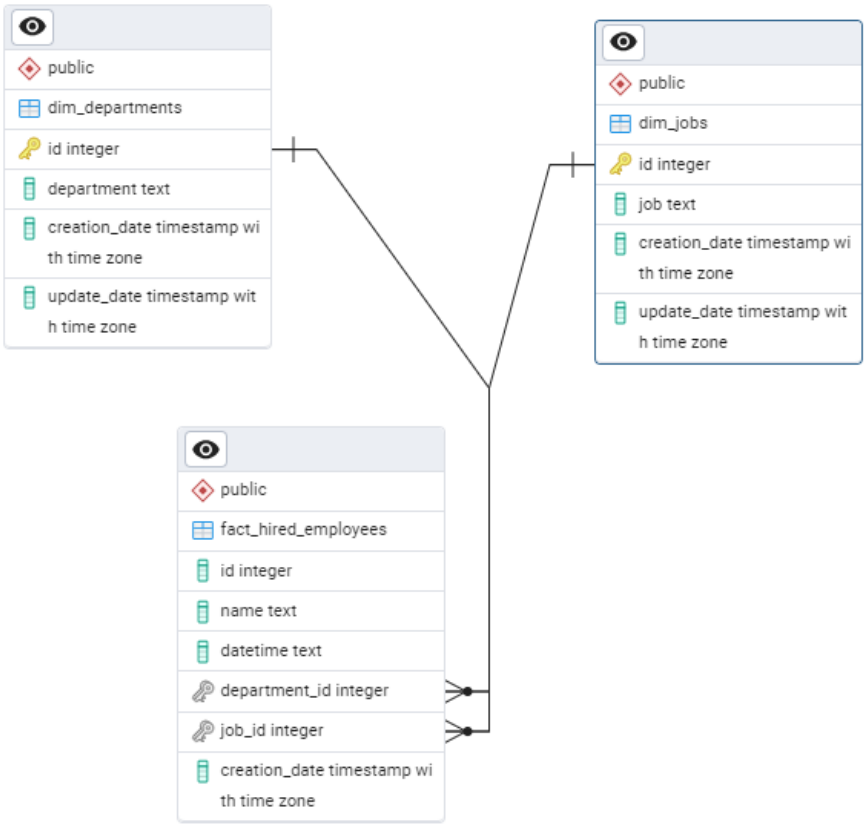
Entity Relation Diagram



Testing

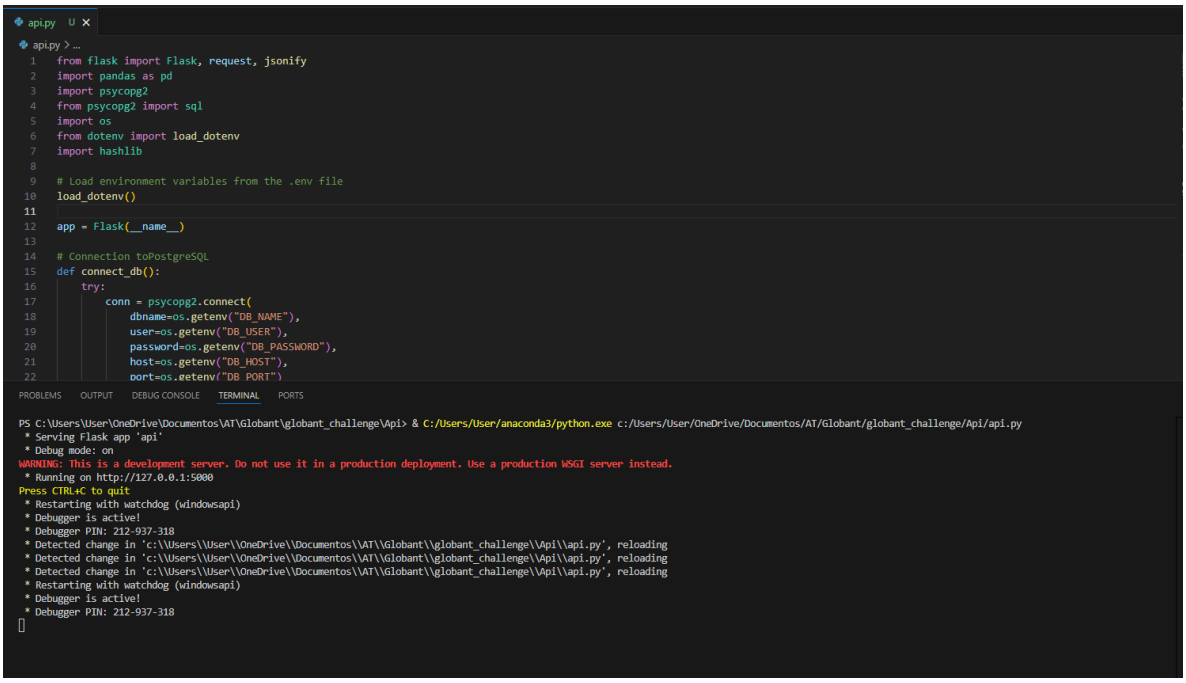
1. API

Test Case 1: Check if the api is running

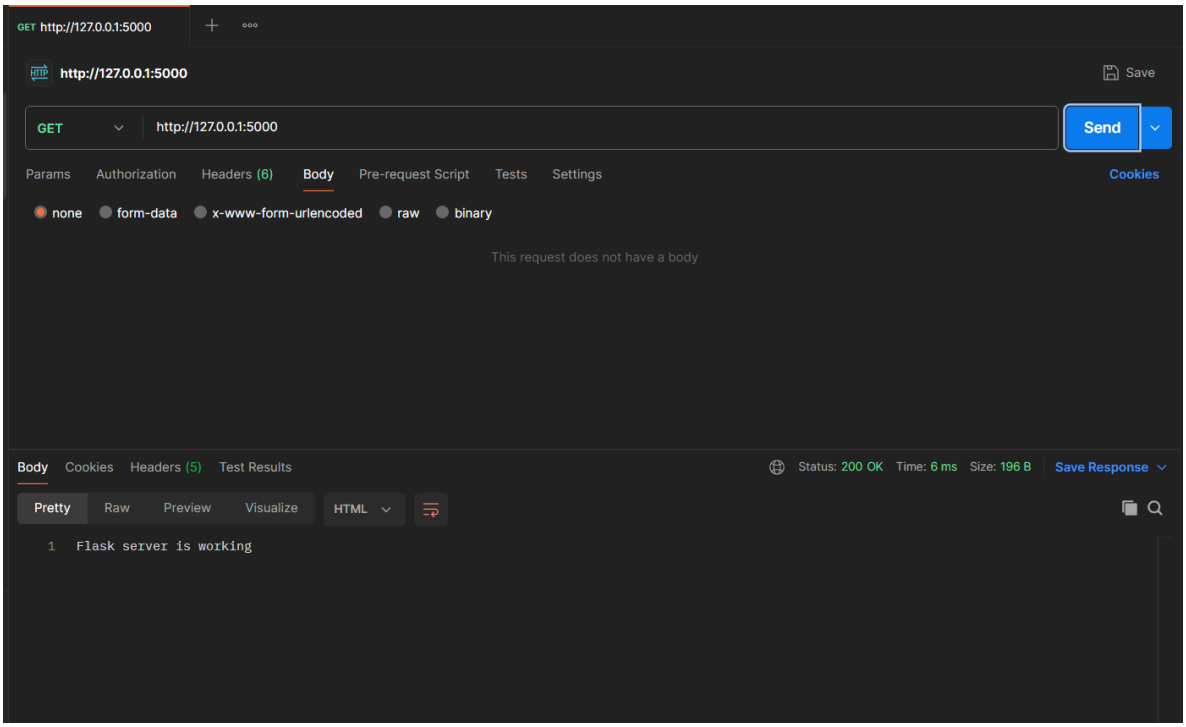
Description: Run the code for the api ('api.py') and check that there are no errors in the execution. Also check that the api is running on http://127.0.0.1:5000 .

Expected results: The api is running on the stablished port, and there are no errors on the terminal due to the execution.

Execution:



Flask server is working



Results: The api is running according to the expected.

Test Case 2: Load a csv file for dim_departments using POST

Description: Now that the api is running, use postman to send a POST with the departments.csv file to load the dim_departments table using the route http://127.0.0.1:5000/upload_csv/dim_departments .

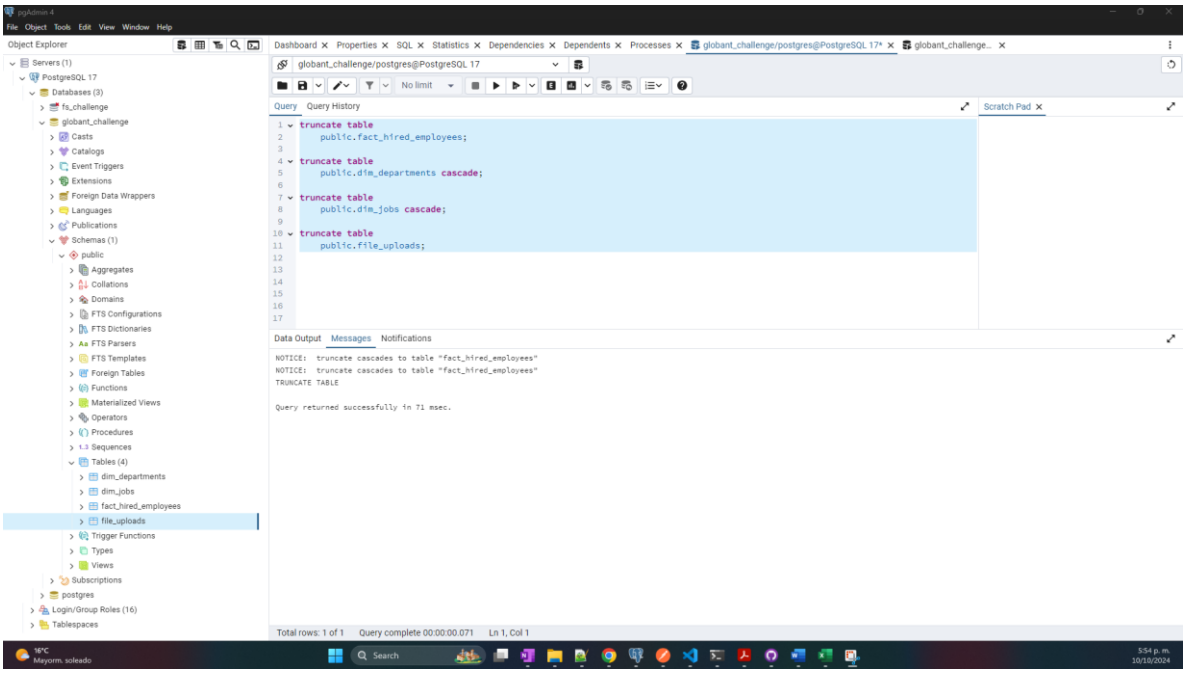
Before to execute the POST ensure to truncate all the tables in the database.

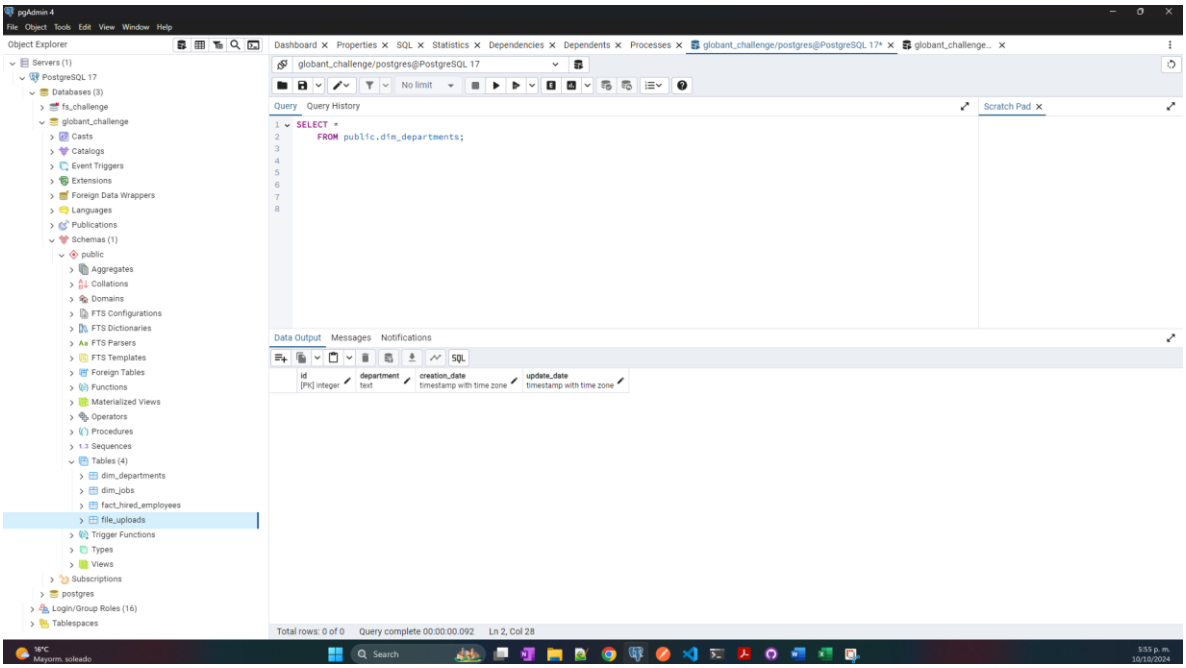
Expected results:

- The message in postman should be 'Data inserted correctly!'.
- Check the *dim_departments* table to confirm that the load is right.
- The number of rows should be 12.
- The table *file_uploads* should have the loaded file and all the columns populated.

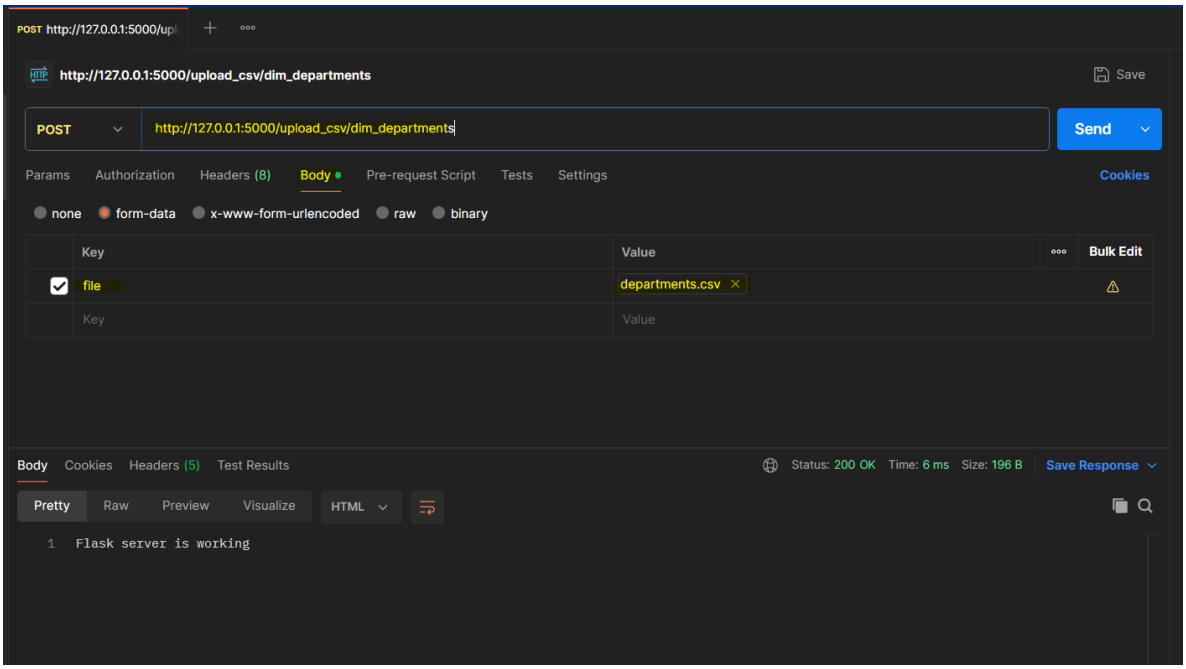
Execution:

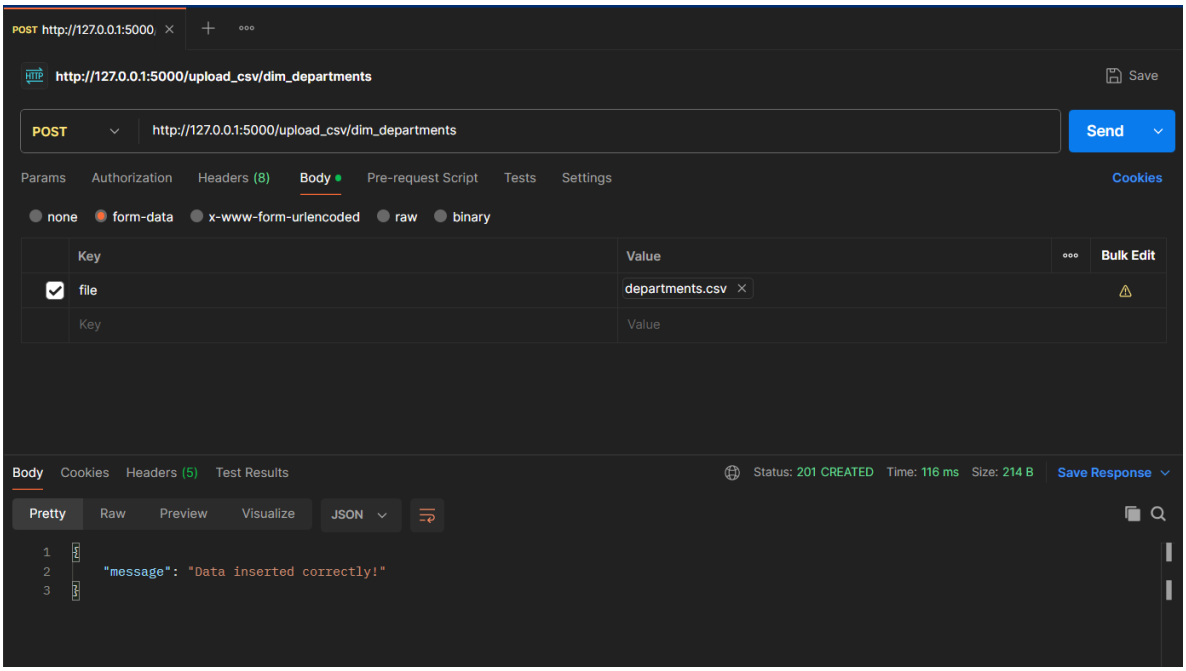
-Truncate tables:



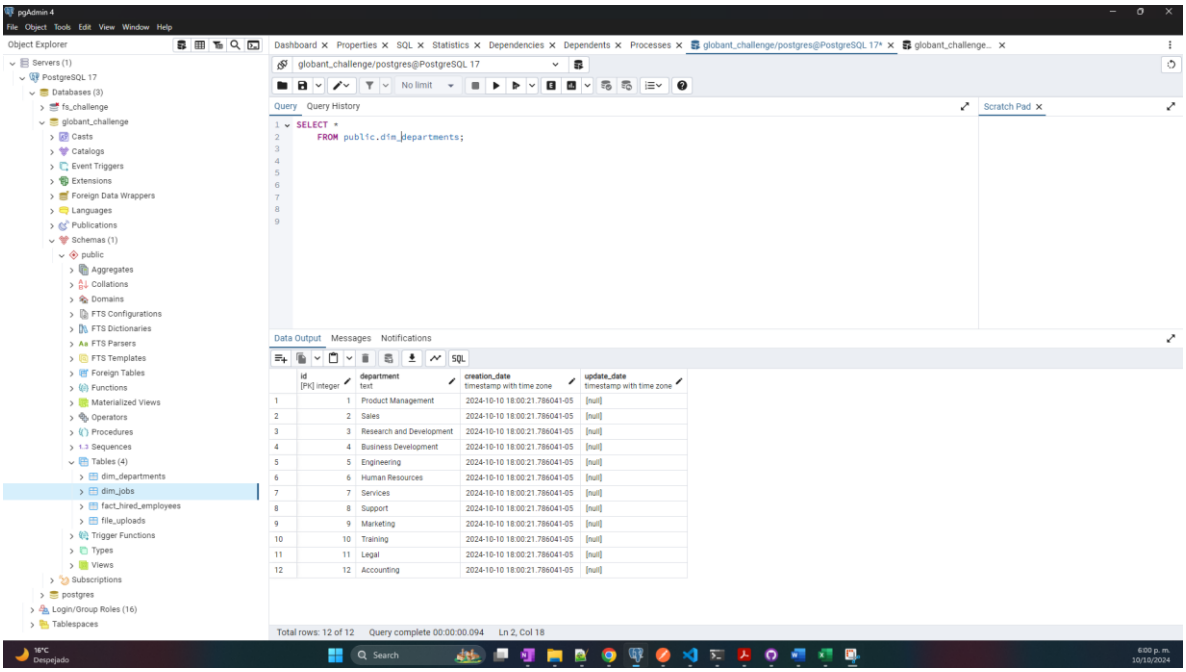


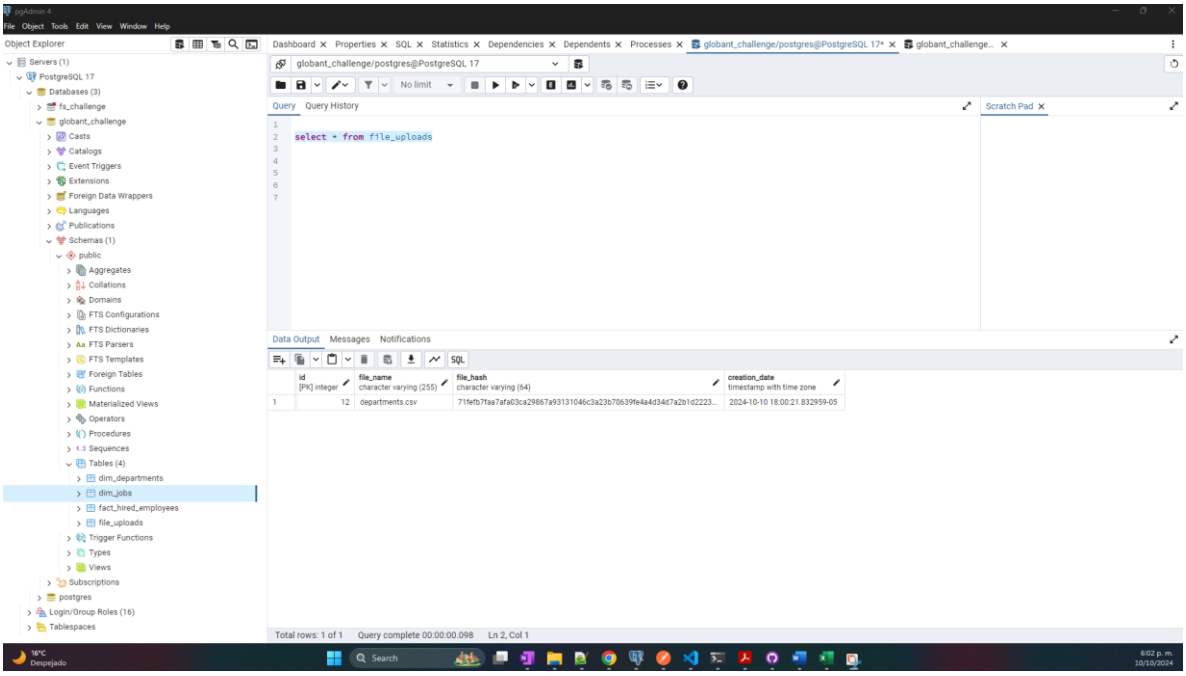
-Now execute the POST in Postman





-Check the tables in the database:





Results: The file was loaded into the dim_departments table and also the row with the hash was inserted into the file_uploads table.

Test Case 3: Load a csv file for dim_jobs using POST

Description: Now that the api is running, use postman to send a POST with the jobs.csv file to load the dim_jobs table using the route http://127.0.0.1:5000/upload_csv/dim_jobs .

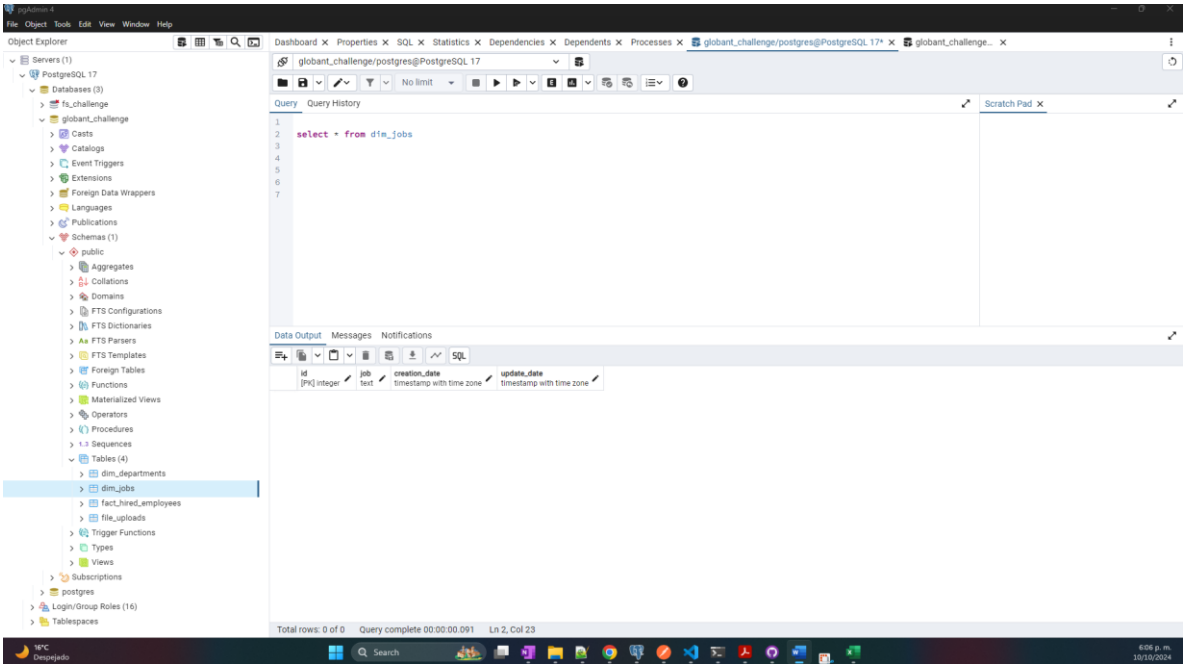
Before to execute the POST check that the dim_jobs table is empty.

Expected results:

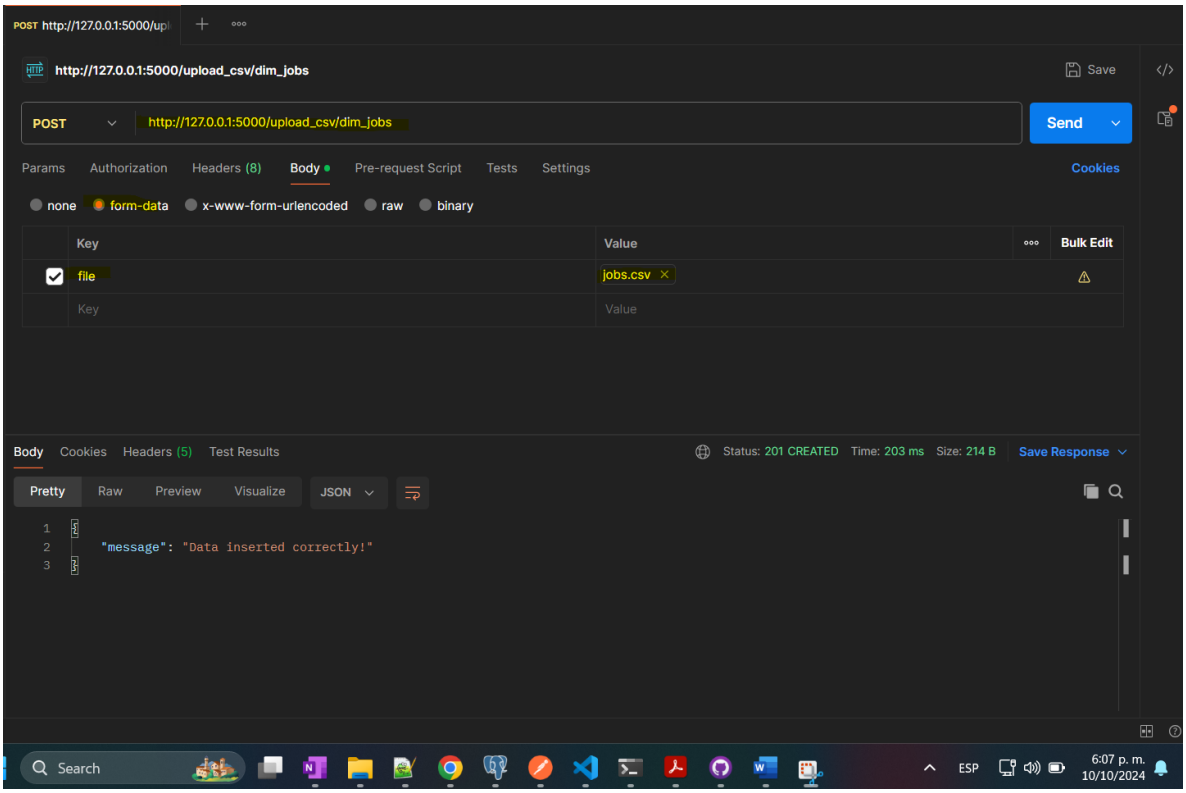
- The message in postman should be ‘Data inserted correctly!’.
- Check the *dim_jobs* table to confirm that the load is right.
- The number of rows should be 183.
- The table *file_uploads* should have the loaded file and all the columns populated.

Execution:

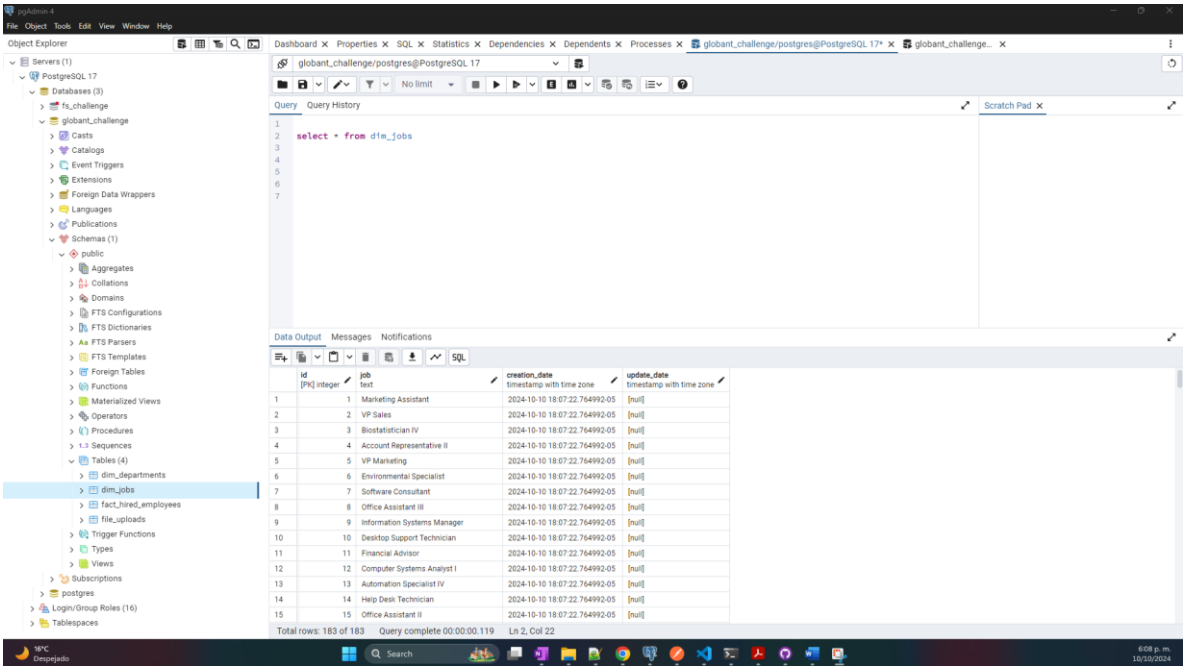
-Check that the dim_jobs table is empty:

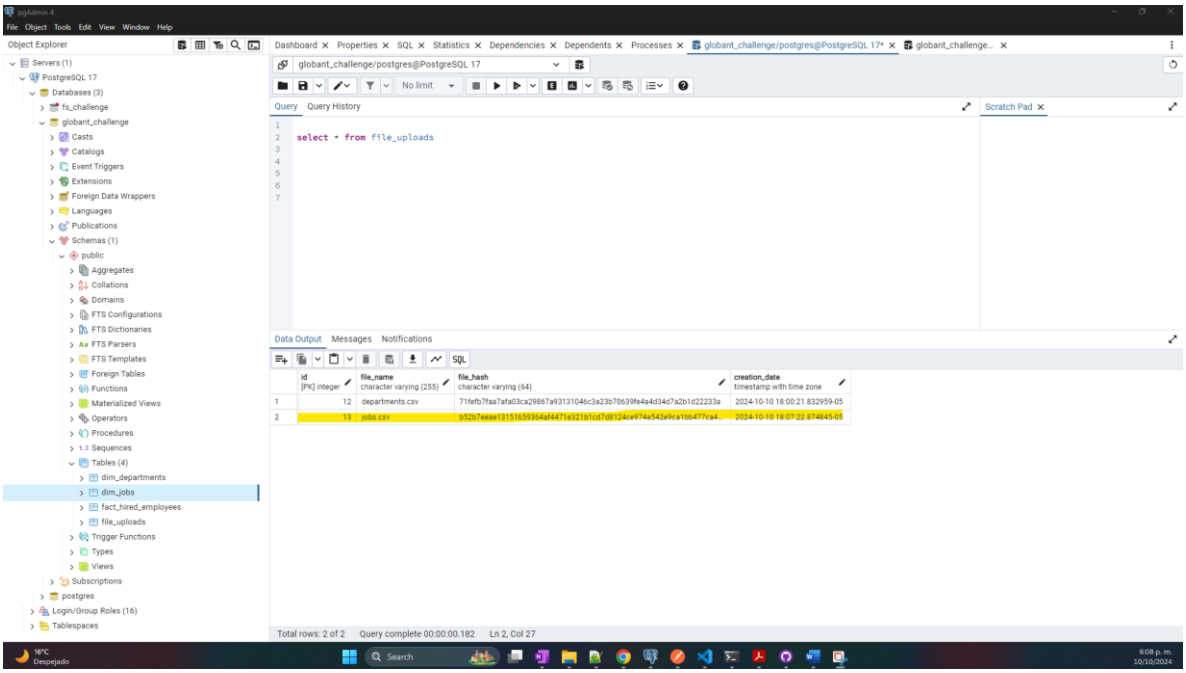


-Now execute the POST in Postman



-Check the tables in the database:





Results: The file was loaded into the *dim_jobs* table with 183 rows and also the row with the hash was inserted into the file_uploads table.

Test Case 4: Load a csv file for fact_hired_employees using POST

Description: Now that the api is running, use postman to send a POST with the hired_employees.csv file to load the *fact_hired_employees* table using the route http://127.0.0.1:5000/upload_csv/fact_hired_employees .

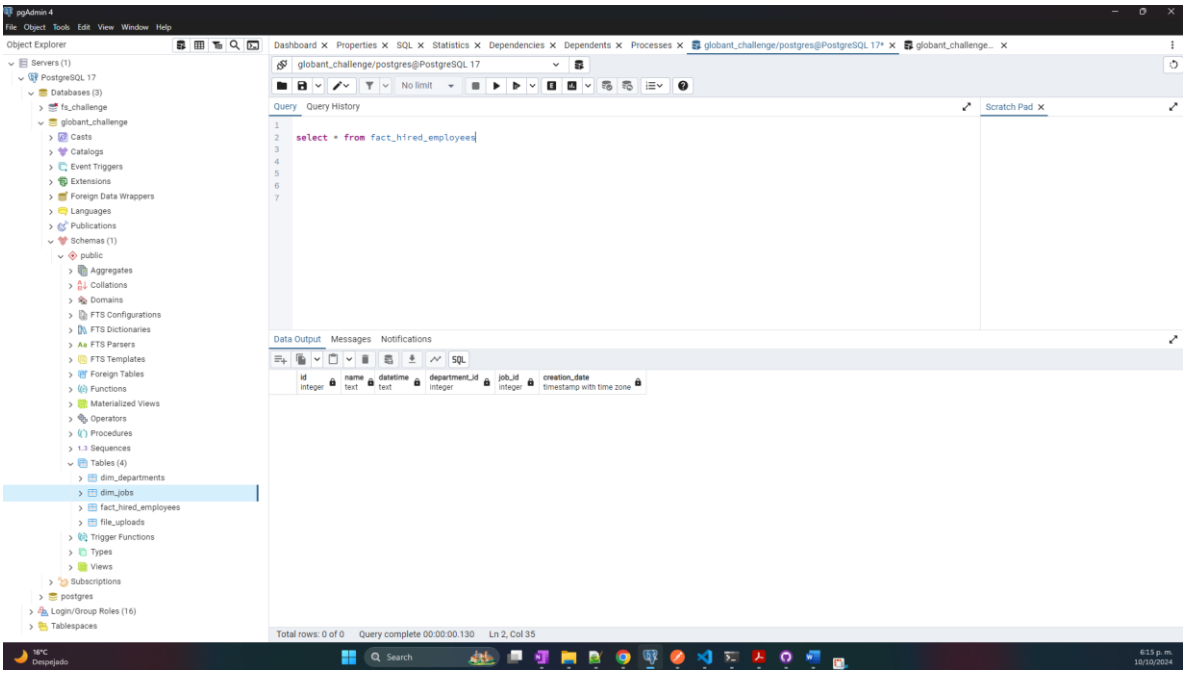
Before to execute the POST check that the *fact_hired_employees* table is empty.

Expected results:

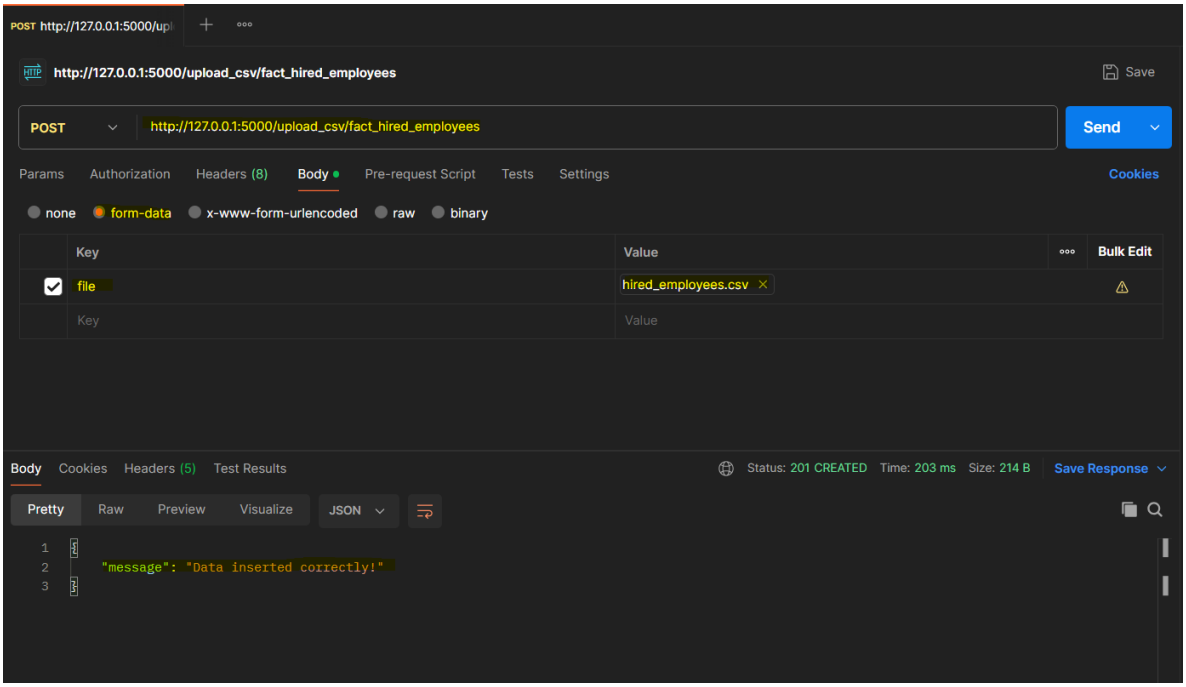
- The message in postman should be 'Data inserted correctly!'.
- Check the *fact_hired_employees* table to confirm that the load is right.
- The number of rows should be 1999.
- The table *file_uploads* should have the loaded file and all the columns populated.

Execution:

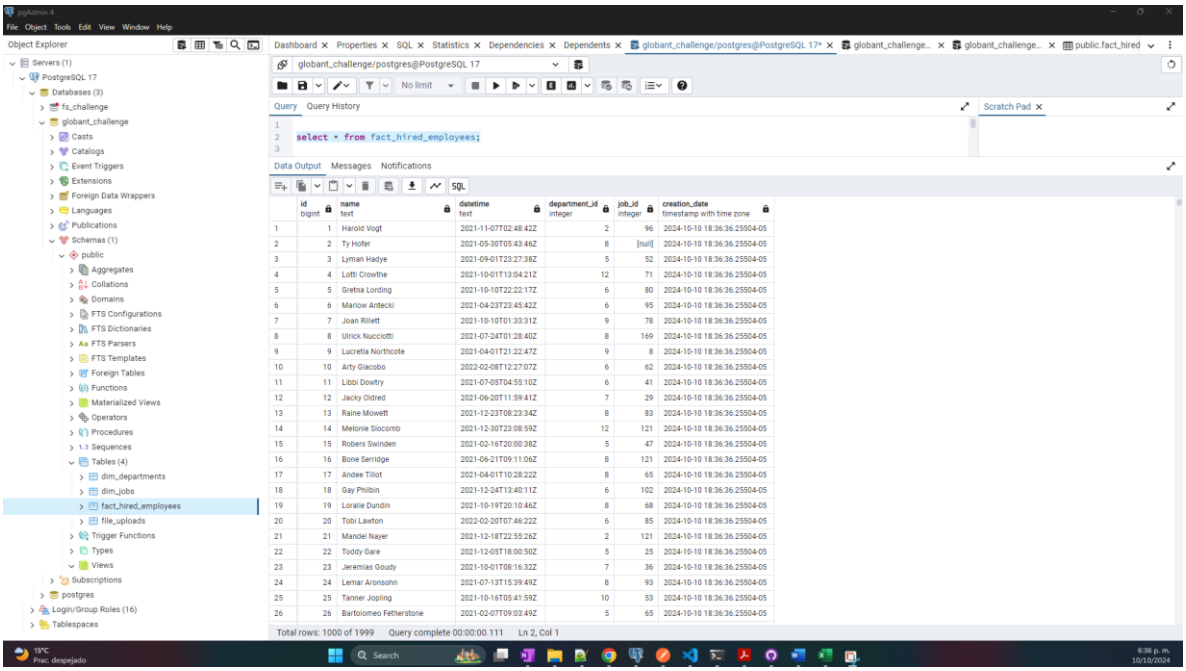
-Check that the *fact_hired_employees* table is empty:

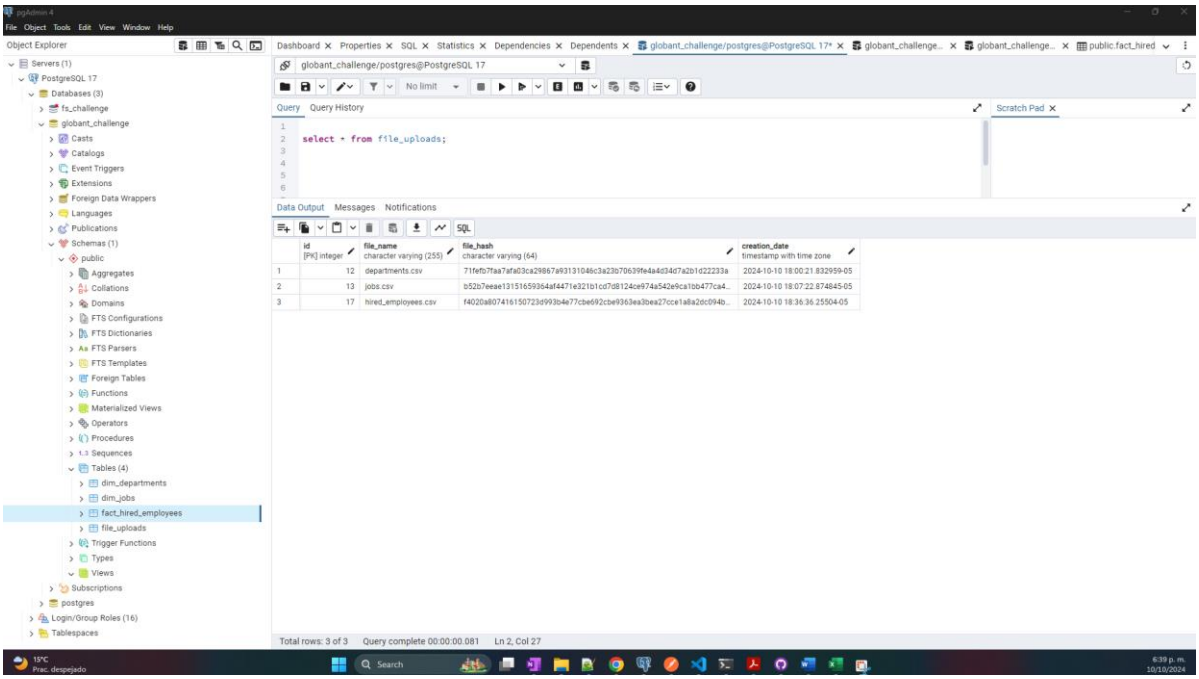


-Now execute the POST in Postman



-Check the tables in the database:





Results: The file was loaded into the *fact_hired_employees* table with 1999 rows and also the row with the hash was inserted into the file_uploads table.

Test Case 5: Try to load a csv file previously loaded using POST

Description: Now that the api is running, use postman to send a POST with the jobs.csv file to try to load the table dim_jobs using the same file loaded previously. Use the route http://127.0.0.1:5000/upload_csv/dim_jobs .

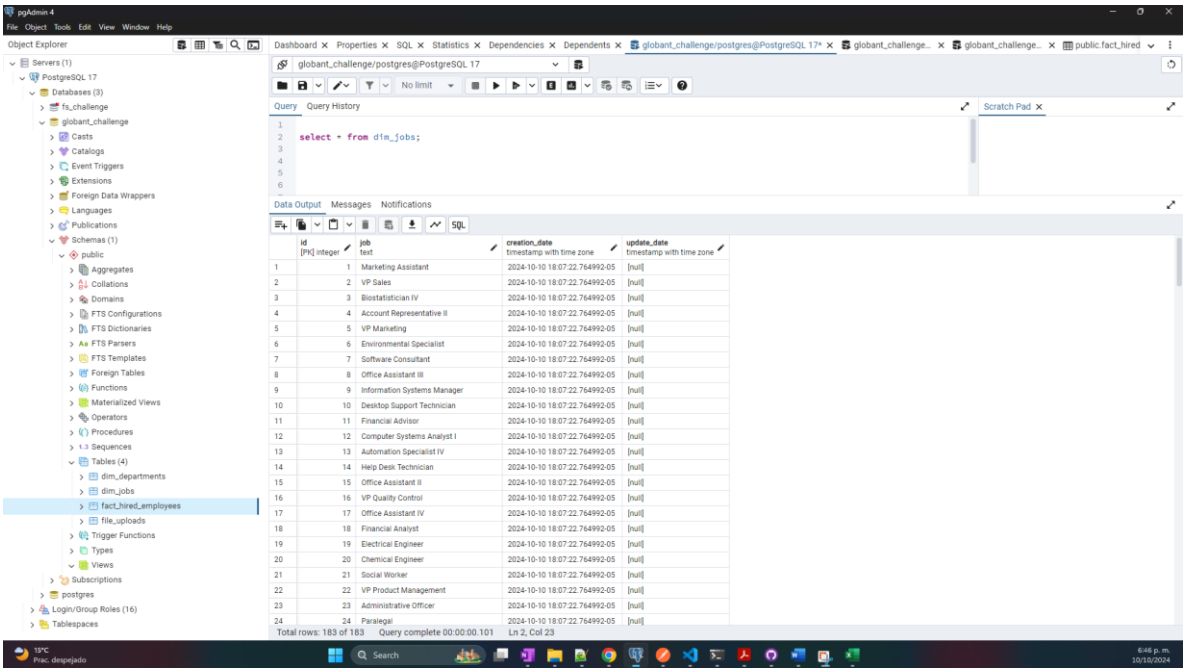
Before to execute the POST check that the *dim_jobs* table is populated and the file *jobs* is in the *file_uploads* table.

Expected results:

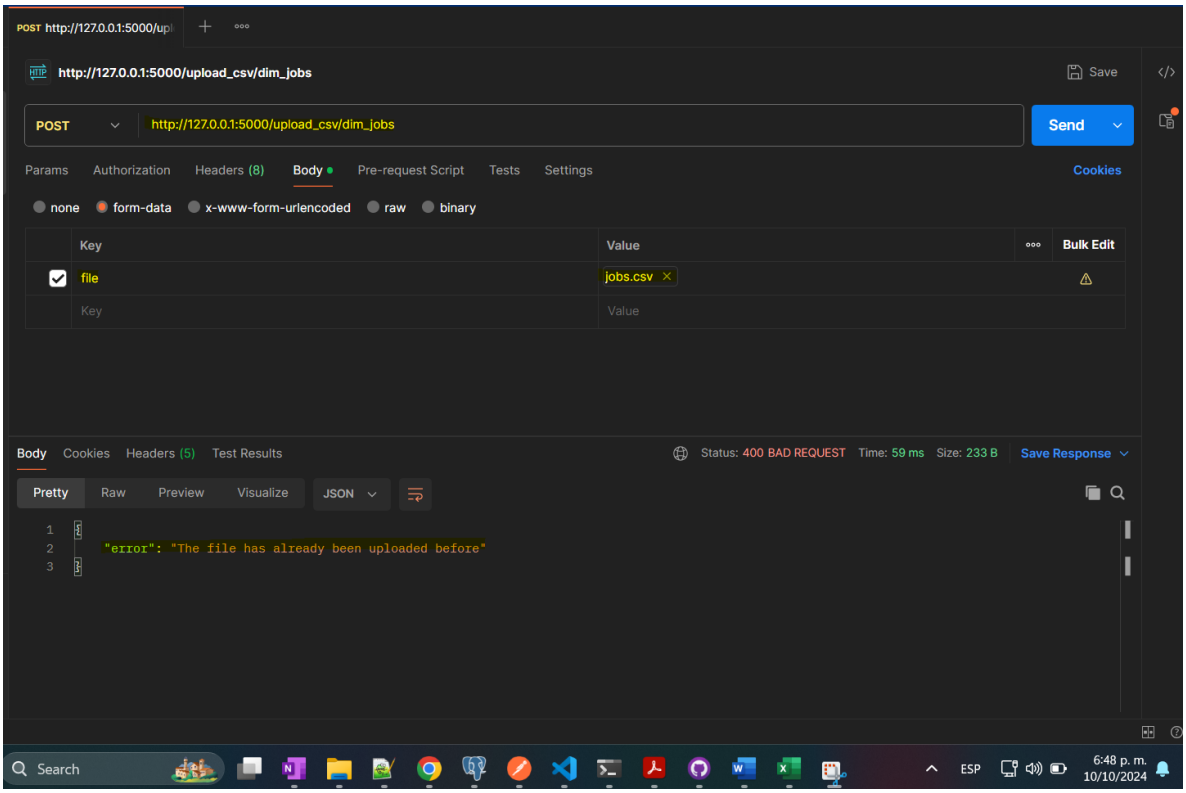
- The message in postman should be ‘The file has already been uploaded before’.
- Check the *dim_jobs* table didn’t load the file again.
- The number of rows should be 183 rows.
- The table *file_uploads* has no changes.

Execution:

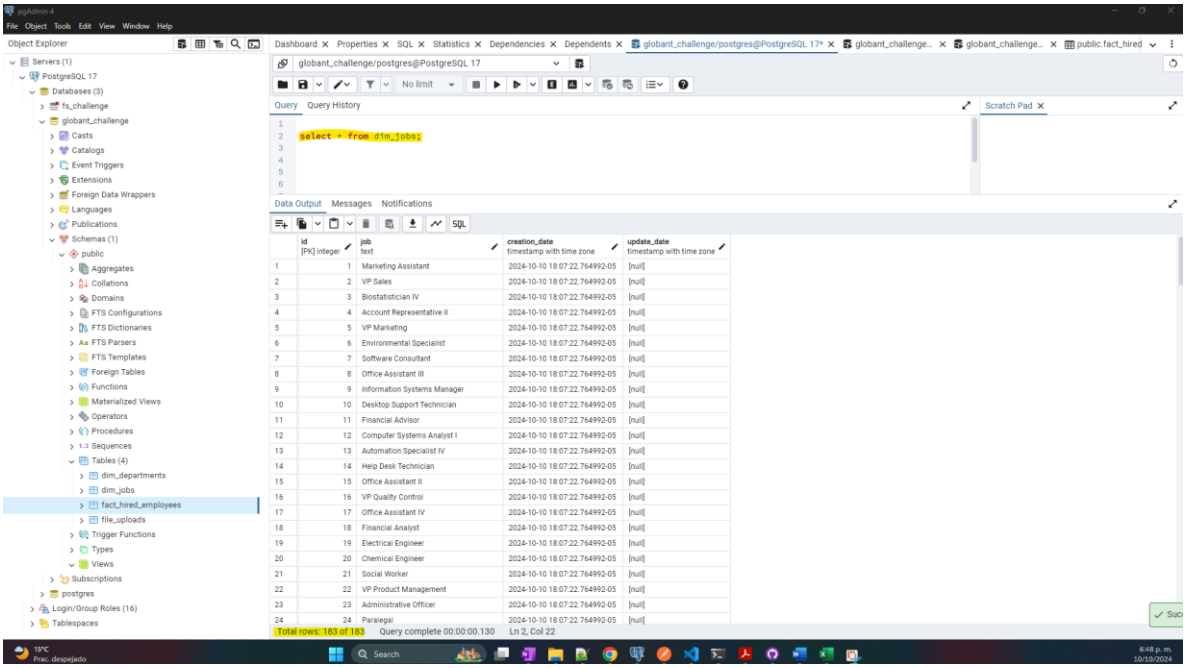
-Check that the *dim_jobs* table has 183 rows:

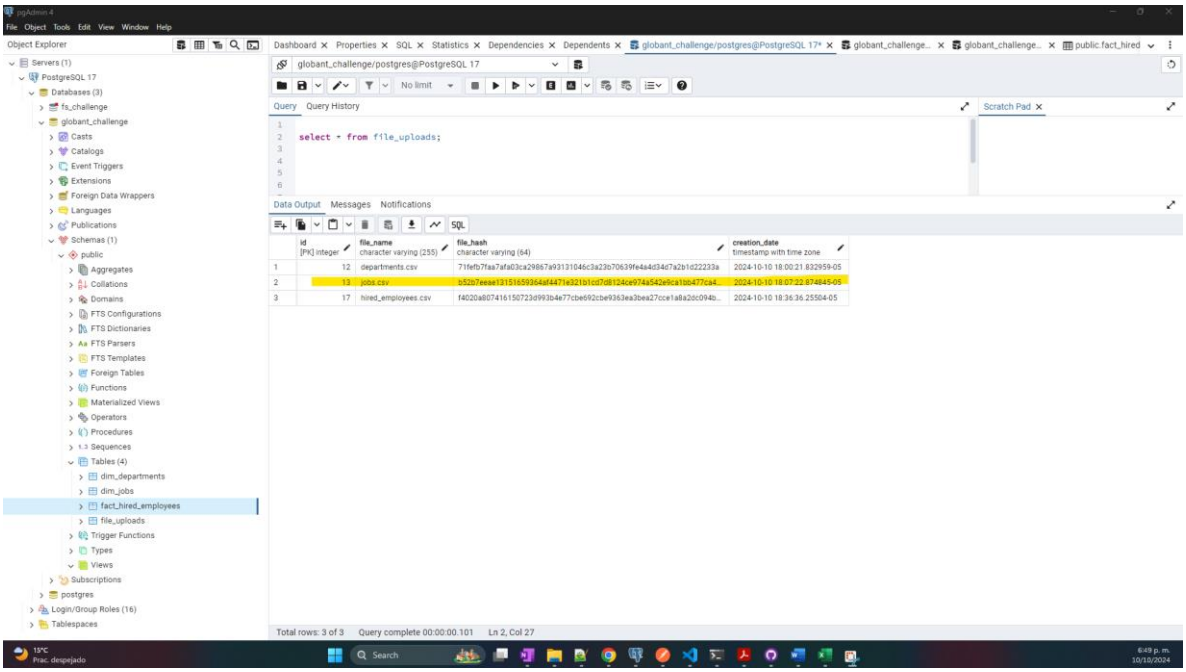


-Now execute the POST in Postman



-Check the tables in the database:





Results: The file wasn't loaded into the *dim_jobs* table. The table has the same 183 rows and no rows were inserted in the *file_uploads* table.

Test Case 6: Try to load a csv file with the incorrect number of columns using POST

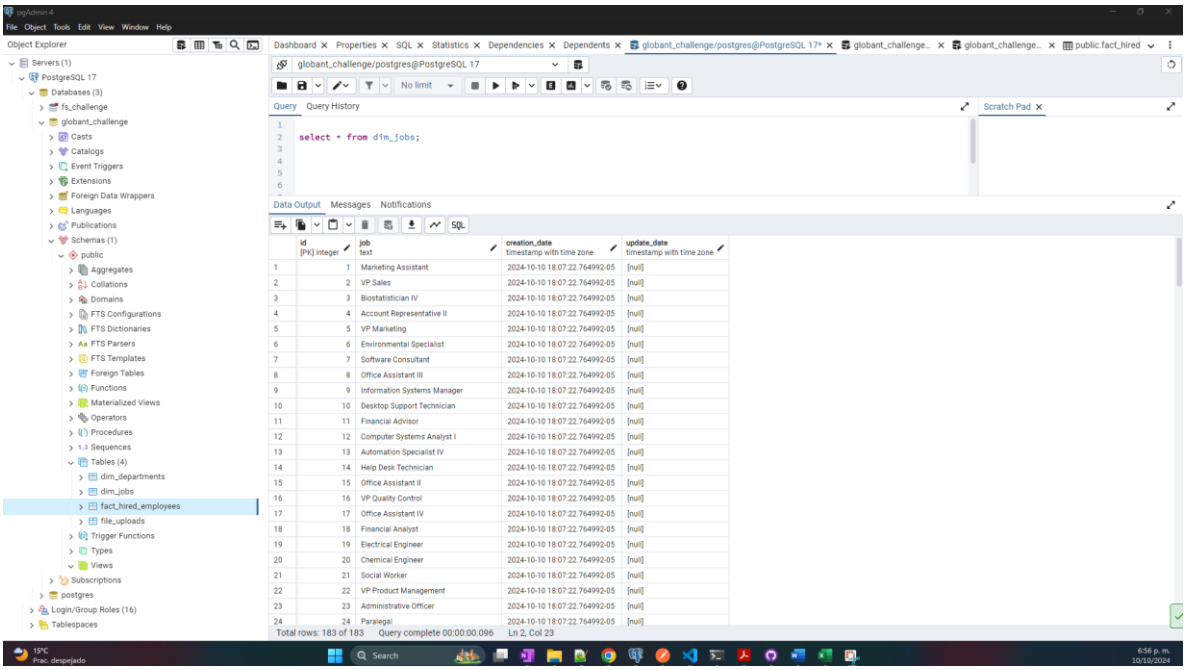
Description: Now that the api is running, use postman to send a POST with the *hired_employees.csv* file to try to load the table *dim_jobs,* since the hired_employees file has more columns than expected by the dim_jobs table, an error should appear. Use the route http://127.0.0.1:5000/upload_csv/dim_jobs .

Expected results:

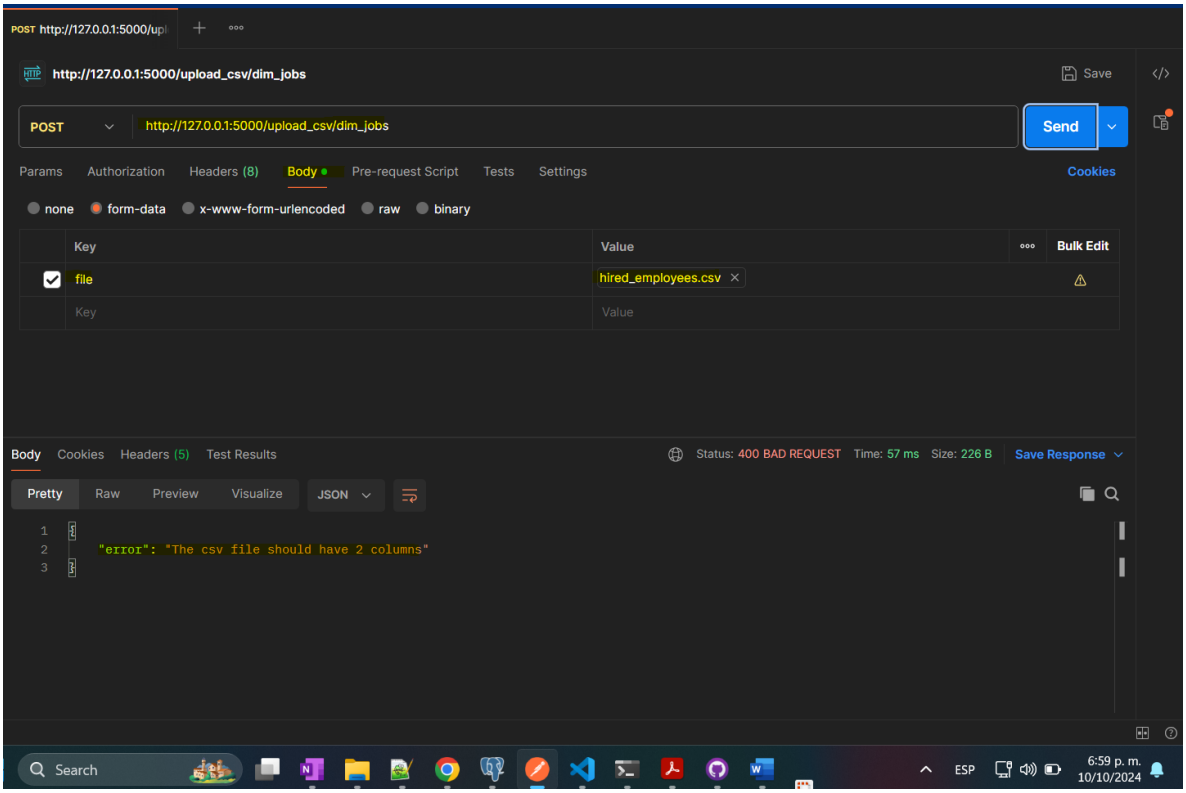
- The message in postman should be 'The csv file should have 2 columns'.
- Check the *dim_jobs* table isn't affected.
- The number of rows should be 183 rows.
- The table *file_uploads* has no changes.

Execution:

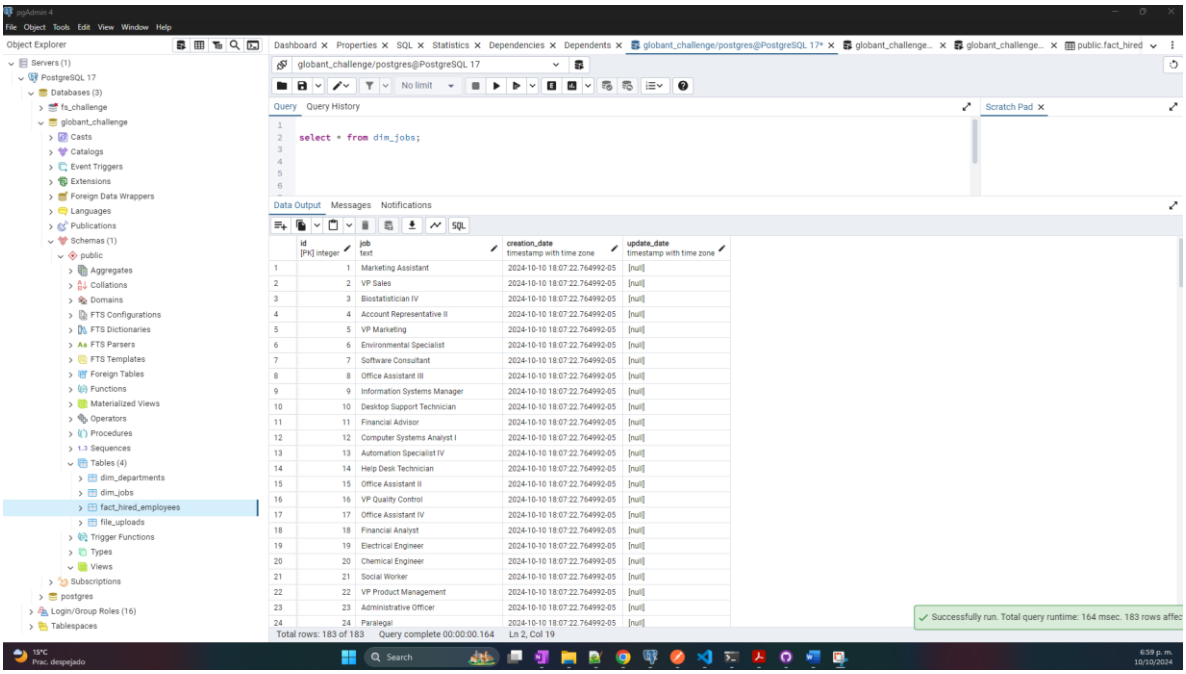
-Check that the *dim_jobs* table has 183 rows:

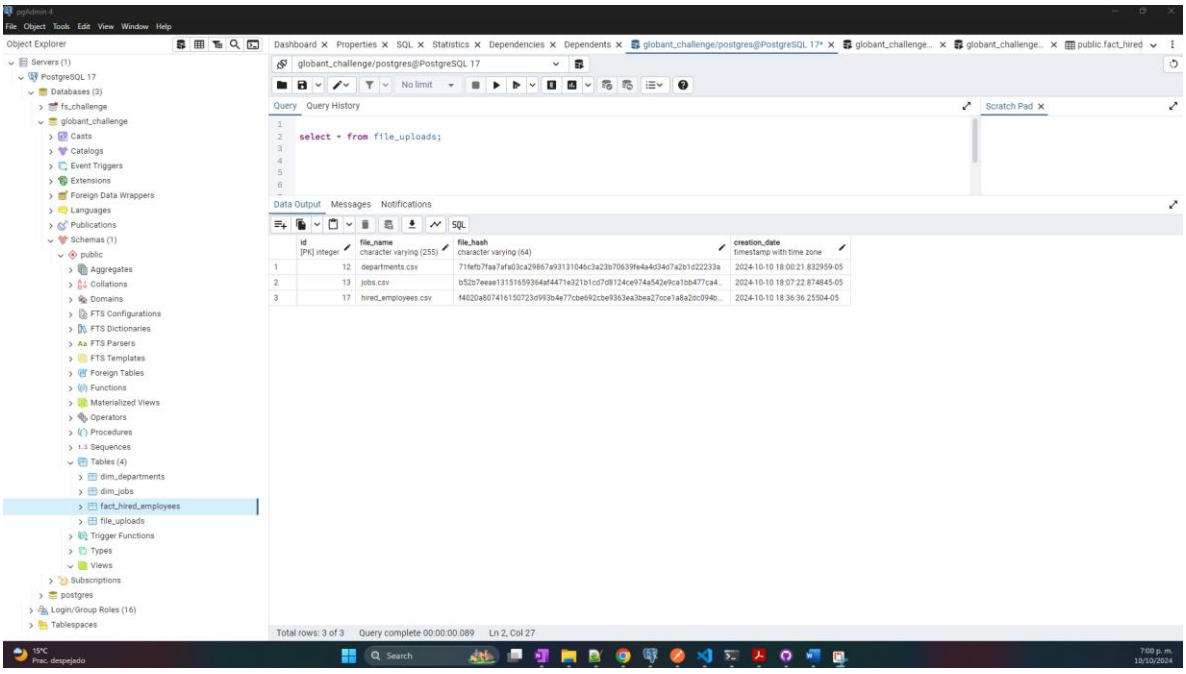


-Now execute the POST in Postman



-Check the tables in the database:





Results: The file wasn't loaded into the *dim_jobs* table. The table has the same 183 rows and no rows were inserted in the *file_uploads* table.

Test Case 7: Try to load an invalid table

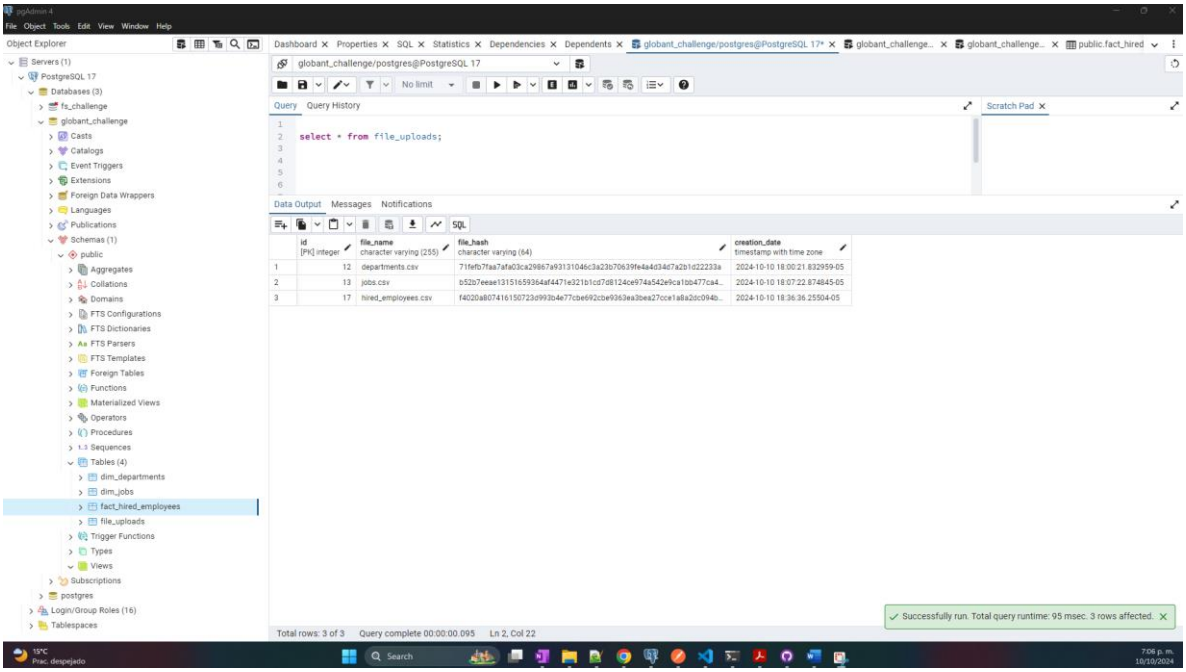
Description: Now that the api is running, use postman to send a POST with the *hired_employees.csv* file to try to load the *file_uploads* table, since the *file_uploads* isn't a valid table to be loaded using csv files, an error should appear. Use the route http://127.0.0.1:5000/upload_csv/file_uploads .

Expected results:

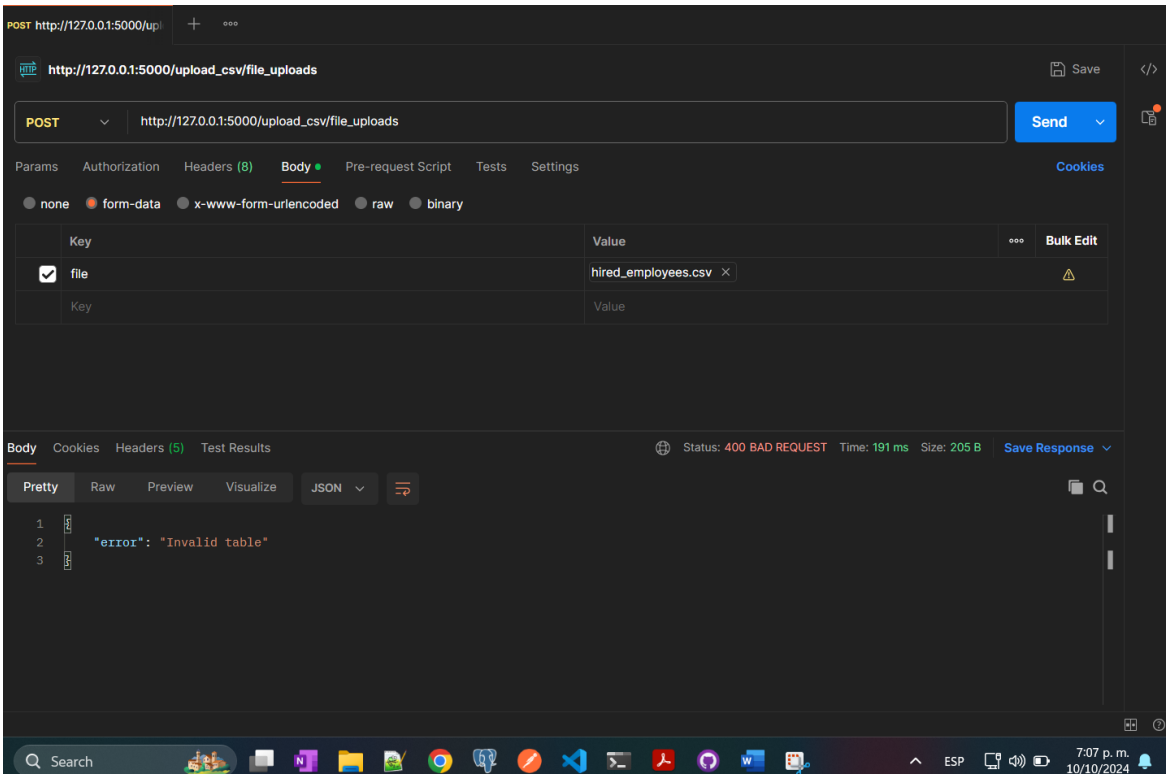
- The message in postman should be ‘Invalid table’.
- The table *file_uploads* has no changes.

Execution:

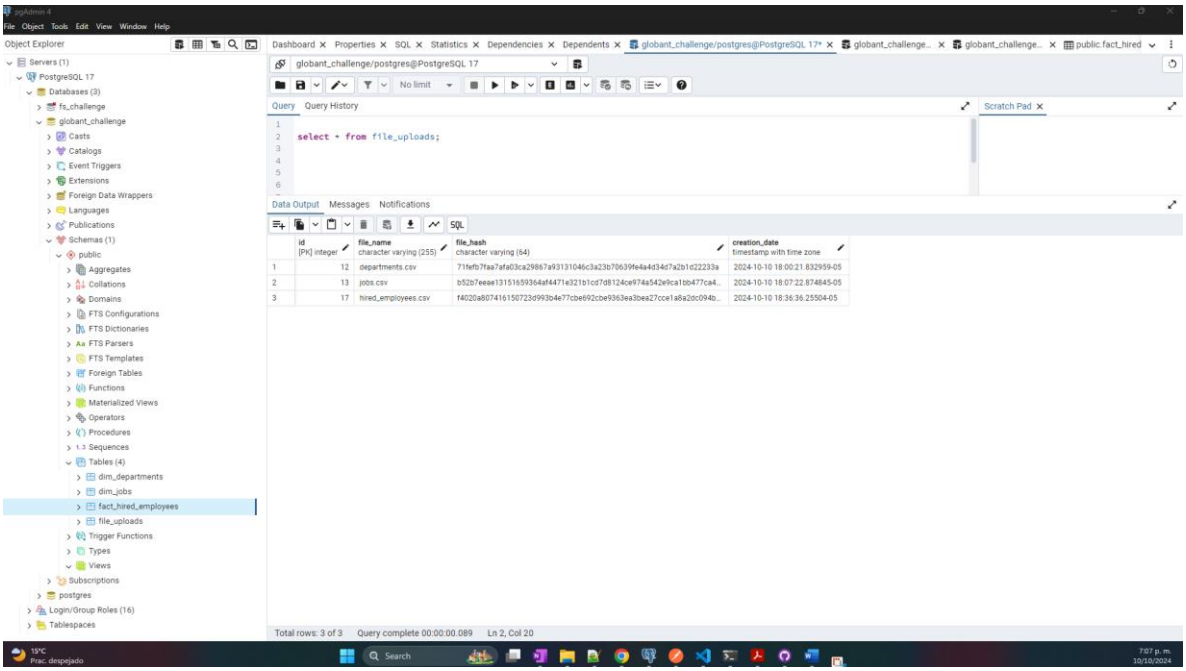
-Check the *file_uploads* table:



-Now execute the POST in Postman



-Check the table in the database:



Results: The file wasn't loaded into the *file_uploads* table. The table has the same 3 rows.

Test Case 8: Try to load a file with more than 1000 rows

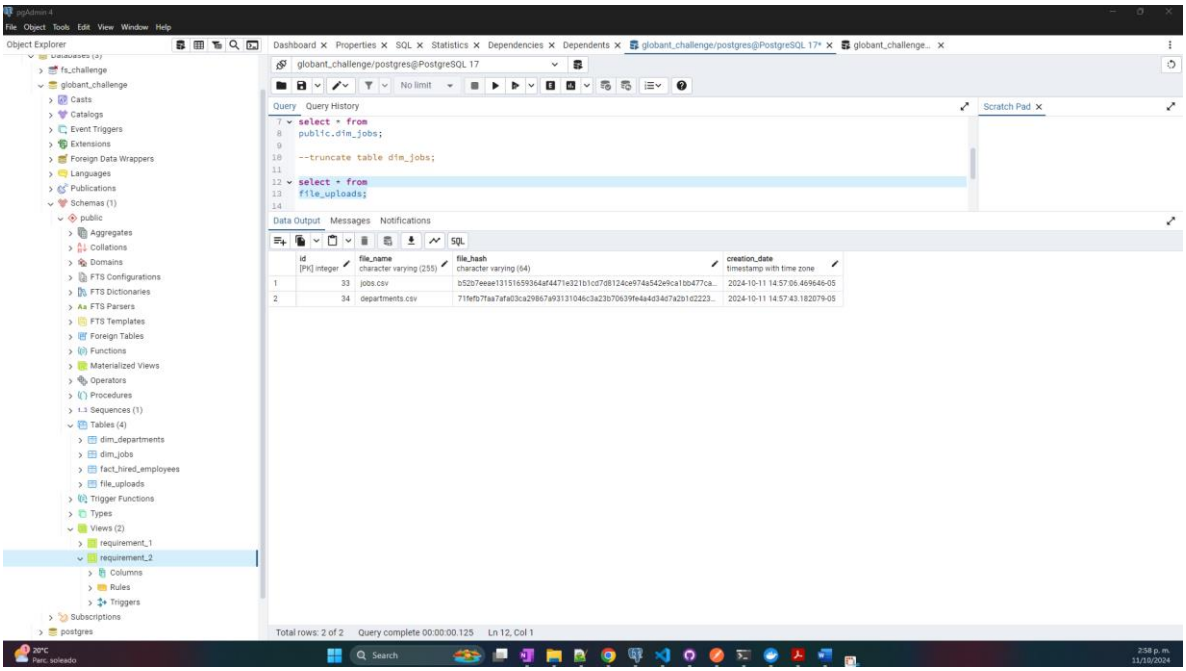
Description: Now that the api is running, use postman to send a POST with the *hired_employees.csv* file to try to load the *fact_hired_employees* table*,* since the csv file contains more than 1000 rows, an error should appear. Use the route http://127.0.0.1:5000/upload_csv/fact_hired_employees .

Expected results:

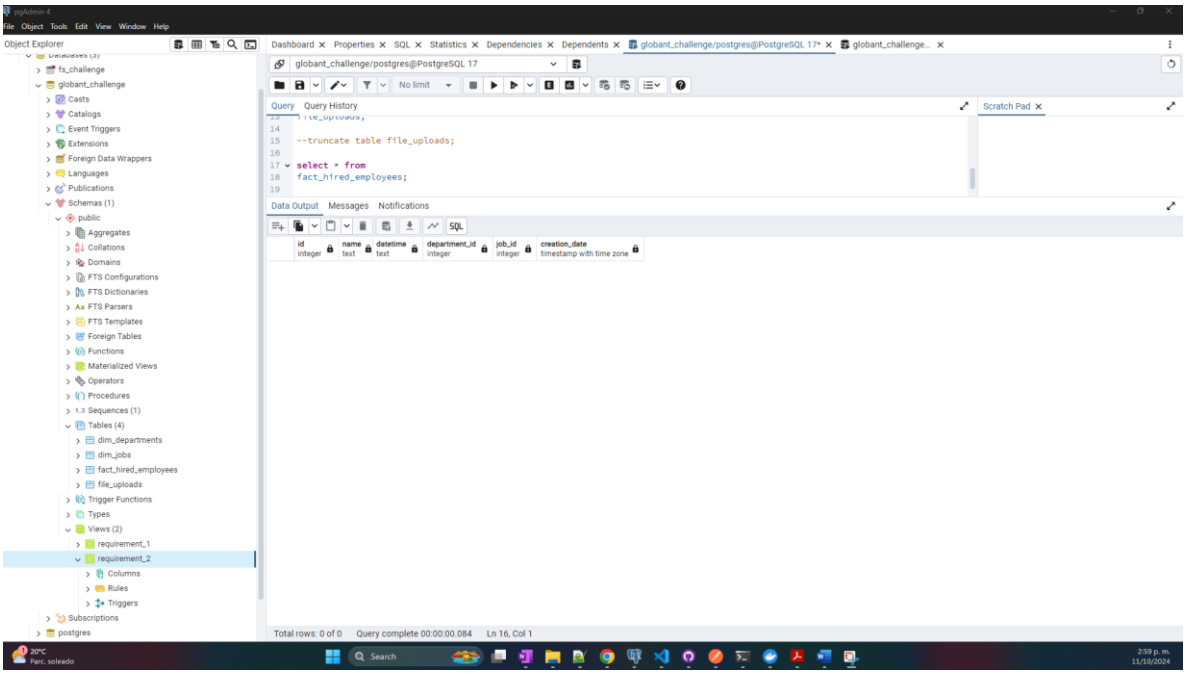
- The message in postman should be 'Batch limit exceeded. You can only insert up to 1000 rows at a time'.
- The table *fact_hired_employees* has no changes.
- The table file_uploads shouldn't have the new row with the file.

Execution:

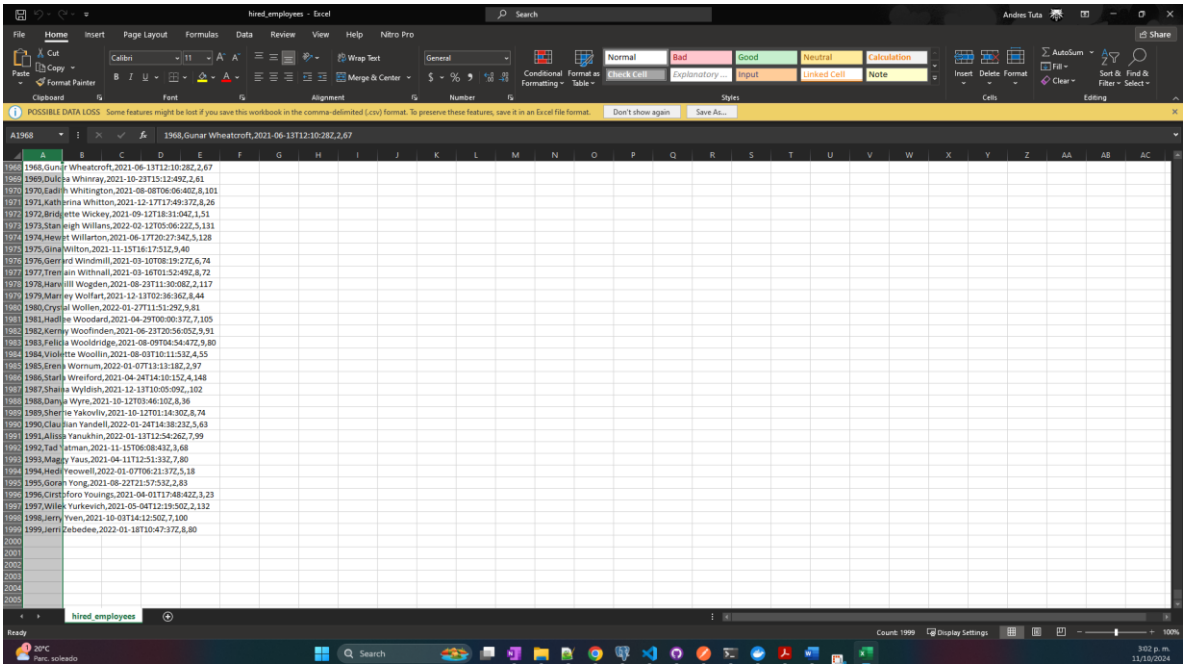
-Check the *file_uploads* table:



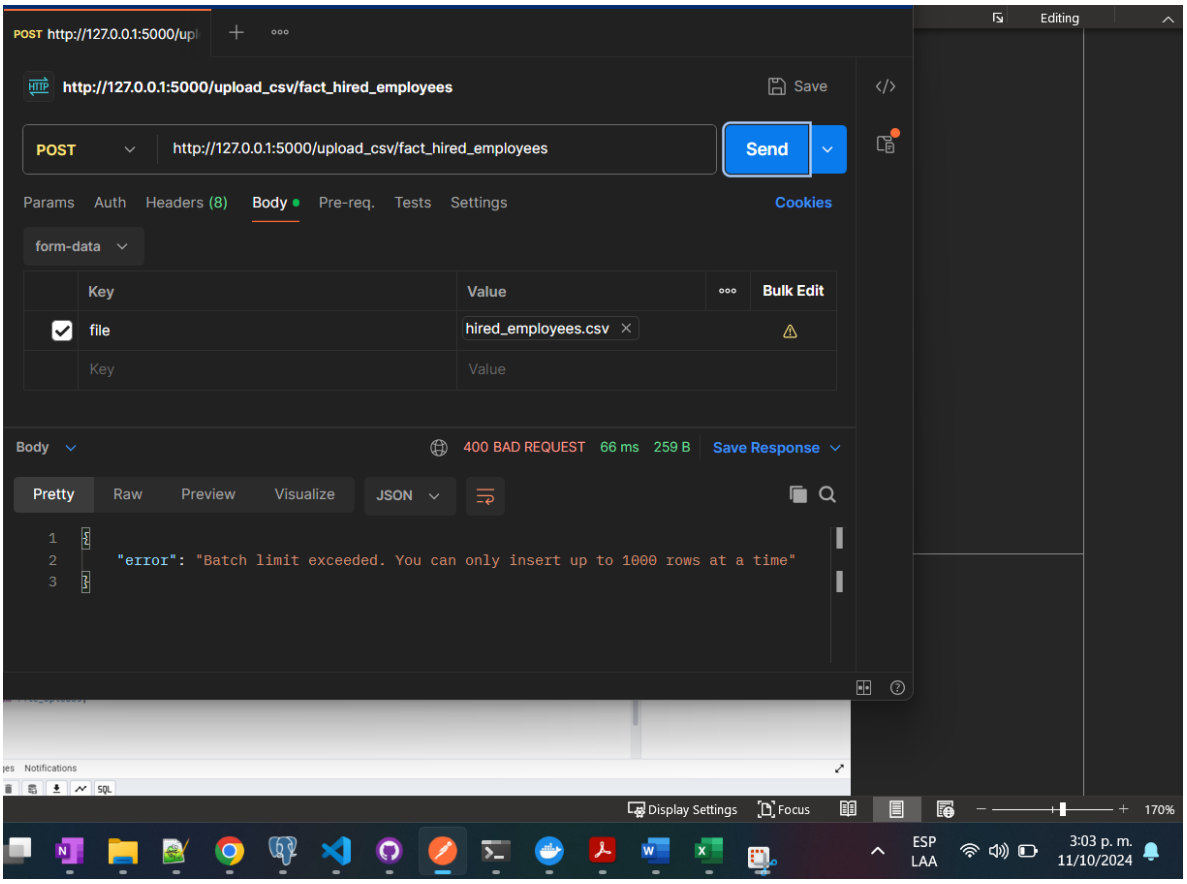
-Check the fact_hired_employees table:



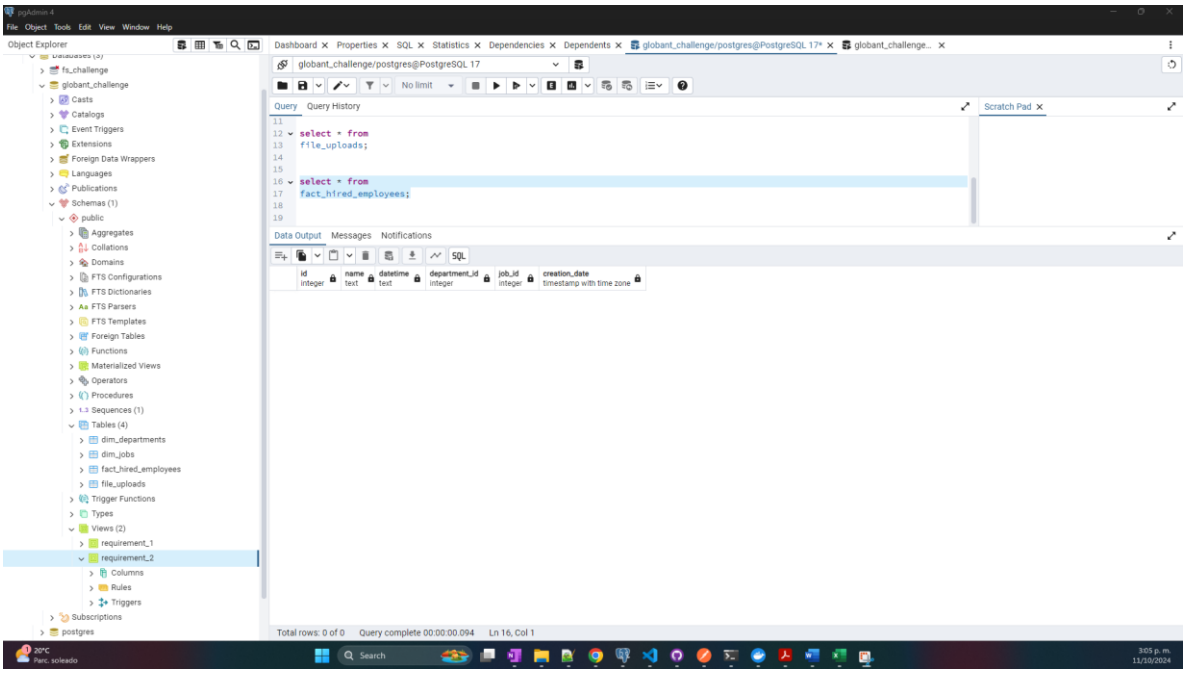
-Check the file to be loaded, the file has more than 1000 rows:

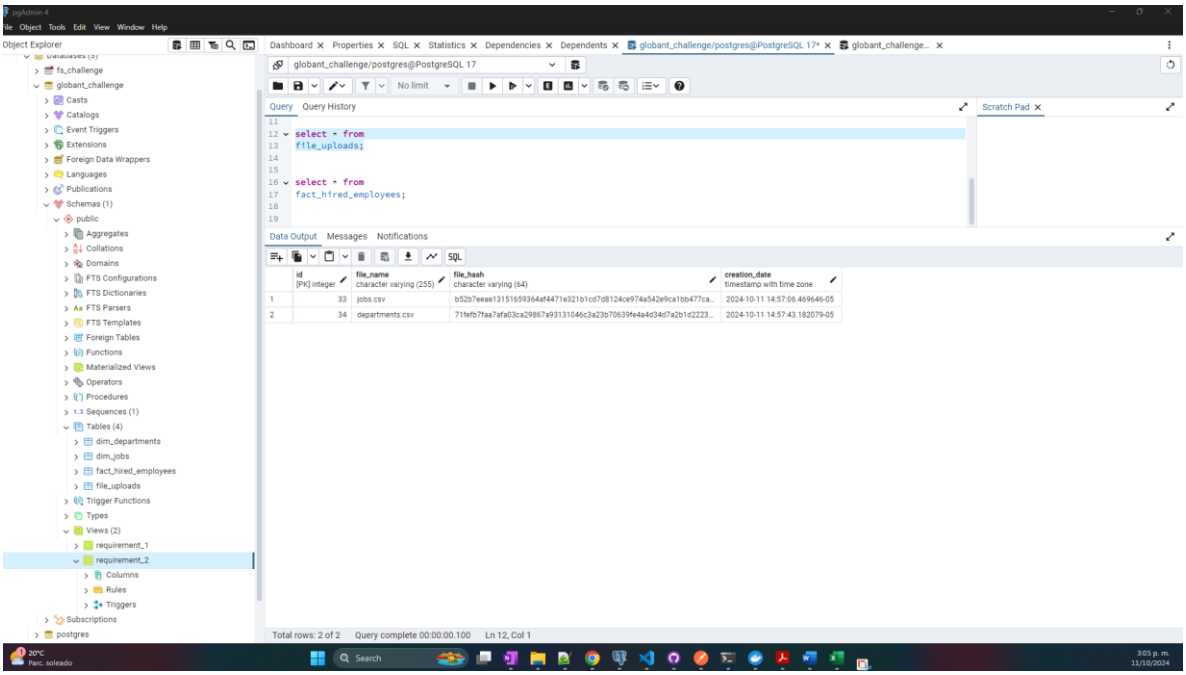


-Now execute the POST in Postman



-Check the tables in the database:





Results: The file wasn't loaded into the *fact_hired_employees* table. And the api returns an error.

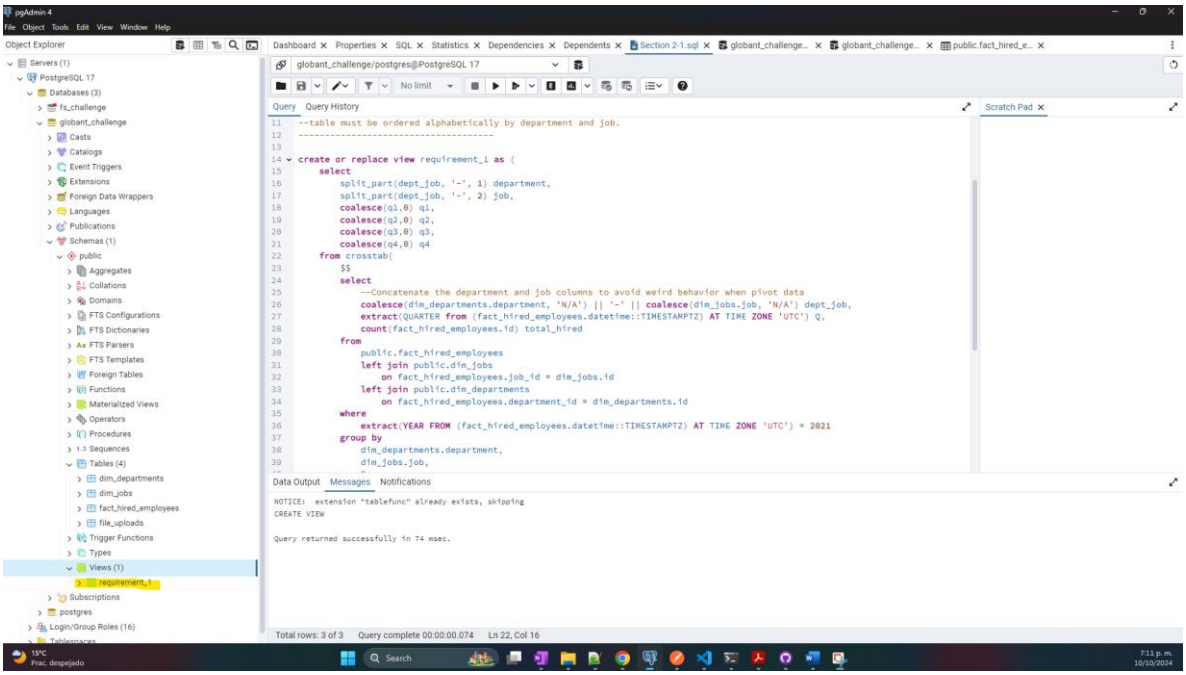
2. SQL

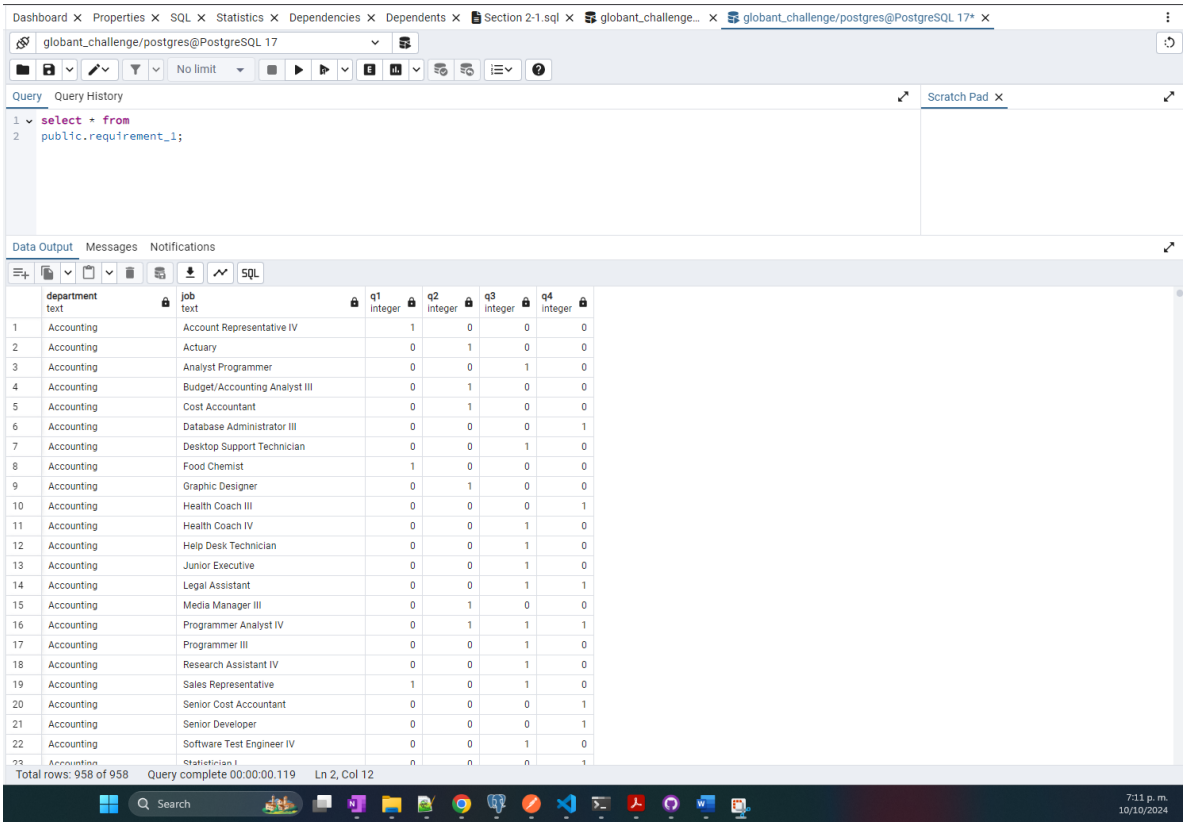
- Number of employees hired for each job and department in 2021 divided by quarter. The table must be ordered alphabetically by department and job.

Output example:

department		job	Q1	Q2	Q3	Q4
Staff		Recruiter	3	0	7	11
Staff		Manager	2	1	0	2
Supply Chain		Manager	0	1	3	0

The query used creates a view with the resulting data:





	department text	job text	q1 integer	q2 integer	q3 integer	q4 integer
1	Accounting	Account Representative IV	1	0	0	0
2	Accounting	Actuary	0	1	0	0
3	Accounting	Analyst Programmer	0	0	1	0
4	Accounting	Budget/Accounting Analyst III	0	1	0	0
5	Accounting	Cost Accountant	0	1	0	0
6	Accounting	Database Administrator III	0	0	0	1
7	Accounting	Desktop Support Technician	0	0	1	0
8	Accounting	Food Chemist	1	0	0	0
9	Accounting	Graphic Designer	0	1	0	0
10	Accounting	Health Coach III	0	0	0	1
11	Accounting	Health Coach IV	0	0	1	0
12	Accounting	Help Desk Technician	0	0	1	0
13	Accounting	Junior Executive	0	0	1	0
14	Accounting	Legal Assistant	0	0	1	1
15	Accounting	Media Manager III	0	1	0	0
16	Accounting	Programmer Analyst IV	0	1	1	1
17	Accounting	Programmer III	0	0	1	0
18	Accounting	Research Assistant IV	0	0	1	0
19	Accounting	Sales Representative	1	0	1	0
20	Accounting	Senior Cost Accountant	0	0	0	1
21	Accounting	Senior Developer	0	0	0	1
22	Accounting	Software Test Engineer IV	0	0	1	0
23	Accounting	Statistician I	0	0	0	1

You can see the full results on the file /SQL/requirement_1_results.csv .

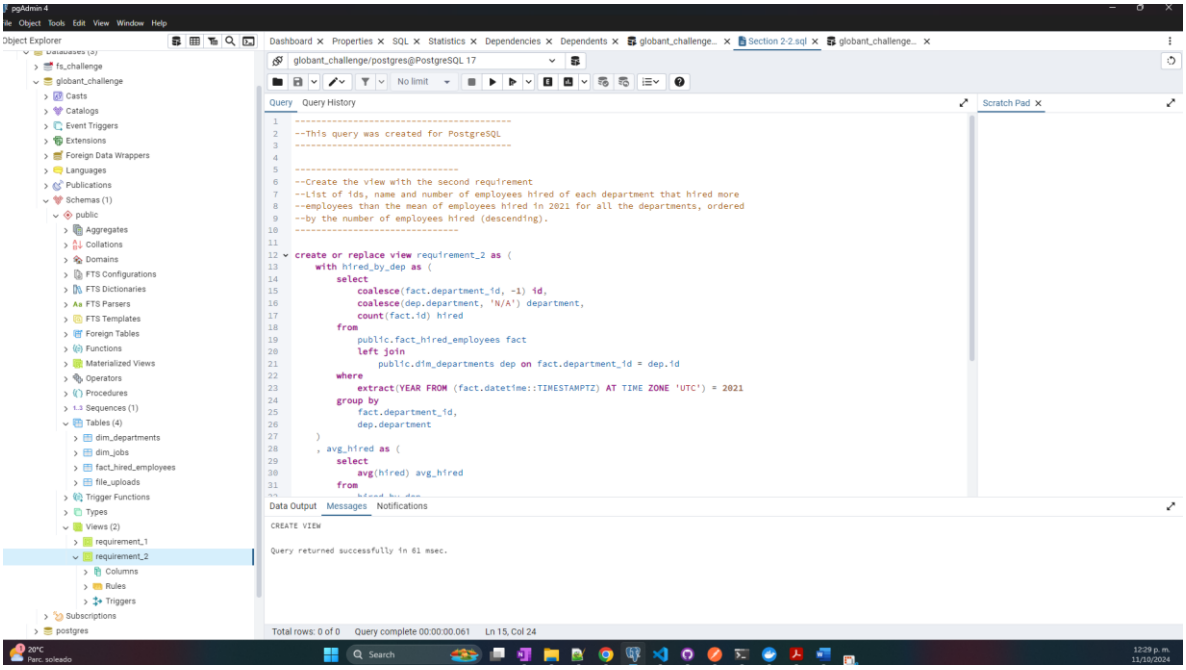
- List of ids, name and number of employees hired of each department that hired more

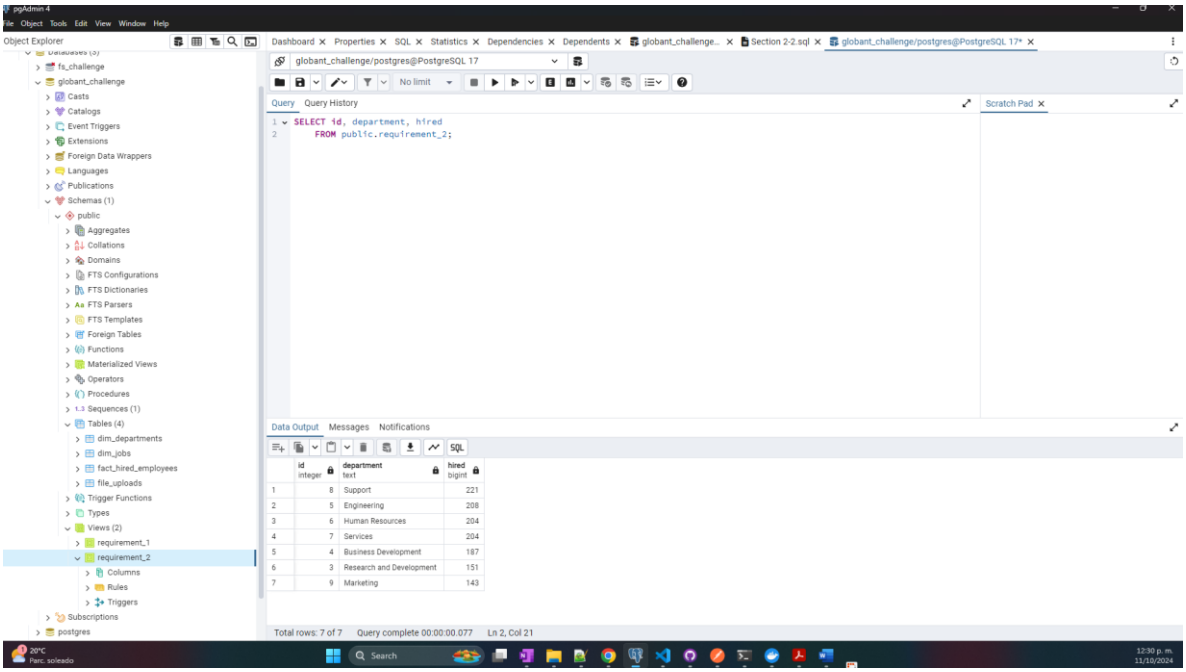
employees than the mean of employees hired in 2021 for all the departments, ordered by the number of employees hired (descending).

Output example:

id	department	hired
7	Staff	45
9	Supply Chain	12

The query used, creates a view with the resulting data:





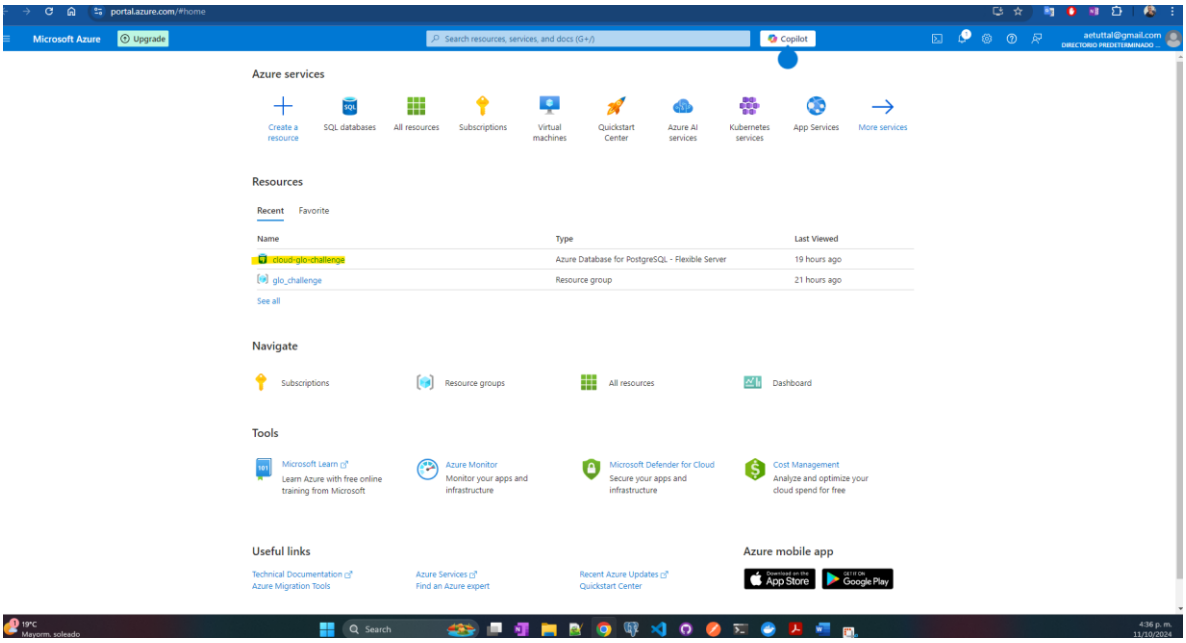
You can see the full results on the file /SQL/requirement_2_results.csv .

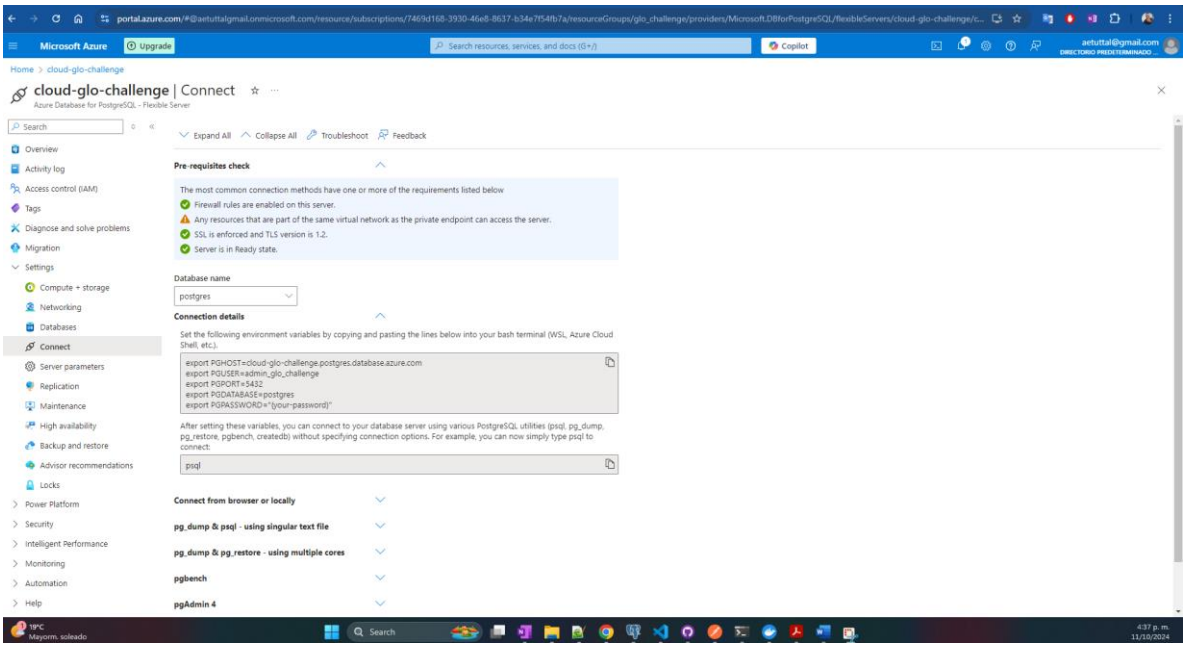
3. Bonus Track! Cloud, Testing & Containers

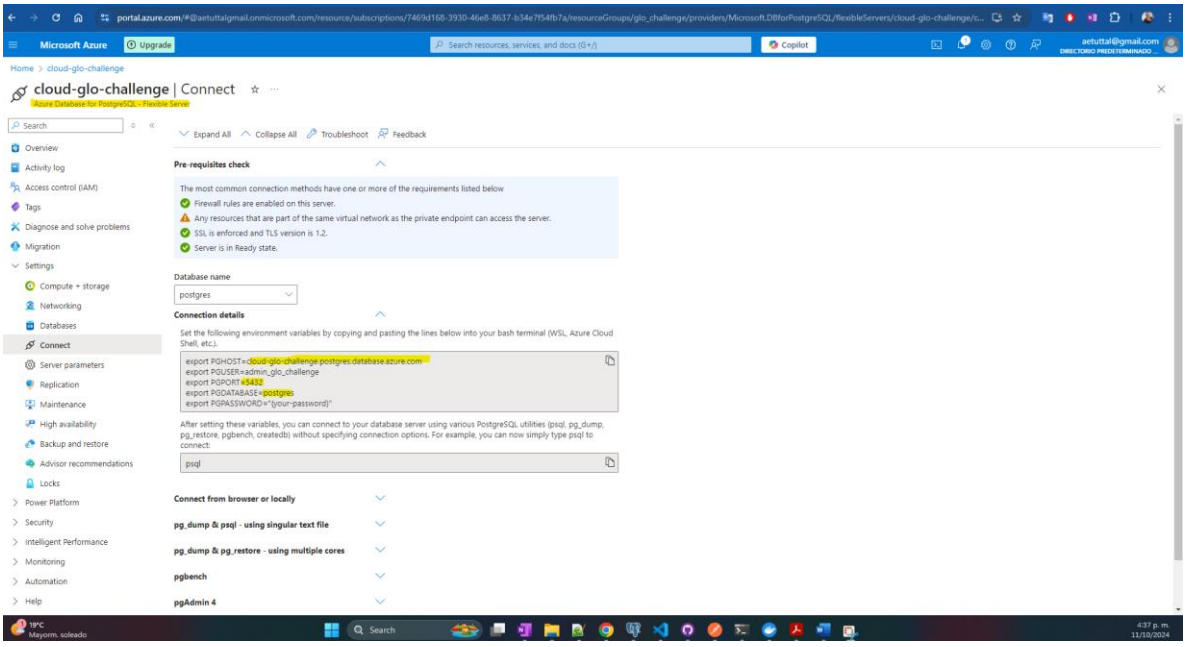
Add the following to your solution to make it more robust:

Host your architecture in any public cloud (using the services you consider more adequate)

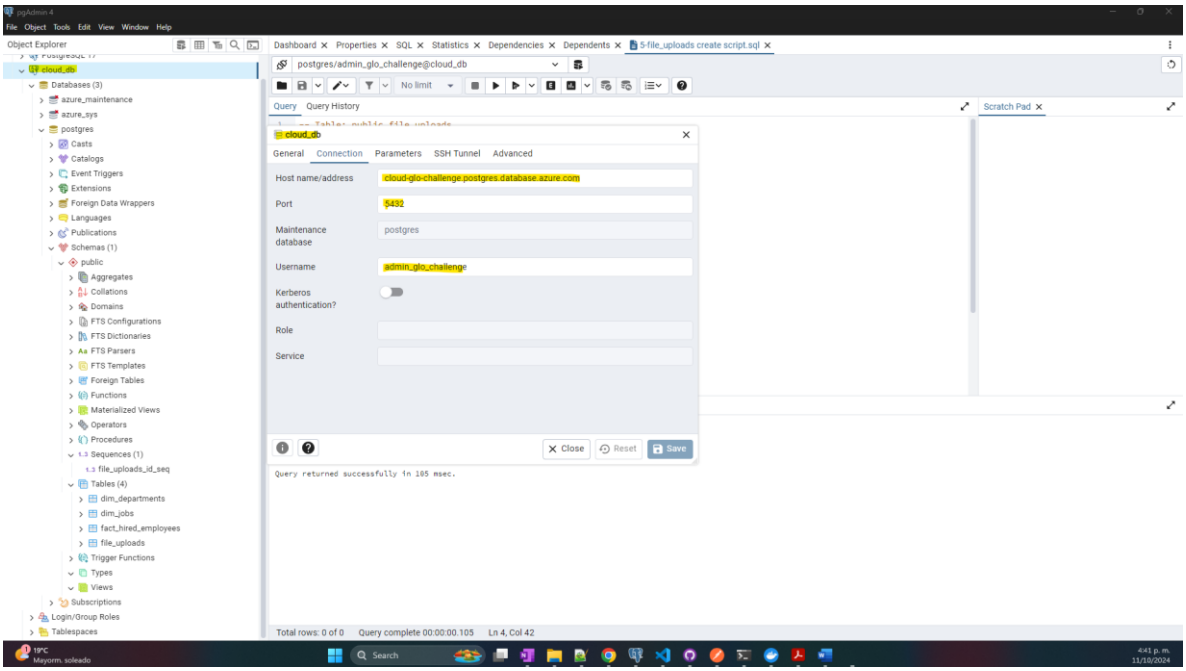
The database was created on Azure:

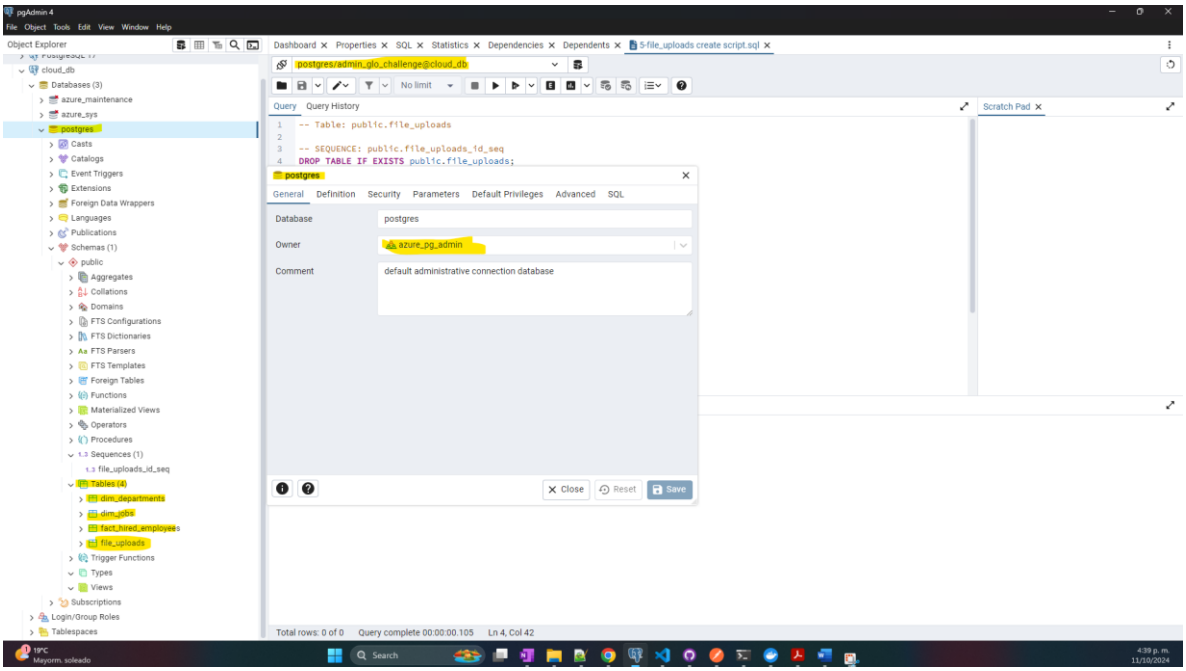




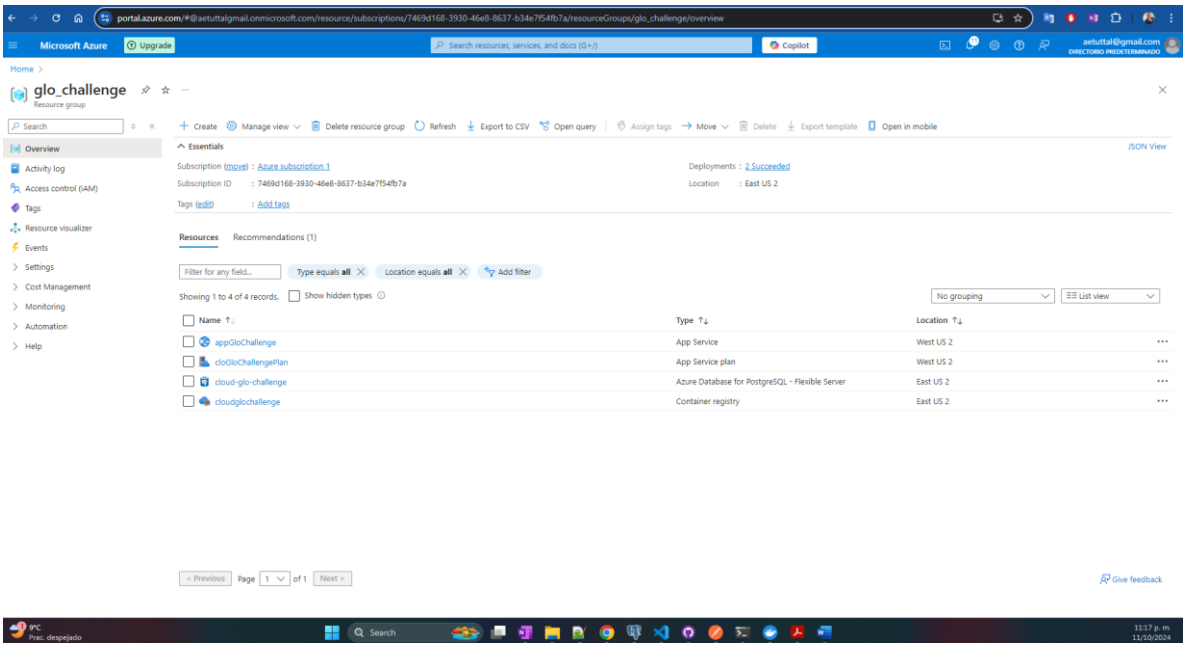


Here you can see the connection from pgAdmin.



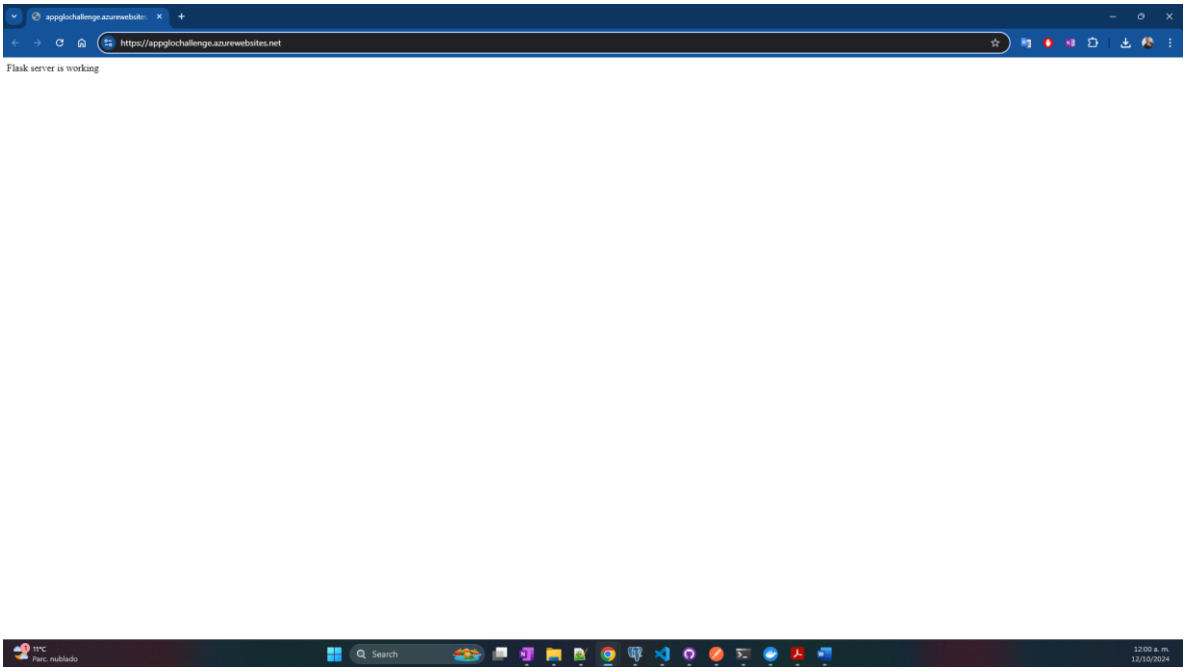


Also, the docker container was uploaded into azure:

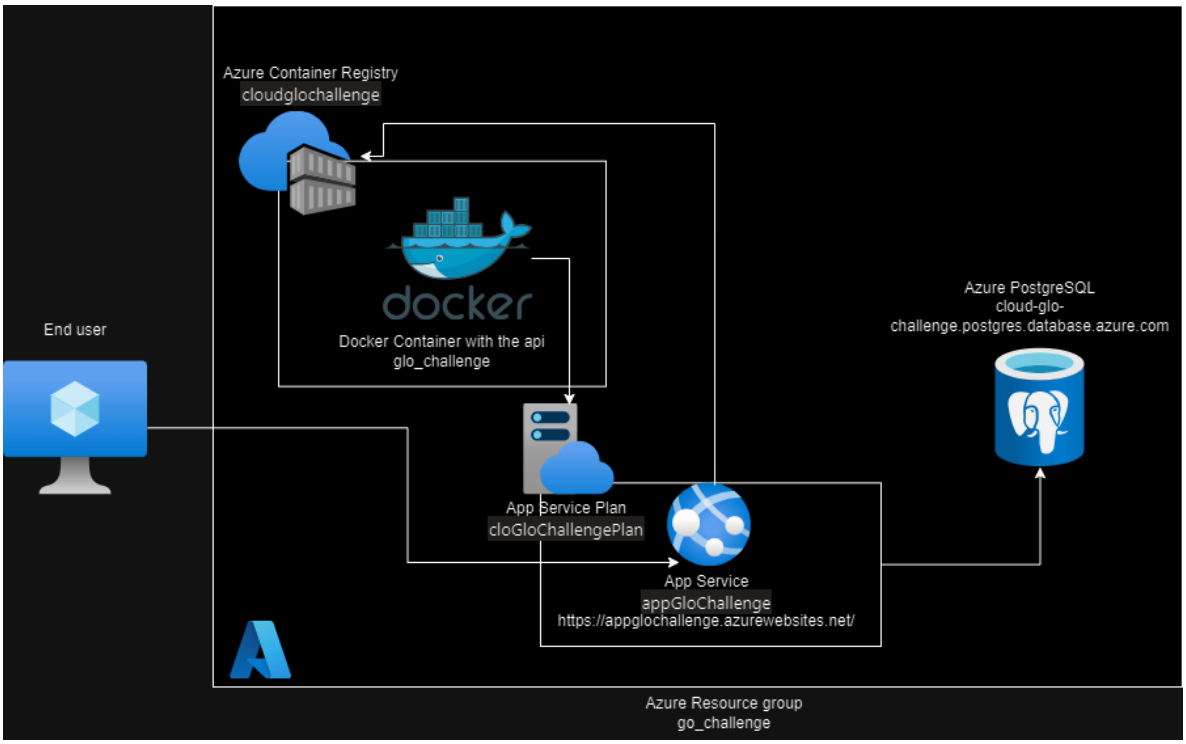


Here you can access to the api.

https://appglochallenge.azurewebsites.net/



Architecture diagram:

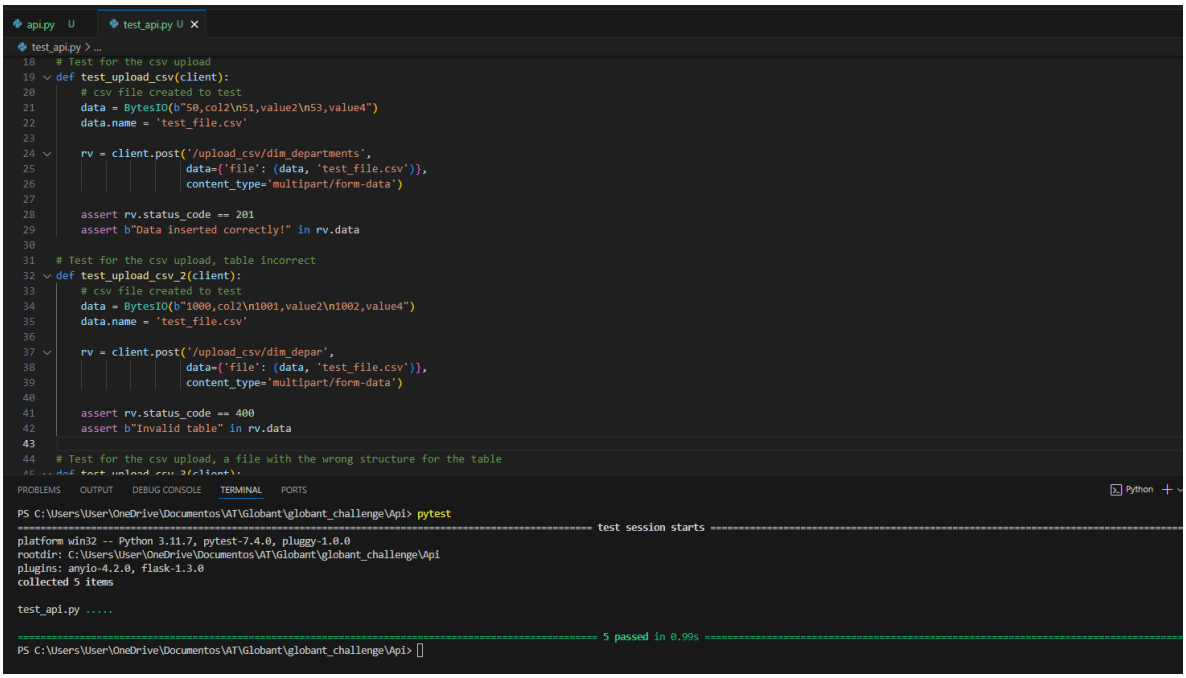


Add automated tests to the API

○ You can use whichever library that you want

○ Different tests types, if necessary, are welcome

The file test_api.py was created to test automatically the api, the library used was pytest.



```
# Test for the csv upload
def test_upload_csv(client):
    # csv file created to test
    data = BytesIO(b"50,col2\n51,value2\n53,value4")
    data.name = 'test_file.csv'

    rv = client.post('/upload_csv/dim_departments',
                     data={'file': (data, 'test_file.csv')},
                     content_type='multipart/form-data')

    assert rv.status_code == 201
    assert b"Data inserted correctly!" in rv.data

# Test for the csv upload, table incorrect
def test_upload_csv_2(client):
    # csv file created to test
    data = BytesIO(b"1000,col2\n1001,value2\n1002,value4")
    data.name = 'test_file.csv'

    rv = client.post('/upload_csv/dim_depar',
                     data={'file': (data, 'test_file.csv')},
                     content_type='multipart/form-data')

    assert rv.status_code == 400
    assert b"Invalid table" in rv.data

# Test for the csv upload, a file with the wrong structure for the table
```

```
PS C:\Users\User\OneDrive\Documentos\AT\Globant\globant_challenge\Api> pytest
platform win32 -- Python 3.11.7, pytest-7.4.0, pluggy-1.0.0
rootdir: C:\Users\User\OneDrive\Documentos\AT\Globant\globant_challenge\Api
plugins: anyio-4.2.0, flask-1.3.0
collected 5 items

test_api.py .....

5 passed in 0.99s
PS C:\Users\User\OneDrive\Documentos\AT\Globant\globant_challenge\Api>
```

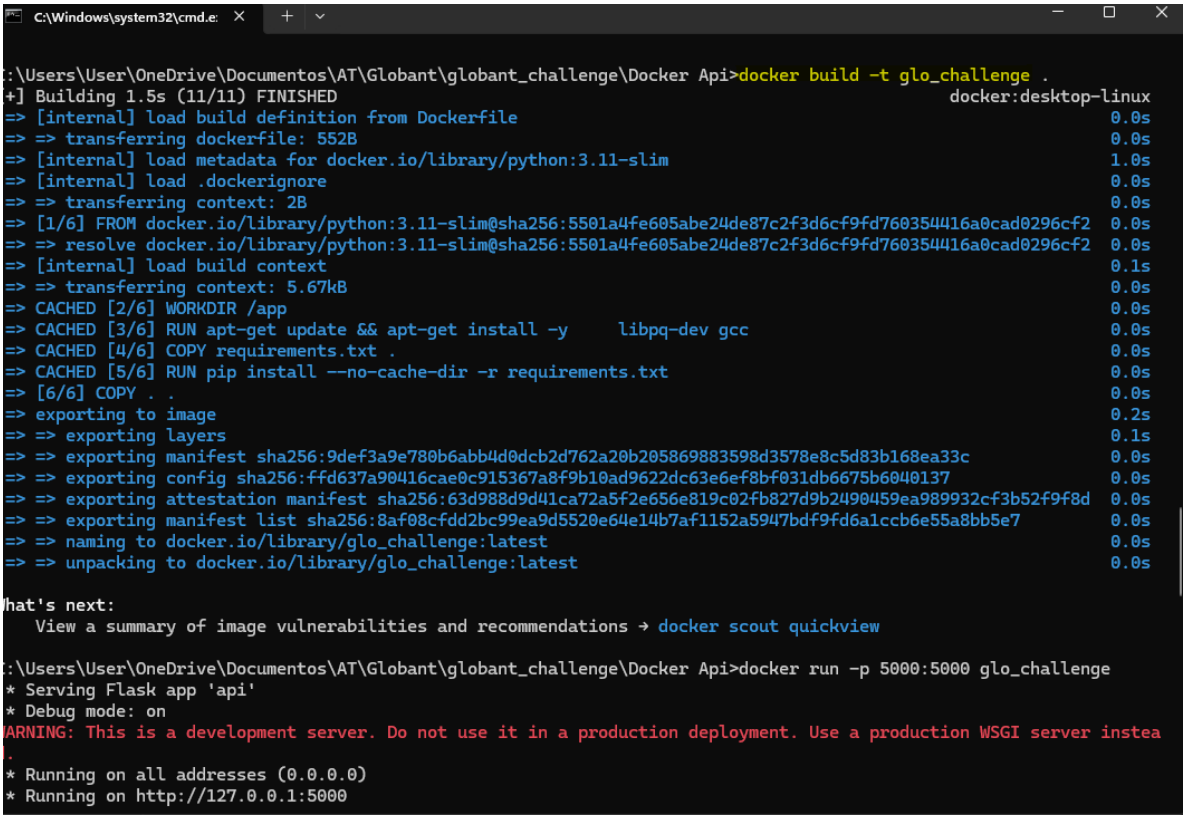
You can find the test file under the **Api** folder.

Containerize your application

○ Create a Dockerfile to deploy the package

The docker container was created, you can find the code used to configure the docker under the *docker_api* folder.

Here is the execution of the build for docker:



And the docker running with the api:

```
C:\Windows\system32\cmd.e:
 => => unpacking to docker.io/library/glo_challenge:latest                    0.0s

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\User\OneDrive\Documentos\AT\Globant\globant_challenge\Docker Api>docker run -p 5000:5000 glo_challenge
 * Serving Flask app 'api'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instea
d.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 461-683-321

C:\Users\User\OneDrive\Documentos\AT\Globant\globant_challenge\Docker Api>docker run -p 5000:5000 glo_challenge
 * Serving Flask app 'api'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instea
d.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 439-577-306
172.17.0.1 - - [11/Oct/2024 16:01:18] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [11/Oct/2024 16:05:31] "POST /upload_csv/dim_jobs HTTP/1.1" 500 -
```

Flask server is working