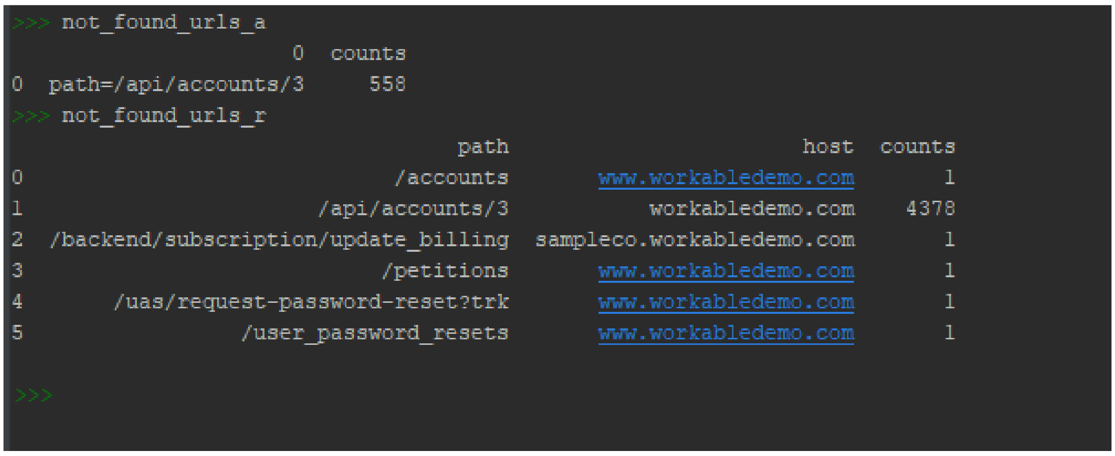
TASK 1

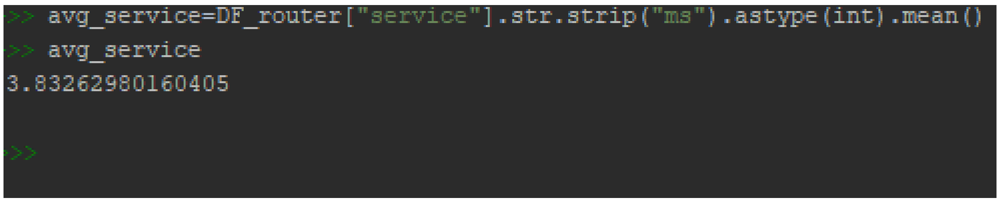
1. In order to answer to this question, only the heroku/[router] and app/web.X log entries were used. According to heroku's documentation page, the inbound requests are received by a set of routers that forward the request to one of the application's web dynos. The web dynos are the only type of dynos that can receive HTTP requests. The router logs can be correlated with the web dynos logs using the request_id field. Two different dataframes were created, one for the router logs and one for the web dynos logs. To count how many times the status=404 had appeared in each dataframe, it was ensured that the same request_id is only counted once.

The not found url with the highest count is the : workabledemo.com/api/accounts/3



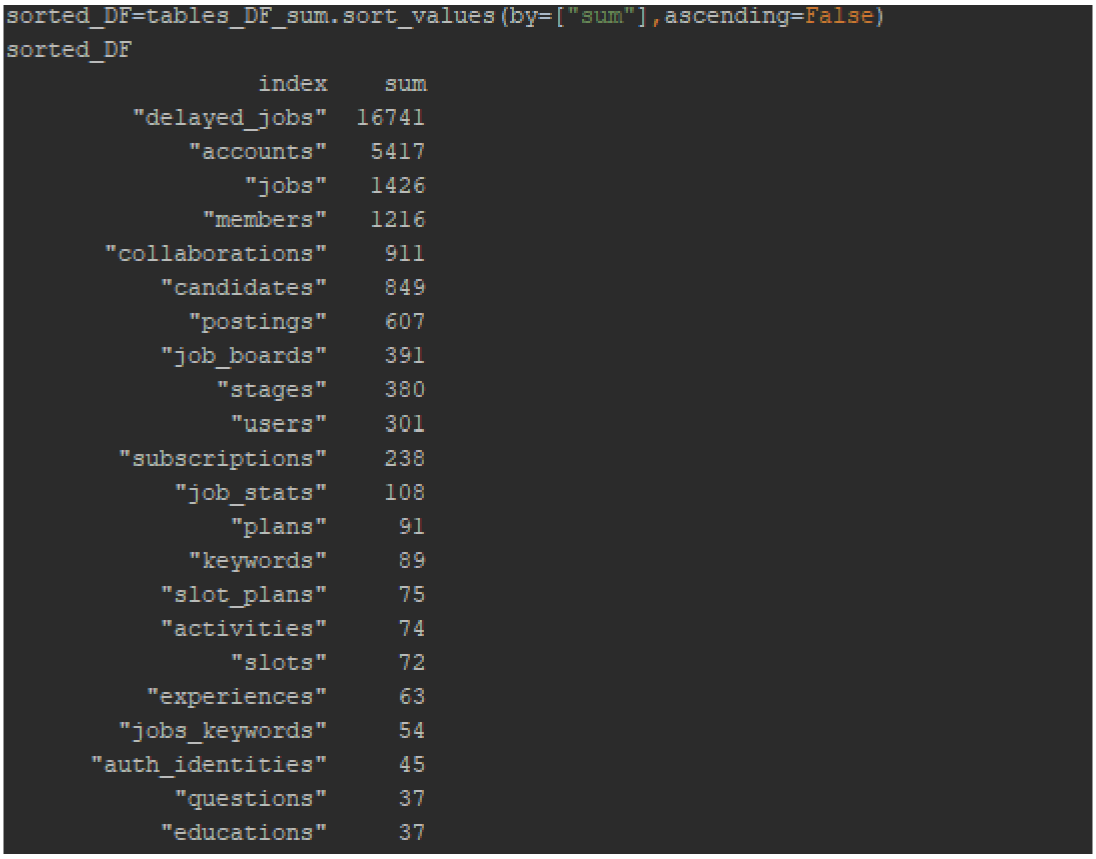
```
>>> not_found_urls_a
                        0  counts
0  path=/api/accounts/3     558
>>> not_found_urls_r
                                   path                       host  counts
0                             /accounts        www.workabledemo.com       1
1                        /api/accounts/3            workabledemo.com    4378
2  /backend/subscription/update_billing  sampleco.workabledemo.com       1
3                            /petitions        www.workabledemo.com       1
4       /uas/request-password-reset?trk        www.workabledemo.com       1
5                  /user_password_resets        www.workabledemo.com       1

>>>
```

2.



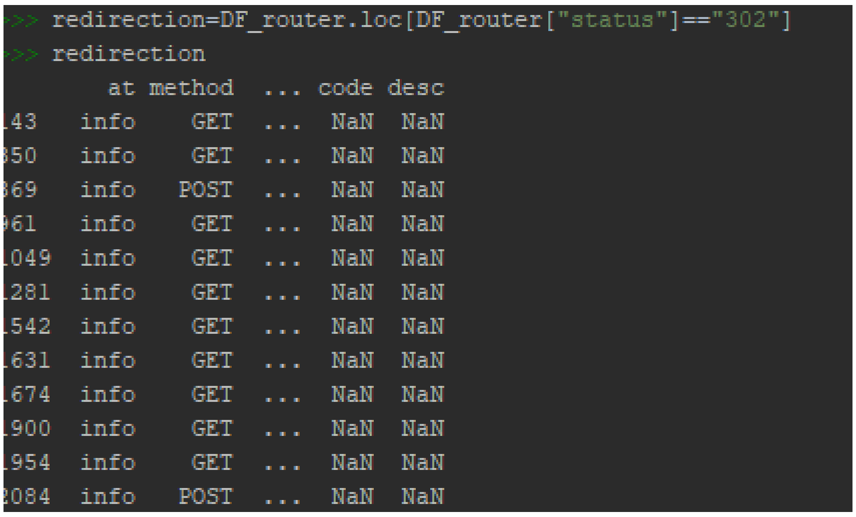
```
>> avg_service=DF_router["service"].str.strip("ms").astype(int).mean()
>> avg_service
3.83262980160405

>>
```

3. The table “delayed jobs” is more frequently loaded.



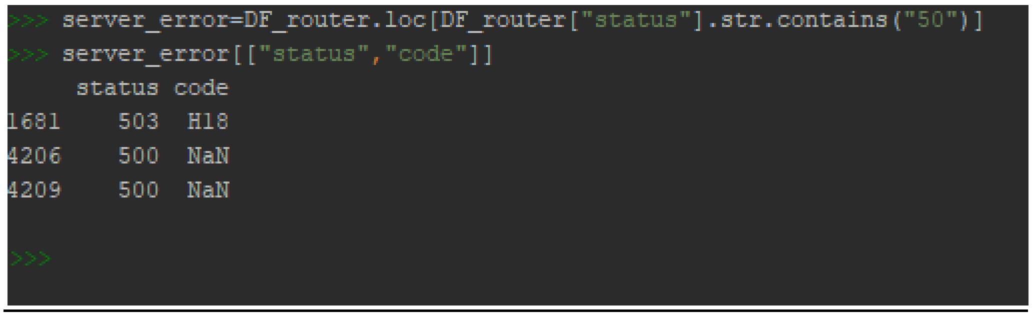
```
sorted_DF=tables_DF_sum.sort_values(by=["sum"],ascending=False)
sorted_DF
                  index    sum
         "delayed_jobs"  16741
             "accounts"   5417
                 "jobs"   1426
              "members"   1216
       "collaborations"    911
           "candidates"    849
             "postings"    607
           "job_boards"    391
               "stages"    380
                "users"    301
        "subscriptions"    238
            "job_stats"    108
                "plans"     91
             "keywords"     89
           "slot_plans"     75
           "activities"     74
                "slots"     72
          "experiences"     63
        "jobs_keywords"     54
      "auth_identities"     45
            "questions"     37
           "educations"     37
```

4. Yes URL redirection is taking place.



```
>> redirection=DF_router.loc[DF_router["status"]=="302"]
>> redirection
        at method  ... code desc
43    info    GET  ...  NaN  NaN
50    info    GET  ...  NaN  NaN
69    info   POST  ...  NaN  NaN
61    info    GET  ...  NaN  NaN
049   info    GET  ...  NaN  NaN
281   info    GET  ...  NaN  NaN
542   info    GET  ...  NaN  NaN
631   info    GET  ...  NaN  NaN
674   info    GET  ...  NaN  NaN
900   info    GET  ...  NaN  NaN
954   info    GET  ...  NaN  NaN
084   info   POST  ...  NaN  NaN
```

5. Yes, for example the error with code=H18, which according to heroku's documentation, it stands for "Server Request Interrupted" and signifies that the socket connected, some data was sent as part of a response by the app, but then the socket was destroyed without completing the response.



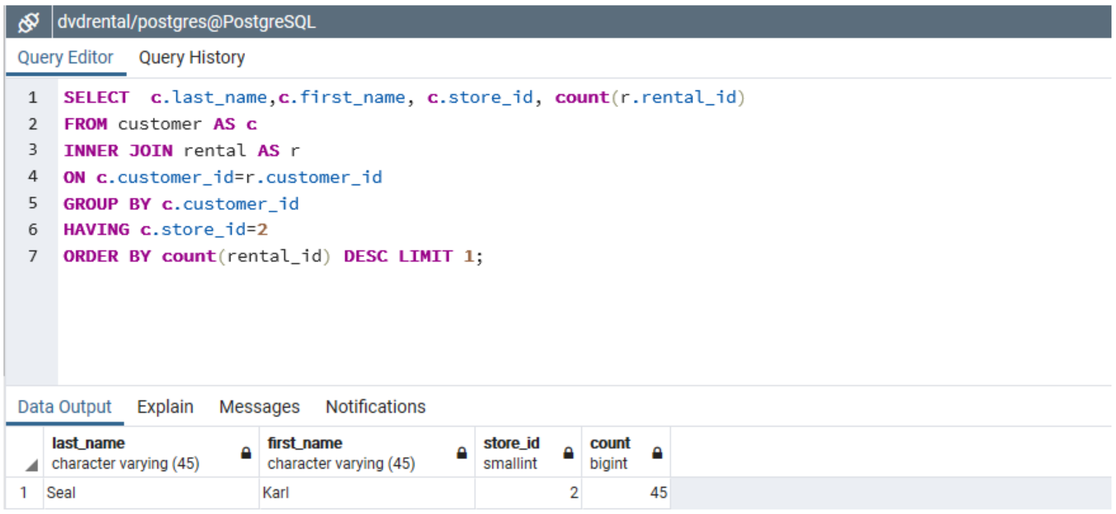
```
>>> server_error=DF_router.loc[DF_router["status"].str.contains("50")]
>>> server_error[["status","code"]]
     status code
1681    503  H18
4206    500  NaN
4209    500  NaN

>>>
```

TASK 2

1.

SQL_query= *SELECT c.last_name,c.first_name, c.store_id, count(r.rental_id) FROM customer AS c INNER JOIN rental AS r ON c.customer_id=r.customer_id GROUP BY c.customer_id HAVING c.store_id=2 ORDER BY count(rental_id) DESC LIMIT 1;*

Answer=Seal Karl has the most rentals at store 2.



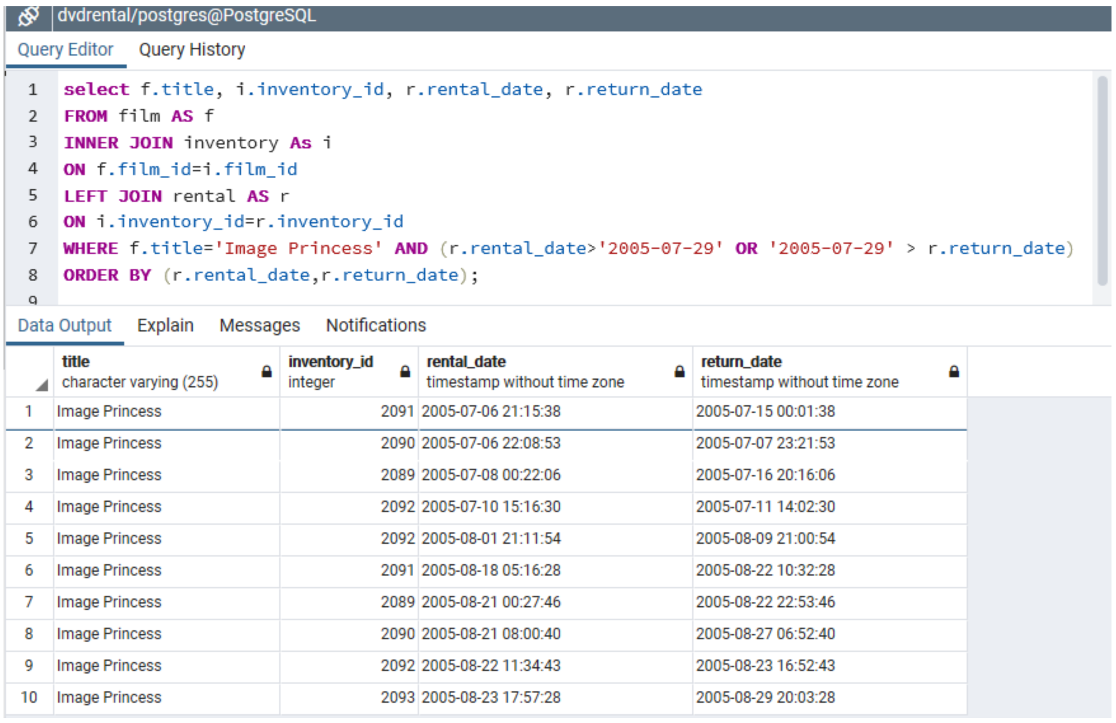
```
SELECT  c.last_name,c.first_name, c.store_id, count(r.rental_id)
FROM customer AS c
INNER JOIN rental AS r
ON c.customer_id=r.customer_id
GROUP BY c.customer_id
HAVING c.store_id=2
ORDER BY count(rental_id) DESC LIMIT 1;
```

	last_name character varying (45)	first_name character varying (45)	store_id smallint	count bigint
1	Seal	Karl	2	45

2.

SQL_query= *select f.title, i.inventory_id, r.rental_date, r.return_date FROM film AS f INNER JOIN inventory As i ON f.film_id=i.film_id LEFT JOIN rental AS r ON i.inventory_id=r.inventory_id WHERE f.title='Image Princess' AND (r.rental_date>'2005-07-29' OR '2005-07-29' >r.return_date) ORDER BY (r.rental_date,r.return_date);*

Answer=Yes he would be able to rent the copy of "Image Princess" with inventory id=2092, under the assumption that both stores have access to all inventory_ids(copys).



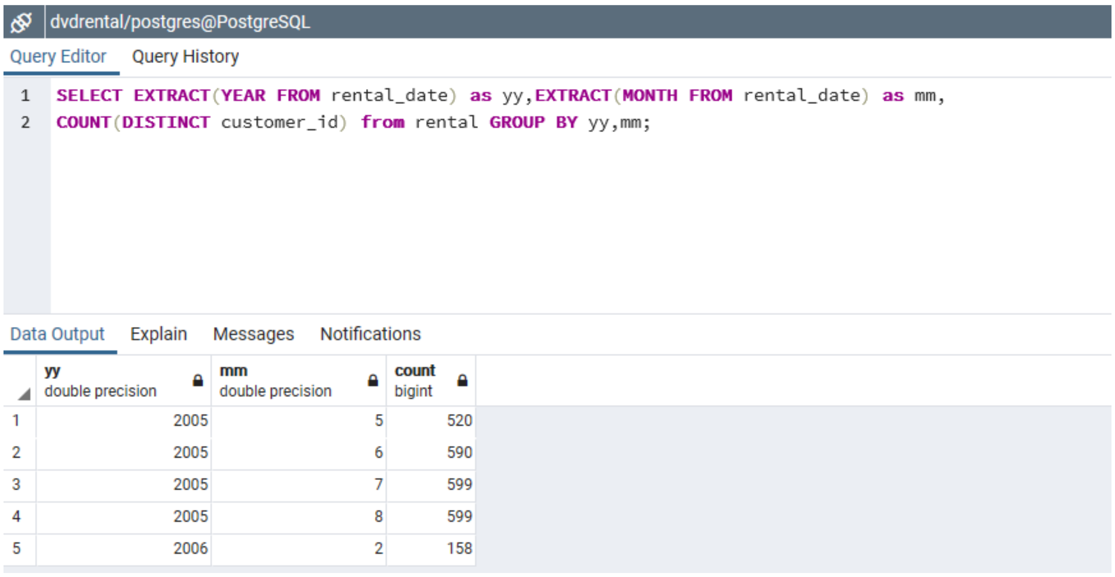
```sql
select f.title, i.inventory_id, r.rental_date, r.return_date
FROM film AS f
INNER JOIN inventory As i
ON f.film_id=i.film_id
LEFT JOIN rental AS r
ON i.inventory_id=r.inventory_id
WHERE f.title='Image Princess' AND (r.rental_date>'2005-07-29' OR '2005-07-29' > r.return_date)
ORDER BY (r.rental_date,r.return_date);
```

	title character varying (255)	inventory_id integer	rental_date timestamp without time zone	return_date timestamp without time zone
1	Image Princess	2091	2005-07-06 21:15:38	2005-07-15 00:01:38
2	Image Princess	2090	2005-07-06 22:08:53	2005-07-07 23:21:53
3	Image Princess	2089	2005-07-08 00:22:06	2005-07-16 20:16:06
4	Image Princess	2092	2005-07-10 15:16:30	2005-07-11 14:02:30
5	Image Princess	2092	2005-08-01 21:11:54	2005-08-09 21:00:54
6	Image Princess	2091	2005-08-18 05:16:28	2005-08-22 10:32:28
7	Image Princess	2089	2005-08-21 00:27:46	2005-08-22 22:53:46
8	Image Princess	2090	2005-08-21 08:00:40	2005-08-27 06:52:40
9	Image Princess	2092	2005-08-22 11:34:43	2005-08-23 16:52:43
10	Image Princess	2093	2005-08-23 17:57:28	2005-08-29 20:03:28

3.

SQL_query= *SELECT EXTRACT(YEAR FROM rental_date) as yy,EXTRACT(MONTH FROM rental_date) as mm, COUNT(DISTINCT customer_id) from rental GROUP BY yy,mm;*

Answer=The counts are presented in the picture below.


dvdrental/postgres@PostgreSQL

Query Editor · Query History

```
SELECT EXTRACT(YEAR FROM rental_date) as yy,EXTRACT(MONTH FROM rental_date) as mm,
COUNT(DISTINCT customer_id) from rental GROUP BY yy,mm;
```

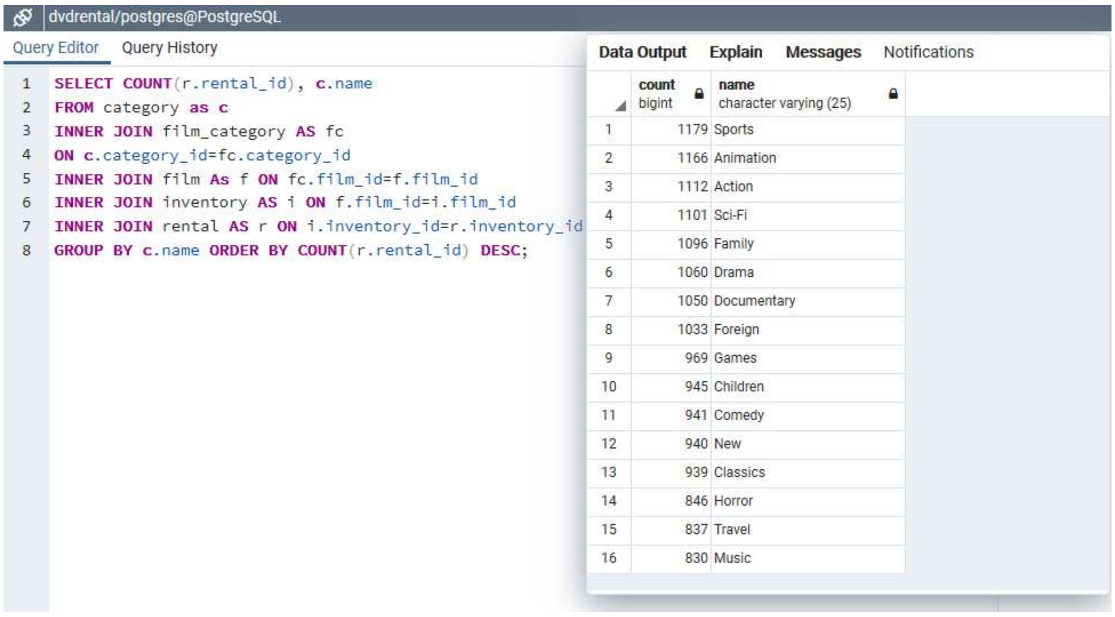
Data Output · Explain · Messages · Notifications

	yy double precision	mm double precision	count bigint
1	2005	5	520
2	2005	6	590
3	2005	7	599
4	2005	8	599
5	2006	2	158

4.

SQL_query= *SELECT COUNT(r.rental_id), c.name FROM category as c INNER JOIN film_category AS fc ON c.category_id=fc.category_id INNER JOIN film As f ON fc.film_id=f.film_id INNER JOIN inventory AS i ON f.film_id=i.film_id INNER JOIN rental AS r ON i.inventory_id=r.inventory_id GROUP BY c.name ORDER BY COUNT(r.rental_id) DESC;*

Answer=Sports is the most popular category among the store's customers.


dvdrental/postgres@PostgreSQL

Query Editor · Query History

```
SELECT COUNT(r.rental_id), c.name
FROM category as c
INNER JOIN film_category AS fc
ON c.category_id=fc.category_id
INNER JOIN film As f ON fc.film_id=f.film_id
INNER JOIN inventory AS i ON f.film_id=i.film_id
INNER JOIN rental AS r ON i.inventory_id=r.inventory_id
GROUP BY c.name ORDER BY COUNT(r.rental_id) DESC;
```

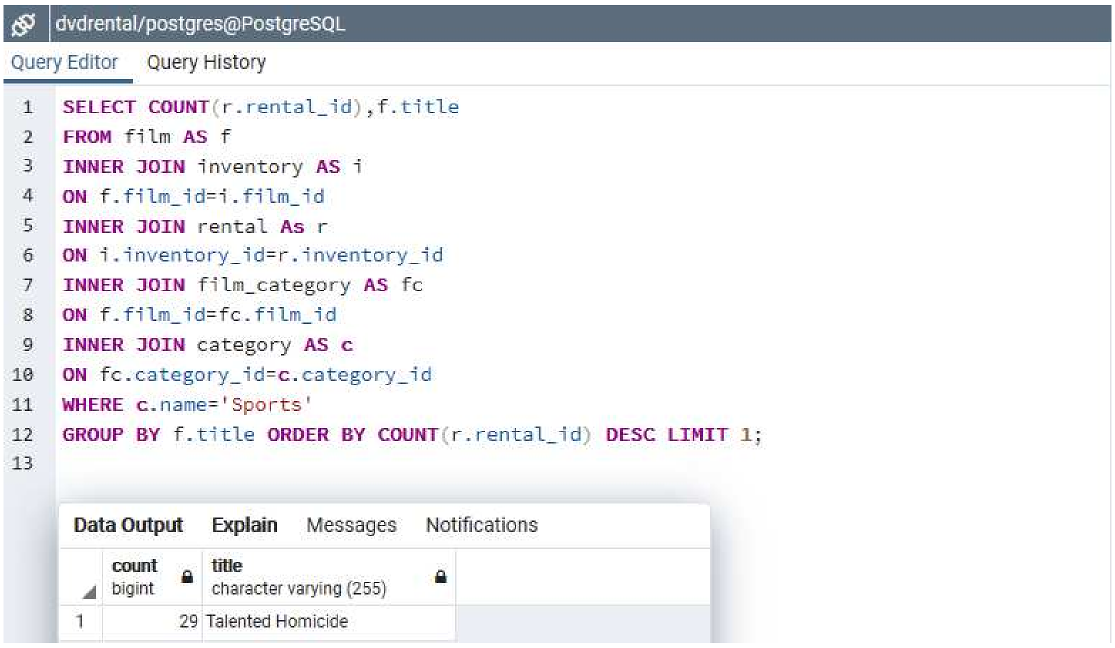
Data Output · Explain · Messages · Notifications

	count bigint	name character varying (25)
1	1179	Sports
2	1166	Animation
3	1112	Action
4	1101	Sci-Fi
5	1096	Family
6	1060	Drama
7	1050	Documentary
8	1033	Foreign
9	969	Games
10	945	Children
11	941	Comedy
12	940	New
13	939	Classics
14	846	Horror
15	837	Travel
16	830	Music

5.

SQL_query= *SELECT COUNT(r.rental_id),f.title FROM film AS f INNER JOIN inventory AS I ON f.film_id=i.film_id INNer JOIN rental As r ON i.inventory_id=r.inventory_id INNER JOIN film_category AS fc ON f.film_id=fc.film_id INNER JOIN category AS c ON fc.category_id=c.category_id WHERE c.name='Sports' GROUP BY f.title ORDER BY COUNT(r.rental_id) DESC LIMIT 1;*

Answer=Talented Homicide.



```
SELECT COUNT(r.rental_id),f.title
FROM film AS f
INNER JOIN inventory AS i
ON f.film_id=i.film_id
INNER JOIN rental As r
ON i.inventory_id=r.inventory_id
INNER JOIN film_category AS fc
ON f.film_id=fc.film_id
INNER JOIN category AS c
ON fc.category_id=c.category_id
WHERE c.name='Sports'
GROUP BY f.title ORDER BY COUNT(r.rental_id) DESC LIMIT 1;
```

	count bigint	title character varying (255)
1	29	Talented Homicide

6. Examples of other insights that we can obtain from the data are the following:

- Find out which customers are the oldest and more active in order to reward them with a discount.
- Find out which customers exceed the movies' rental duration , in order to impose a penalty.
- Find out for each individual customer what their favorite category is and recommend them the highest rated movies from that category that they haven't already seen.

These actions could increase the business' income and ensure the existence of a loyal customer base.

<u>**TASK 3**</u>

Web Application using the Flask framework for accessing TMDB API and storing information into MySQL Server.

The project contains 3 different files:

<u>*app.py*</u> : script for creating and launching the application.

<u>*index.html*</u>: html page to display the list of movies currently in theatres in Greece

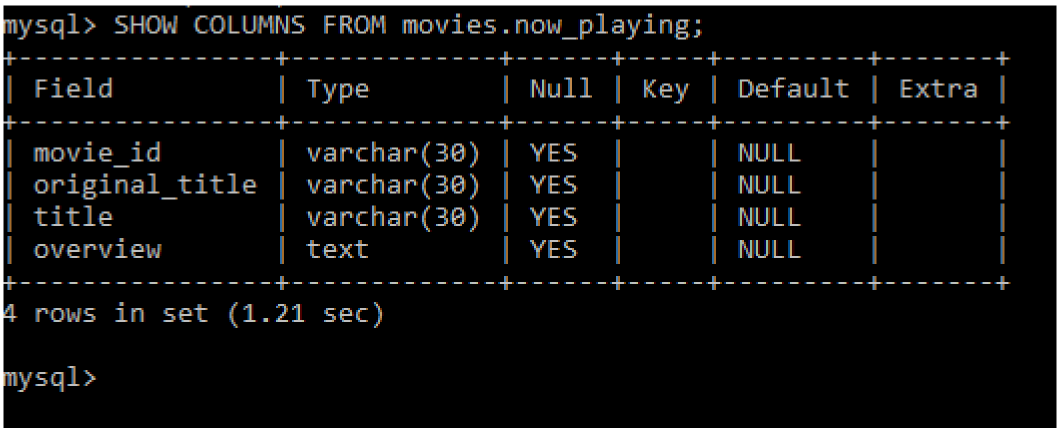
<u>*db.yaml*</u>: configuration file for connection with db.

<u>*DDL Commands*</u>

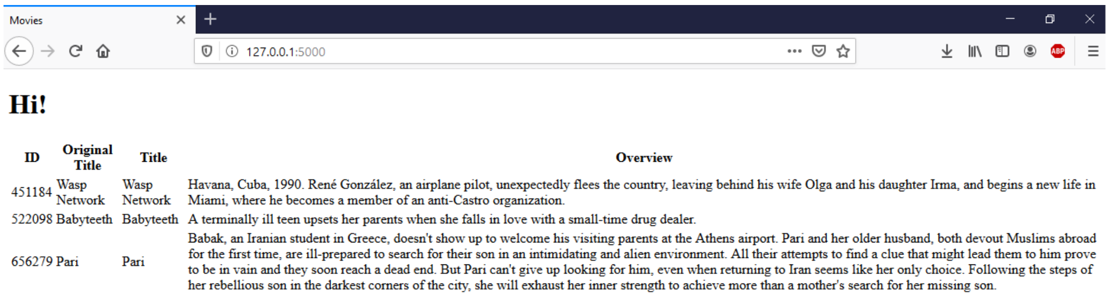
CREATE DATABASE movies;

CREATE TABLE now_playing(movie_id varchar(30),original_title varchar(30),title varchar(30),overview text(300));

ALTER TABLE now_playing MODIFY overview text(1000);



```
mysql> SHOW COLUMNS FROM movies.now_playing;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| movie_id       | varchar(30) | YES  |     | NULL    |       |
| original_title | varchar(30) | YES  |     | NULL    |       |
| title          | varchar(30) | YES  |     | NULL    |       |
| overview       | text        | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
4 rows in set (1.21 sec)

mysql>
```



Hi!

ID	Original Title	Title	Overview
451184	Wasp Network	Wasp Network	Havana, Cuba, 1990. René González, an airplane pilot, unexpectedly flees the country, leaving behind his wife Olga and his daughter Irma, and begins a new life in Miami, where he becomes a member of an anti-Castro organization.
522098	Babyteeth	Babyteeth	A terminally ill teen upsets her parents when she falls in love with a small-time drug dealer.
656279	Pari	Pari	Babak, an Iranian student in Greece, doesn't show up to welcome his visiting parents at the Athens airport. Pari and her older husband, both devout Muslims abroad for the first time, are ill-prepared to search for their son in an intimidating and alien environment. All their attempts to find a clue that might lead them to him prove to be in vain and they soon reach a dead end. But Pari can't give up looking for him, even when returning to Iran seems like her only choice. Following the steps of her rebellious son in the darkest corners of the city, she will exhaust her inner strength to achieve more than a mother's search for her missing son.