Part 1

Below you will find the code for the search page search.py.
Alternatively, you can access the webpage using the following link.
Link: http://34.67.206.69/search.py

The code:

```
#!/usr/bin/python
import cgi
import cgitb
import MySQLdb.f

cgitb.enable()

#make html
print ('content-type: text/html; charset=utf-8')
print ()
print ('<html>')
print ('  <head>')
print ('    <title> Search Page </title>')
print ('  </head>')
print ('  <body>')
print ('    <h1> Movies Search Page </h1>')

#make form (search button)
print ('    <form action="search.py" method="get">')
print ('    <input type="text" name="query" placeholder="Movie Title" />')
print ('    <input type="submit" value="search" />')
print ('    </form>')
print ('    <br><br>')

# get query
form = cgi.FieldStorage()
if 'query' in form:
      query=form['query'].value
      # connect to database
      db = MySQLdb.connect('localhost','eba3420','EBA3420','movies')
      cursor = db.cursor()
      #get results
      query_one = "select id, title from movies where title like '%{}%'".format(query)
      n = cursor.execute(query_one)
      if n > 0:
            myresult = cursor.fetchall()
            cursor.close()
            db.close()
```

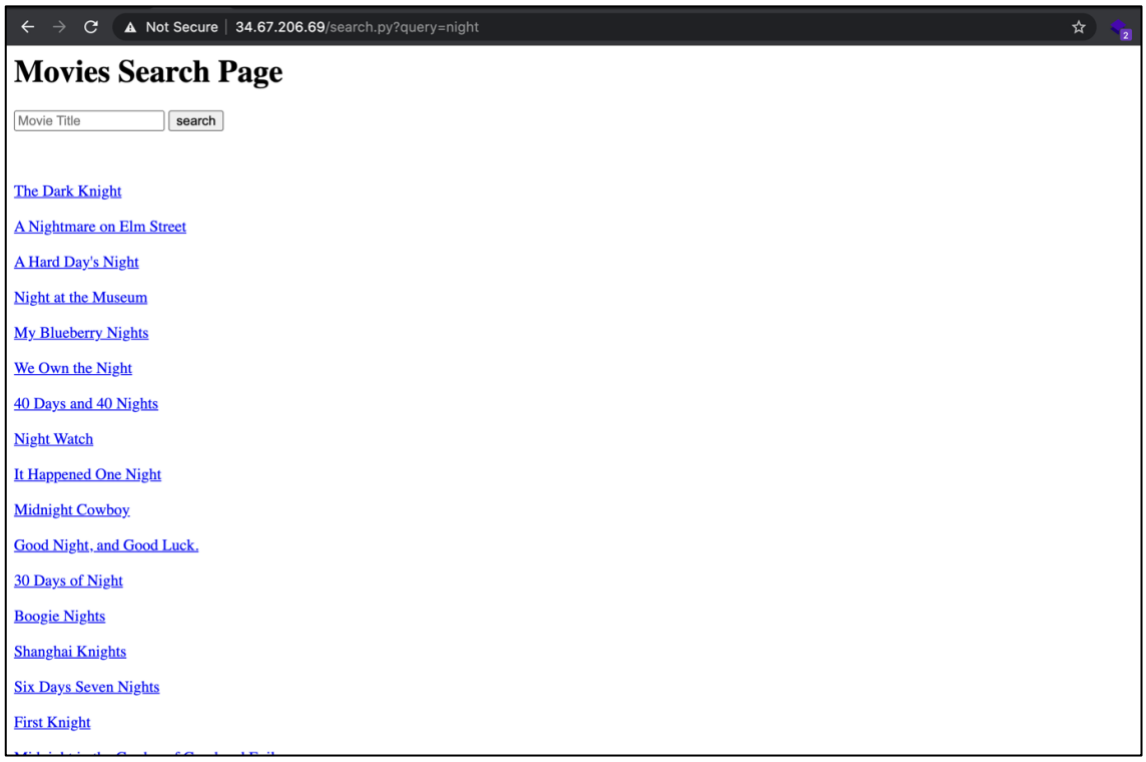
```
        #create links
        for x in myresult:
            print ('       <A href="movies.py?id={}">{}</A>'.format(x[0], x[1]))
            print ('<br></br>')

print ('  </body>')
print ('</html>')
```

Code explanation:

The above code represents a search page “search.py”. It contains a simple html code written in python (server-side scripting). The code uses an html form to create a search button from which the user will search the movie name. Then we use cgi.FielStorage to get the data from the http request. If there is a query in the http request, then the code connects to MySQL database where we give it an SQL statement to search for all the results that match the search term. On the SQL statement, we use “LIKE” instead of “=” in order to get all results that contain the search query. For example, if we search “love” all results containing the term “love” in their title will appear. We use myresult = cursor.fetchall since we want all results that match the search term. Then, for every result, the program will iterate and provide the Title name as an html link that will redirect to the page “movies.py” which contains information and details about all of the movies.

For example: A search of the string “night” will give the following result:

Screenshot (search.py):



Part 2

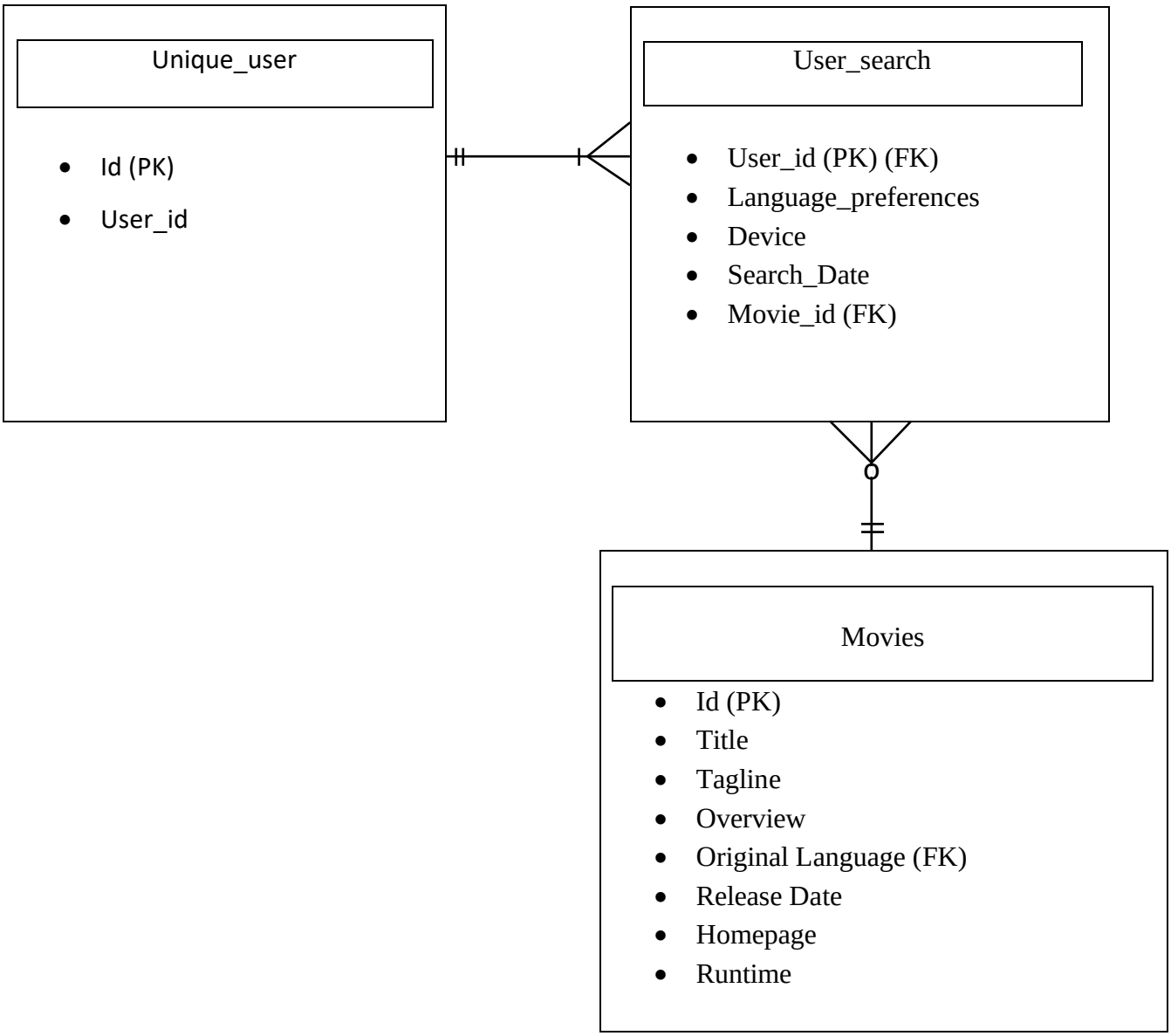
In the following part, we have created two tables called “User_search” and “Unique_user”. Below you will find the SQL code and description of the tables as well as ER Diagrams for both tables and explanations for the diagrams.

```
CREATE TABLE IF NOT EXISTS User_search (
User_id int,
Language_preference Varchar (100),
Device Varchar (100),
Search_Date DATETIME default CURRENT_TIMESTAMP,
Movie_id Int,
PRIMARY KEY (User_id)
CONSTRAINT FOREIGN KEY (Movie_id) REFERENCES movies(id),
CONSTRAINT FOREIGN KEY (User_id) REFERENCES Unique_user(id)
);

CREATE TABLE IF NOT EXISTS Unique_user (
id INT NOT NULL AUTO_INCREMENT,
User_id varchar (100) NOT NULL,
Primary key (id)
);
```

Explanation:

Each unique user from the table “Unique_user” will have a permanent cookie (permid) registered under User_id in order for the user to be automatically recognized and not have to register them as a new user every time. Then, for each User_id there will be an assigned “id” by the statement “id INT NOT NULL AUTO_INCREMENT” which will assign values progressively (i.e. 1,2,3, 4,…n+1) for as many Unique User_id-s there are. For example, a User_id with permid as SoexqRs23IH1EzFT9Q8kAPG6yYCp40Ni will have the id value of 12 in order for it to be recognized more easily and not have to read the entire permid letter by letter. We will set “id” as the primary key which we will then use in table “Users” as a foreign key.

Next, we will create a new table dubbed “Users” that will store the data of the users and their activity. For every search from the user, we will be able to see from what device they searched from, what language preferences did they have, the time when they searched the movie and which movie they searched for (Movie_id). Since this data is sensitive to change for every search, we put it in this table whereas the User_id (permid) will remain the same, hence we put that in a different table. For us to identify to which user this data belongs to, we have set User_id as the foreign key with reference to “id” from Unique_users. This way we will have the user’s data registered under user “i.e., 12” rather than the data of user “SoexqRs23IH1EzFT9Q8kAPG6yYCp40Ni “, which is easier to understand and read.



Explanation:

For each user, there can be one or many searches. The reason why there cannot be zero or many searches is that in order for us to get the User_id we need to get it from a permanent cookie, which is generated only after the search has been made. However, each specific search can only have one user, hence, we have a one-to-many relations diagram between Unique_user and User_search.

Likewise, for each movie, there can be many different searches. However, for each search, there can only be one movie, hence, we have a one-to-many relations diagram between User_search and Movies.

We have not drawn the rest of the diagrams since this is the extension we have made to the database; the rest of ER diagrams remain as they are displayed in Appendix A.

Part 3

In the following part, you will find the code, link, and description of the movies.py page. This is the page the user is redirected to after pressing on any of the results they get from the search page. A link to access the page is found below:

Link: http://34.67.206.69/movies.py?id=155

The code:

```
#!/usr/bin/python
import cgi
import cgitb
import eba3420
import MySQLdb
cgitb.enable()

# get http
print('content-type: text/html; charset=utf-8')

# set/get permanent cookie
cookies=eba3420.get_cookies()
if not 'PERMID' in cookies:
   permid = eba3420.get_random_string32()
   print('Set-Cookie: PERMID=%s; Expires Wed, 21 April 2022 10:00:00 GMT' %permid)
else:
   permid = cookies['PERMID']

user_agent = eba3420.get_user_agent()
language = eba3420.get_language_preferences()

# connect to database
db = MySQLdb.connect('localhost','eba3420','EBA3420','movies')
cursor = db.cursor()#

query = ("select * from Unique_user where User_id=\'{}\'".format(permid))
n = cursor.execute(query)
if not n > 0:
      cursor.execute("INSERT INTO Unique_user (User_id) values (\'{}\')".format(permid))
      db.commit()

# get form
form = cgi.FieldStorage()
```

```
if 'id' in form:
   id = form['id'].value
else:
   id = None

# get user_id of user
query = "Select * from Unique_user where User_id=\'{}\'".format(permid)
n = cursor.execute(query)
if n > 0:
      myresult = cursor.fetchall()
      userId = myresult[0][0]

# store search data in database
query = '''INSERT INTO Users_search (User_id, Language_preference, Device, Movie_id)
values ({},\'{}\',\'{}\',{})'''.format(int(userId), language, user_agent, id)
n = cursor.execute(query)

db.commit()

# get movie from database to display
query = '''Select m.Title,m.Tagline,m.Overview,l.name as Original_Language
,m.Runtime,m.HomePage,m.Release_Date,pct.name as Production_countries
from movies m
join movies_countries mct on mct.movie_id=m.id
join production_countries pct on pct.code=mct.country_code
join languages l on l.code=m.original_language
where m.id={}'''.format(id)

n = cursor.execute(query)
if n > 0:
      movie = cursor.fetchall()

# get genres data
query_genres = '''Select m.id,g.name from movies m
join movies_genres mg on mg.movie_id=m.id
join genres g on g.id=mg.genre_id
where m.id={}'''.format(id)

n = cursor.execute(query_genres)
if n > 0:
      genres_list = cursor.fetchall()

# iterate thorugh genres results
genres_str = ""
for i in genres_list:
```

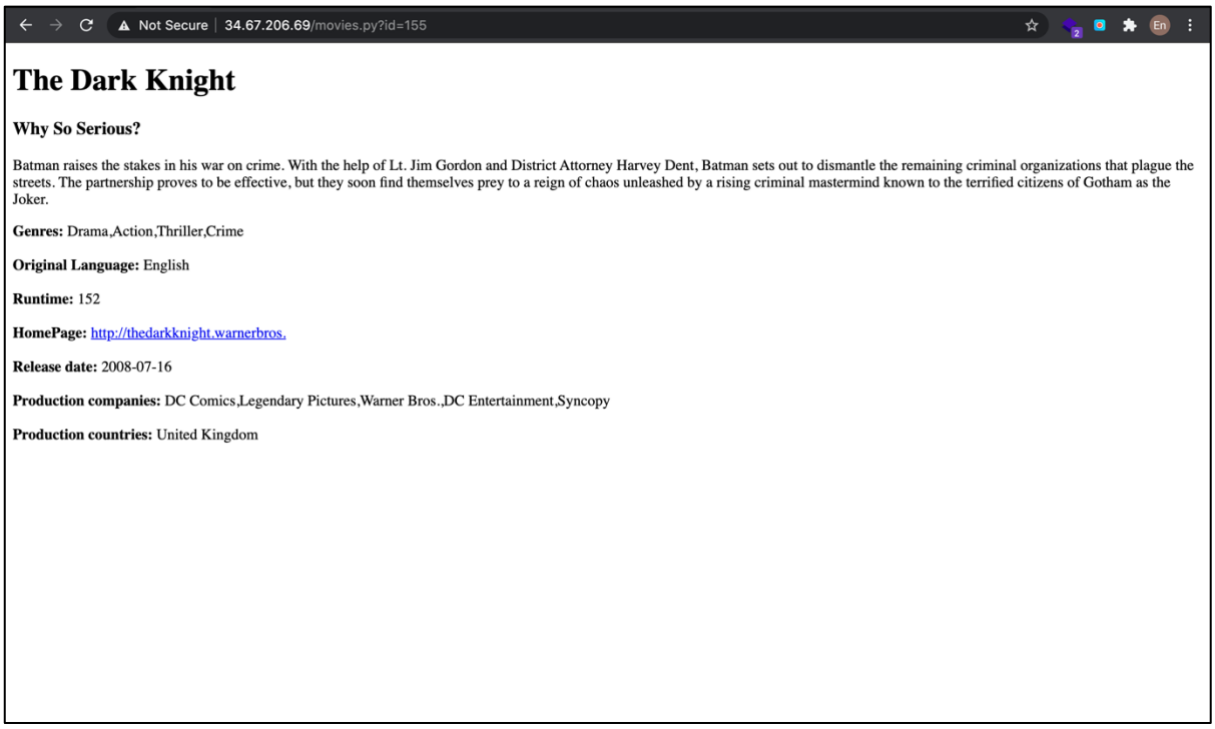
```
        genres_str = genres_str + "," + i[1]

genres_str = genres_str.strip(",")

# get production company data
query_production = '''Select m.id,pc.name from movies m
join movies_companies mc on mc.movie_id=m.id
join production_companies pc on pc.id=mc.company_id
where m.id={}'''.format(id)

n = cursor.execute(query_production)
if n > 0:
        prod_list = cursor.fetchall()

# iterate through production company data
prod_str = ""
for i in prod_list:
        prod_str = prod_str + "," + i[1]
        prod_str = prod_str.strip(",")

# make html
Print ()
print ('<!DOCTYPE html>')
print ('<HTML>')
print ('    <H1>{}</H1>'.format(movie[0][0]))
print ('    <H3>{}</H3>'.format(movie[0][1]))
print ('  <BODY>')
print ('    <p>{}</p>'.format(movie[0][2]))
print ('    <b>Genres:</b> {}'.format(genres_str))
print ('<br></br>')
print ('    <b>Original Language:</b> {}'.format(movie[0][3]))
print ('<br><br>')
print ('    <b>Runtime:</b> {}'.format(movie[0][4]))
print ('<br></br>')
print ('    <b>HomePage:</b> <A href={}>{}</A>'.format(movie[0][5], movie[0][5]))
print ('<br></br>')
print ('    <b>Release date:</b> {}'.format(movie[0][6]))
print ('<br></br>')
print ('    <b>Production companies:</b> {}'.format(prod_str))
print ('<br></br>')
print ('    <b>Production countries:</b> {}'.format(movie[0][7]))
print ('<br></br>')
print ('  </BODY>')
print ('</HTML>')
```

Code explanation:

Explanation:

The above code has several functions. It uses a permanent cookie to identify each unique user which we set at the beginning. If the permanent cookie already exists, it does not generate a new one but gets the value from the query. This cookie is stored in the database "Unique_user" that we created which serves as a unique identifier of each individual user. Moving further, we make the form where we get the movie id that the user presses on through the links on "search.py" where each link redirects to this page and has the movie id as part of the query. Since we identify the users "id" with numbers such as (1,2,3,4,…) instead of the entire permanent id, we create an SQL query to get the "id" where the "user_id" (permid) matches the permid we get from http. Then we create another SQL query that inserts the user search data such as language preferences, device etc and registers it together with the "id" we got from the previous SQL query, so the user becomes easily identified. Furthermore, we create an SQL query to get the data from the database and display it on the movies.py page. The data that we get should match the "id" found in the query so we add a WHERE clause where the movie id from the movies table matches the id in the query which we set by a placeholder. Subsequently, we get the genres and production companies data for display in a similar way but where we also created a loop so that the program iterates through all the results. After getting all the necessary data we need for creating the display page with the information about the movies, we create the html.
Below you will find a screenshot with information for the movie "Dark Knight", which was the first result we got when searching "night" on search.py.

Screenshot:



Part 4

In the following page statistics.py, we have created a webpage that will show the data of user statistics. The data will contain the number of unique users that have visited movies.py, the total number of movies searched, and the average number of movies searched per user.

The link, code and code explanation for statistics.py are found below:

Link: http://34.67.206.69/statistics.py

The code:

```
#!/usr/bin/python
import cgi
import cgitb
import eba3420
import MySQLdb
import datetime
import pandas as pd

cgitb.enable()

img_width = 400
print("Content-Type: text/html;charset=utf-8")
print()
# get form
form = cgi.FieldStorage()

# get start and end date from form
if 'start' in form:
   start = form['start'].value
else:
   start = str(datetime.date.today())

if 'end' in form:
   end = form['end'].value
else:
   end = str(datetime.date.today())

# connect to database
db = MySQLdb.connect('localhost','eba3420','EBA3420','movies')
cursor = db.cursor()
```

```
# find no. of unique users
# Query to find no. of Unique users in the selected date range
query = '''Select count(distinct user_id),cast(Search_Date as date)  from User_search where
date(search_date) >= '{}' and date(search_date) <= '{}'
group by cast(Search_Date as date);'''.format(start, end)

n = cursor.execute(query)

if n > 0:
    result = cursor.fetchall()
    df = pd.DataFrame(list(result), columns=["Unique Users","Date"])
    df = df.set_index("Date")
    plot = df.plot.bar()
    page_view_img = eba3420.plot_to_img_str(plot)
else:
    page_view_img = None

# find total no. of movies searched per day
# query to find total no. of movies searched per day

query = '''Select count(User_id),Search_Date from User_search
where date(search_date) >= '{}' and date(search_date) <= '{}'
group by cast(Search_Date as date)'''.format(start, end)

n = cursor.execute(query)
if n > 0:
    result = cursor.fetchall()
    df = pd.DataFrame(list(result), columns=["Total Movies Searched", "Date"])
    df = df.set_index("Date")
    plot = df.plot.bar()
    unique_page_view = eba3420.plot_to_img_str(plot)
else:
    unique_page_view = None

# Third query to get average of number of unique users/ total movies searched per  day
query = '''Select cast(count(distinct Movie_id) as int)/cast(count(distinct User_id) as int) as
count_user ,cast(Search_Date as date) from User_search where date(search_date) >= '{}'
and date(search_date) <= '{}' group by cast(Search_Date as date)'''.format(start, end)

n = cursor.execute(query)

if n > 0:
    result = cursor.fetchall()
    df = pd.DataFrame(list(result), \
                      columns=["Average number of movies searched by user","Date"])
    df["Average number of movies searched by user"] = df["Average number of movies
searched by user"].astype(float)
```

```
    df = df.set_index("Date")
    plot = df.plot.bar()
    session_img = eba3420.plot_to_img_str(plot)
else:
    session_img = None

cursor.close()
db.close()

# make html
print('<!DOCTYPE html>')
print('<HTML>')
print(' <HEAD>')
print(' <TITLE>Web statistics</TITLE>')
print(' </HEAD>')
print(' <BODY>')
print(' <H1>Web statistics</H1>')
print(' <FORM method="get" action="statistics.py">')
print(' From:')
print(' <INPUT type="date" name="start" value="%s" ' \
     'onchange="this.form.submit()"/>' % start)
print(' To:')
print(' <INPUT type="date" name="end" value="%s" ' \
     'onchange="this.form.submit()"/>' % end)
print('</FORM>')

if page_view_img != None:
       print(' <IMG src="%s" width="%s" />' % \
            (page_view_img, img_width))
else:
    print(' <P>No page view data for period %s to %s </P>' % \
         (start,end))

if unique_page_view != None:
       print(' <IMG src="%s" width="%s" />' %
            (unique_page_view, img_width))
else:
    print(' <P>No unique page view data for period %s to %s</P>' \
         % (start, end))

if session_img != None:
       print(' <IMG src="%s" width="%s" />' % (session_img, img_width))
else:
    print(' <P>No session data for period %s to %s </P>' % (start,end))

print(' </BODY>')
```

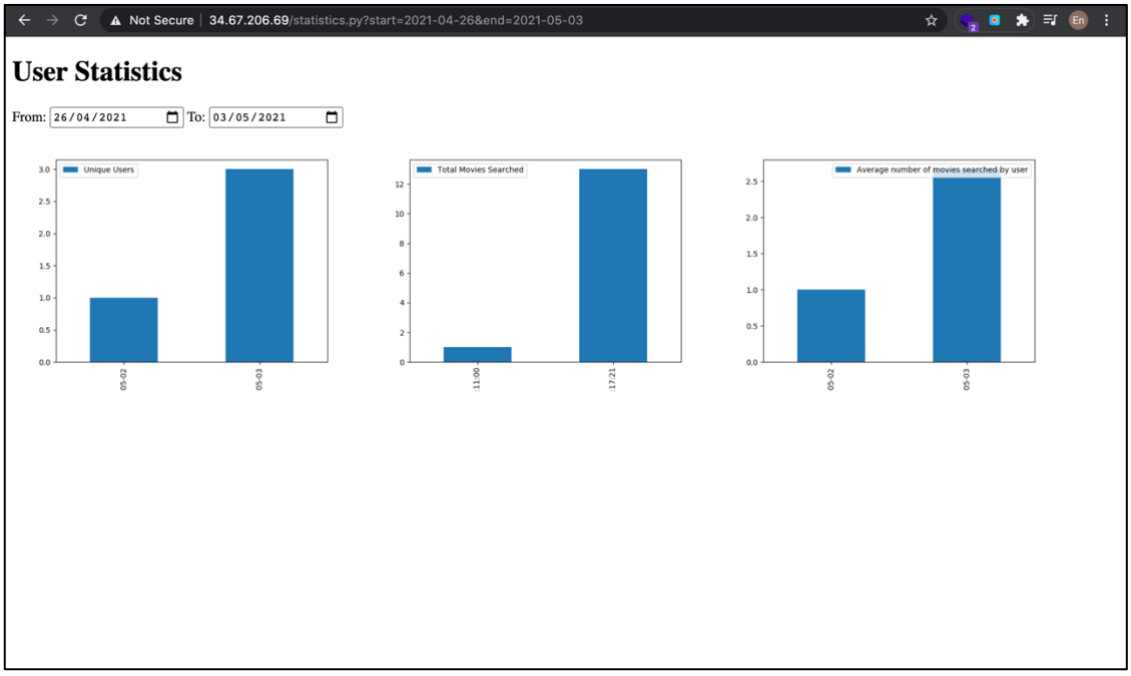
print('</HTML>')

Explanation:

The code for statistics.py has four main challenges: from the query, get the dates on which the user wants to view the data, get the relevant data from the movies.sql database, visualize the data through graphs using pandas, and create the html. First, we import the necessary libraries and get the beginning and end dates from the form. If there is are no chosen dates in the query, the dates will be set to today's date by default. Moving on, we connect to the database movies. Now we will need to make three graphs: one representing the number of unique users who have visited the movies.py on the selected dates, one representing the total number of movies searched per day, and one showing the average number of unique movies searched per unique user. Now we create 3 SQL queries necessary to get the data relevant to each graph. Since in the first graph we need the number of unique users we only select the count (distinct user_id) from User_search since we do not want to register the same user more than once. Then in the next SQL query, we select the count of movie_id from user_searches to get the total number of movie lookups. In this case, we don't use "distinct" since searching for the same movie twice or more count as two or more movie lookups. In the third query, we use simple math to get the average number of unique movies searched by unique user using a fraction (total no. of movie id)/(total no. of users). After having the data we need we create the graphs through pandas and eba3420.plot_to_img_str() function. We set the column "Date" in the x-axis for all three graphs and the data required to visualize on the y-axis. And lastly, we create the html form in order for our code to be displayed on the webpage.

Below you will find a screenshot of the page statistics.py.

Screenshot:



Bonus (Normalization):

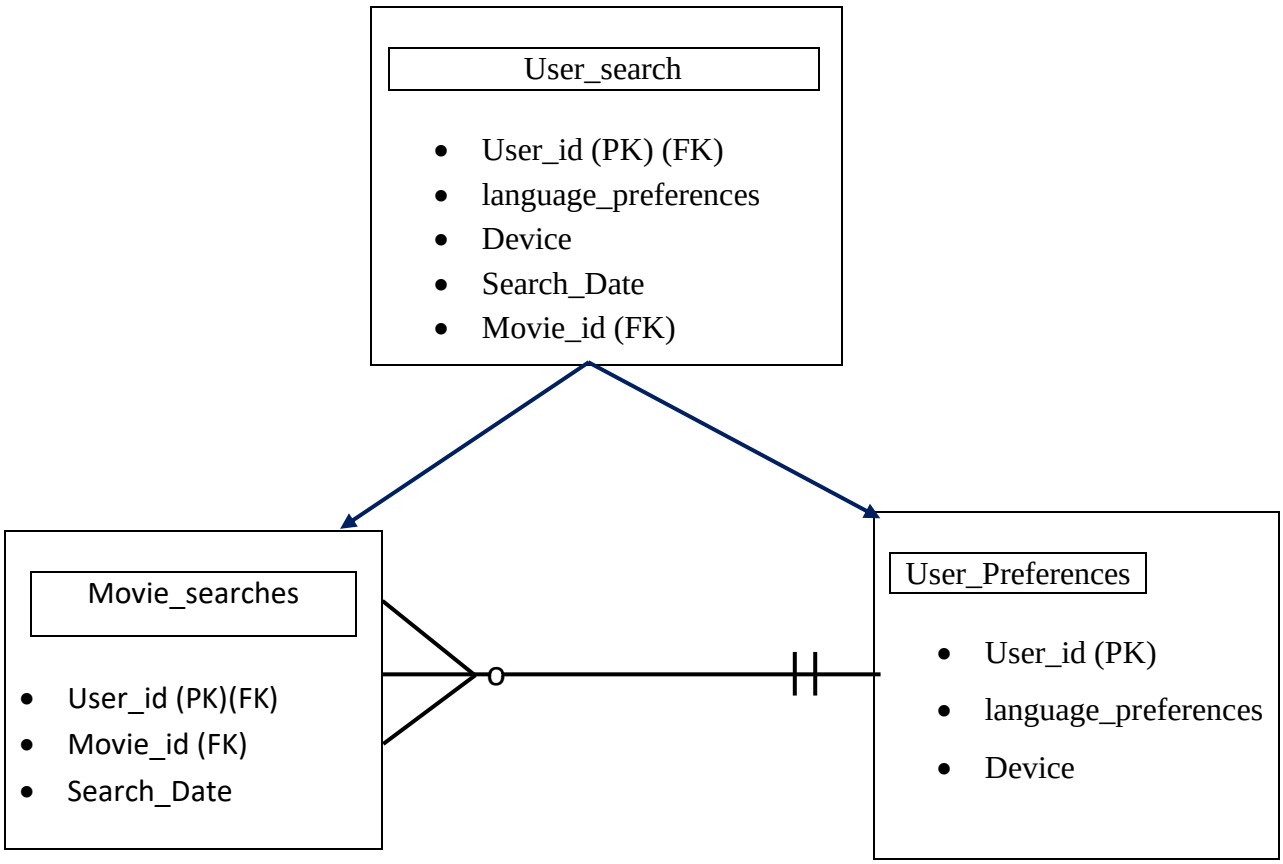
Below we will go through a normalization check for the two tables we created in Part 2: "Unique_user" and "User_search". Normalization is the process of structuring the database in order to reduce/minimize data redundancy. Below, we will check if the tables conform the 1NF, 2NF & 3NF. Our table, Unique_user has only two attributes with the non-key attribute being directly dependent on the primary key, hence it satisfies the condition for the 1NF, 2NF & 3NF. However, the table "User_searches" is not normalized and below we will go through the normalization process for it.

1st Normal Form:

1NF requires each field in the tables to be atomic, meaning that there is not more than 1 value per field. Out table satisfied this condition, hence it is in the 1st Normal Form.

2nd Normal Form:

The 2NF requires a table to not have partial dependents or to not have attributes that don't belong to any key. Our Unique_user table satisfies this condition. However, the table called User_search could be split into two tables. One table called User_preferences would include the user_id, language_preferences, device and Search_date. The other table called "Movie_searches" would consist of the Movie_id and Search_Date.



3rd Normal Form:

Now we have created two tables. Under the table “User_preferences” all attributes are directly dependent on the PK. In this table both “Device” and the “language_preferences” are directly dependent on the User_id. Likewise, under the “Movies_searches”, both movie_id and search_date are directly dependent on the User_id (in simple English; both the movie and the date this movie is searched are dependent on the user who searches this movie). Hence, all our tables are in 3NF.

Note:

These two tables we created above have a one-to-many relations. A user with the selected language and device can search many movies, however for each individual search, there can be only one user with specific language and device.