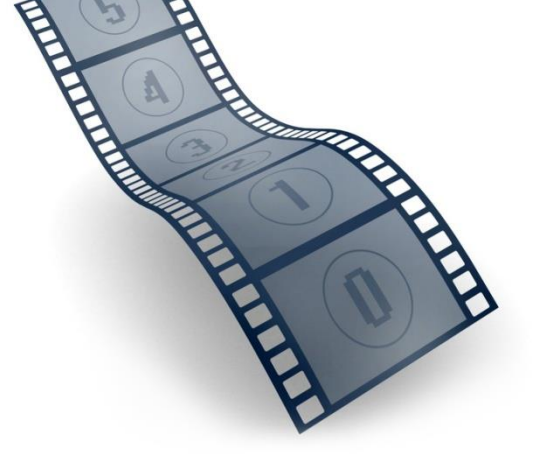
Code structure and general flow



Our code is divided to two files.

The first one is the creation of the tables and inserting the data from the APIs. We called this one API_DATA_RETREIVE.py, as you asked. We asked in the forum if it's ok to have one file for both creating tables and inserting the data and you said yes, so there is no other file for creating tables.

The second one is the queries.

In each file there are functions for each table creation and each query in order to maintain order.

- API_DATA_RETREIVE.py:
  - first of all we have the necessary imports.
  - Then we have the connection and creation of the cursor.

```
cnx = pymysql.connect(host='localhost',
                      port=3305,
                      user='DbMysql35',
                      password='DbMysql35',
                      database='DbMysql35'
                     )

cursor = cnx.cursor()
```

  - Then we have some information that we need for the API itself:

```
## Info for the API ##

querystring = {"page_size":"50"}
headers = {
    'x-rapidapi-host': "data-imdb1.p.rapidapi.com",
    'x-rapidapi-key': "3e4bed8bd9mshd7585f537edbe30p166b99jsn800d1097fd15"
    }
```

  - Then we have all the creations of tables, separated by function, and another 3 functions that will help us get the ids we need:

```
def find_movies_ids_from_2013_till_now():...
def find_actors__ids(ids_of_movies):...
def find_directors__ids(ids_of_movies):...
def create_movie_data_table(ids_of_movies):...
def create_movie_description_table(ids_of_movies):...
def create_awards_table_and_movies_awards_table(ids_of_movies):...
def create_cast_table(ids_of_movies):...
def create_director_data_table(directors_id):...
def create_actor_data_table(actors_ids):...
def create_person_data_table(actor_ids, director_ids):...
```

  - Find_movies_ids_from_2013_till_now(): - helps us retrieve all the ids that we need for the films that we want, using the api we chose. We

use json for that. We chose 50 movies for each year from 2013 till now.

```
# now we'll get the ids of movies ##

ids = []
for i in range(2013, 2022):

    url = "https://data-imdb1.p.rapidapi.com/movie/byYear/%d/" % (i)
    response = requests.request("GET", url, headers=headers, params=querystring)
    json_data = json.loads(response.text)
    print(json_data)

    for j in range(50): ## we want 50 movies per year
        ids.append(json_data["results"][j]["imdb_id"])

return ids
```

- Find_actors_ids(ids_of_movies) and find_directors_ids(ids_of_movies) have the same structure as Find_movies_ids_from_2013_till_now(),

we are just checking the role of the person before we enter it to our data (in the api we have a cast per movie_id).

- The functions of creation of the tables and retrieving the data have the same structure:
    - Creation of the table (primary key, foreign key etc.)
    - Creation of the json_data with the API
    - Sql insertion query
    - Constraints (for example if we want movies with rating that is higher than 6).
    - Execution of the sql query with the right data that we retrieved
    - Creating index (if needed).
- Of course retrieving the data is a bit different for each table, because we use different APIs and different endpoints, and each one has a bit different data structure.

  Lets see an example:

  We'll look at movie_data creation.

- Here is the creation of the table itself. It's in 2 rows just here, for your comfort. as you can see, movie_id is the primary key.

```
cursor.execute("CREATE TABLE IF NOT EXISTS movie_data (movie_id VARCHAR(255) PRIMARY KEY,
```

```
title VARCHAR(255), genre VARCHAR(255), year VARCHAR(255), length VARCHAR(255), rating VARCHAR(255))")
```

- Here is the creation of the json from the API

```
for id in ids:
    print(id)
    url = "https://data-imdb1.p.rapidapi.com/movie/id/%s/" % (id)
    response = requests.request("GET", url, headers=headers)
    json_data = json.loads(response.text)
```

- And the sql insertion query:

```
    sql = "INSERT INTO movie_data (movie_id, title, genre, year, length, rating) VALUES (%s, %s, %s, %s, %s, %s)"
```

- For this table we have constraint that we wanted to have, insert only movies with rating higher than 6 (only the best movies).

```
if json_data["results"]["rating"] > 6: ## We only want good movies, with rating more than 6.
    val = (id,
           json_data["results"]["title"],
           json_data["results"]["gen"][0]["genre"],
           json_data["results"]["year"],
           json_data["results"]["movie_length"],
           json_data["results"]["rating"])
```

- And then the execution:

```
    try:
        cursor.execute(sql, val)
        cnx.commit()
    except:
        cnx.rollback()
```

- And then the creation of the index:

```
try:
    # SQL Statement to create an index
    sqlCreateIndex = """CREATE INDEX rating_ind ON movie_data(rating);"""
    cursor.execute(sqlCreateIndex)
except:
    print("Index rating is already exists.")
```

- And after all these functions, we have the main function, so you can run each function that you want.

  We have 3 values that we need for the functions- ids_of_movies, actors_ids, directors_ids. Each function needs different value.

  And then we have all the functions for the creation of the tables. If you don't want to run all of them, please put # before the function. Notice what value the function needs.

```
def main():
    ### Variables for the different creation of tables.

    ids_of_movies = find_movies_ids_from_2013_till_now()
    director_ids = find_directors__ids(ids_of_movies)
    actor_ids = find_actors__ids(ids_of_movies)

    ### Creation of all tables

    create_movie_data_table(ids_of_movies)
    #create_actor_data_table(actor_ids)
    #create_cast_table(ids_of_movies)
    #create_director_data_table(director_ids)
    create_person_data_table(director_ids,actor_ids)
    #create_movie_description_table(ids_of_movies)
    #create_awards_table_and_movies_awards_table(ids_of_movies)
```

- And then we have the closing of cnx and cursor:

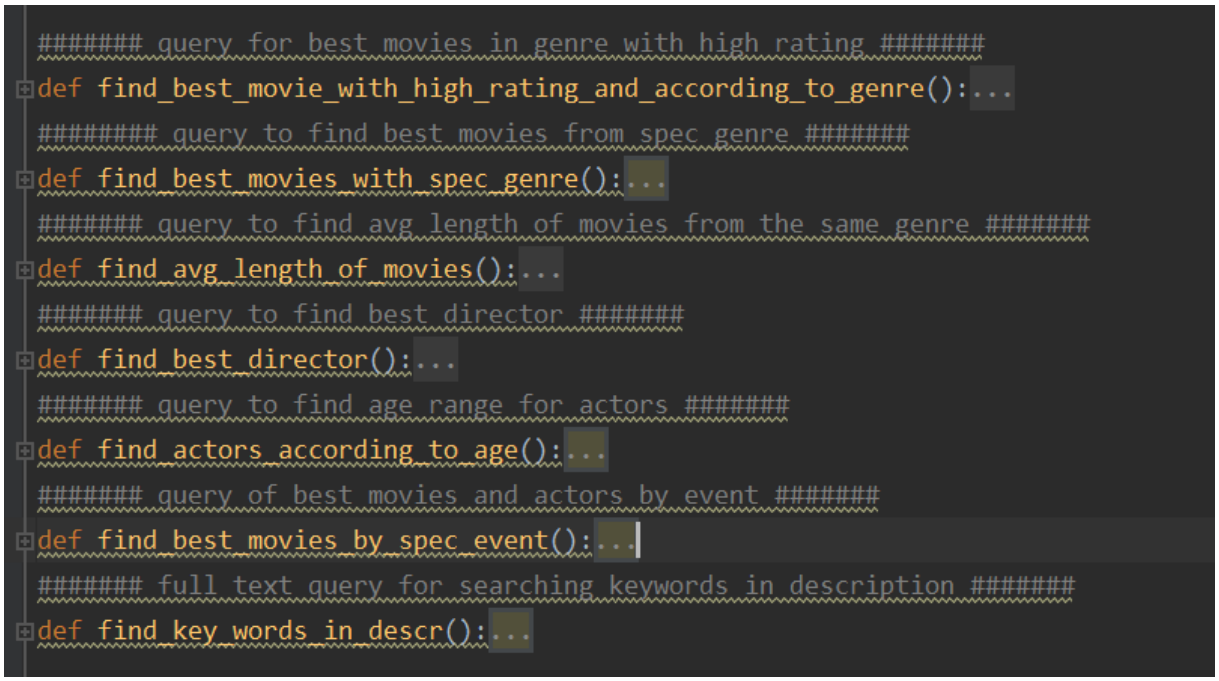
```
    cursor.close()
    cnx.close()
```

- queries.py:
  - first of all we have the necessary imports.
  - Then we have the connection and creation of the cursor.

```
cnx = pymysql.connect(host='localhost',
                      port=3305,
                      user='DbMysql35',
                      password='DbMysql35',
                      database='DbMysql35'
                     )

cursor = cnx.cursor()
```

- then we have the queries, each one in a separate function:



- There are some queries that have input from the user. In order to do that and to maintain order, after you run a function, for example

"find_actors_according_to_age():", we will guide you through the terminal.

Let's make an example:

We will run **find_actors_according_to_age()**

Then we will see this on the terminal:

```
Please choose a number from 1 to 120 for minimum age:30
Please choose a number from 1 to 120 for maximum age:60
```

The green numbers are our input.

And then we will give you the full list of actors from age 30 to 60.

- The main structure for each query is:
    - User input (if needed) and validation of it
    - Sql query itself
    - Execution
    - Fetch
- Of course each query has a little bit different structure according for what we need.

  For example, in **find_actors_according_to_age()** we'll need an input from the user.

  So first of all we'll get that input and we'll validate it.

```
success = 0
while success ==0:
    min_age = float(input("Please choose a number from 1 to 120 for minimum age:"))
    max_age = float(input("Please choose a number from 1 to 120 for maximum age:"))

    if min_age < 1 or min_age > 120:
        print("Min age is not valid")
        continue
    if max_age < 1 or max_age > 120:
        print("Max age is not valid")
        continue
    else:
        success = 1
```

We validate that the user gave us a reasonable minimum age and maximum age. If not, we'll continue asking for them (success won't be 1 and the while loop will continue).

After that, we have the SQL query itself:

```
query_of_age = ("""SELECT DbMysql35.actor_data.full_name
FROM DbMysql35.actor_data AS actor_data
WHERE actor_data.age BETWEEN %s AND %s
AND actor_data.actor_id IN (SELECT DbMysql35.actor_data.actor_id
                            FROM DbMysql35.actor_data
                            INNER JOIN DbMysql35.cast ON DbMysql35.actor_data.actor_id = DbMysql35.cast.person_id
                            INNER JOIN DbMysql35.movies_awards ON DbMysql35.cast.movie_id =  DbMysql35.movies_awards.movie_id
                            INNER JOIN DbMysql35.awards ON DbMysql35.movies_awards.award_id = DbMysql35.awards.award_id
                            WHERE DbMysql35.awards.nomineeORwinner = "Winner"
                            GROUP BY DbMysql35.actor_data.actor_id
                            HAVING COUNT(*)>=3)

ORDER BY DbMysql35.actor_data.full_name""")
```

And then we have the execution and the fetch:

```
cursor.execute(query_of_age, (min_age, max_age))
cnx.commit()


rows = cursor.fetchall()
for row in rows:
    print(row)
```

- And then we have the main function that let you easily run each query. If you don't want to run them all together, please put # before the function you don't want to run.

```
def main():
    print("*********")
    find_best_movie_with_high_rating_and_according_to_genre()
    print("*********")
    find_best_movies_with_spec_genre()
    print("*********")
    find_avg_length_of_movies()
    print("*********")
    find_best_director()
    print("*********")
    find_actors_according_to_age()
    print("*********")
    find_best_movies_by_spec_event()
    print("*********")
    find_key_words_in_descr()
    print("*********")
```

- And of course ,in the end we are closing cursor and cnx.

```
cursor.close()
cnx.close()
```

So now that we know the structure of the code, let's understand what is the flow.

The tables with the data are ready to use. All the right data is in them (for example details about the movie, about the actor etc.). The user doesn't know about them and doesn’t have access to them directly. Only through queries (in the queries.py).

Now my producer will open the application and will choose a query according to what he needs. We recommend to use all of our queries in order to make the best film. As we described, if we don’t want to use a query, we need to put # before the right function in the code.

So first of all he can choose between movie analysis, actors analysis and production analysis (as you can see in the user manual).

In each one, there are queries. So when he chooses a query, our code begins to work.

If we need a user input, we will print it out and the producer will give us all the answers we need in order to give him the best information. Everything is explained to the user in order for him to give us the valid answers we need (For example, if we need an int for the age, and he will accidently give us input of "yay", we won't accept it).

So after our SQL query will retrieve the data we need, we will print it to the user's screen.

Of course he can run the queries as many times as he wants, with different inputs. Just click on the right query and our code will do the rest.