Middle East Technical University | Department of Computer Engineering

CENG 352

Database Management Systems

Spring 2019-2020
Project 3 : Transactions

Due date: May 17, 2020, Sunday, 23:59

1 Project Description

In this project you are asked to develop a simple video streaming service application (e.g. Netflix, Amazon Prime Video, etc.). This is a typical application using a database. You will import data to a PostgreSQL database. Database details will be explained later.

Your service provides video streams to customers. You have videos of all the movies in the IMDB database and you provide this content to your customers. Firstly, you will construct the database and import the data for this database. After those, you will create an application to allow customers to use the service.

** There is **an important restriction** of this application:

For each customer, there exists a maximum number of parallel sessions allowed. This number is determined by the plan to which the customer is subscribed. Each customer is subscribed to exactly one plan. Once the number of sessions of a customer reaches the maximum, you will deny following sign in requests (sign in command) until some sessions are terminated by signing out (sign out command).

2 Database

2.1 Creating the database

The database name will be **"ceng352_mp3_data"** for this mini project. You will have 2 steps while creating the database:

- Create tables and insert sample data by running SQL queries in **"construct_db.sql"**.
- Import IMDB data from .csv files under **data** directory. Import the data as-is, without doing any kind of data processing or cleaning. **Consider column delimiter as semicolon (;),**

consider quote character as quote (”) and fill empty values with NULL while importing IMDB data to tables.

After these steps, you will get following tables in the database:

```
Customer (customer_id serial, email text, password text,
          first_name text, last_name text, session_count int, plan_id int)

Plan (plan_id serial, plan_name text, resolution text,
      max_parallel_sessions int, monthly_fee int)

Actors (actor_id text, actor_name text, birth_year int,
        death_year int, gender text)
Directors (director_id text, director_name text)
Directed (movie_id text, director_id text)
Genres (movie_id text, genre_name text)
Casts (movie_id text, actor_id text, "characters" text)
Movies (movie_id text, title text, movie_year int, rating decimal, votes int)
Watched (customer_id serial, movie_id text)
```

You should keep following information in your mind:

- **Customer** represents the customers of the service. **session_count** is the number of active sessions of customers. This number is incremented when a customer signs in to the service and decremented when he/she signs out.

  For example, a sample customer can be *email: “cj@mp3.com”, password: “pass123”, first_name: “Carl”, last_name: “Johnson”, session_count: 0, plan_id: 1.*



Table 1: Possible customers of your platform: CJ, Big Smoke, Ryder

- **Plan** represents the available plans to subscribe for customers. You should insert at least 3 plans with different **max_parallel_sessions** values. **“max_parallel_sessions”** is a value to allow customers to have **at most N** connections at any moment.
  If max_parallel_sessions = 2, then the customer who is subscribed to this plan can connect to the platform with at most 2 devices at the same time.

Every customer must have a plan and one customer is dedicated to one plan only. You are free to invent your own plans. For example, a sample plan can be *name: "Basic", resolution: "720P", max_parallel_sessions: 2, monthly_fee: 30.*

- One movie can have multiple genres in **Genres** table.
- One actor/actress can act in multiple movies in **Casts** table.
- One customer can watch multiple movies in **Watched** table.

3 The Software

For helpful tutorials and links, check these websites: **link 1**, **link 2**, **link 3**, **link 4**.

After the database is created, you will implement a simple program for the customers to connect to the platform. For simulating multiple device connections, you can open multiple terminal windows and sign in from those terminals.

For simplicity, the platform will be a command line-like software written in Python3. You will use **psycopg2** to do PostgreSQL tasks like insert, update, read etc. These are the source files inside **"source"** directory:

- **main.py:** This file is for retrieving inputs, processing requests and sending response messages back to the customer. **main()** function in this file acts like a service layer for the software. You should not modify this file, you will run the software like:

  ```
  >_ python main.py
  ```

  only. How this software works will be explained shortly.

- **mp3.py: You will ONLY modify this file.** You will implement the functions in this file and make customers available to sign in, sign out and read contents from the database. You will return success message or error messages, depending on the situation.
- **validators.py:** This file is for validating commands. You don't have to worry about validation, it is written for you and you can try entering incorrect commands to break the program.
- **customer.py:** This file contains the Customer class, which will be used to store currently authenticated customer information.
- **messages.py:** You should **ONLY** use and return messages defined in this file, to get points from this assignment.
- **database.cfg:** This file contains database configurations. Change them with your configurations to connect the database on your local machine.

When the program starts, list of commands are shown and the program waits for command input:

```
_> python main.py

*** Please enter one of the following commands ***
> help
> sign_up <email> <password> <first_name> <last_name> <plan_id>
> sign_in <email> <password>
> sign_out
> show_plans
> show_subscription
> subscribe <plan_id>
> watched_movies <movie_id_1> <movie_id_2> <movie_id_3> ... <movie_id_n>
> search_for_movies <keyword_1> <keyword_2> <keyword_3> ... <keyword_n>
> suggest_movies
> quit
ANONYMOUS >
```

At first, there is no signed in customer. Customer is shown as "ANONYMOUS" and you can only call **"help", "sign_up", "sign_in", "quit"** commands with anonymous customer. "help" command will remind you what are the available commands provided by this software.

If you implement "sign_in" command correctly and sign in with an existing customer information, the look will change to authenticated customer like below:

```
ANONYMOUS > sign_in cj@mp3.com pass123
OK
Carl Johnson (cj@mp3.com) >
```

Authenticated customers can use **"sign_out", "show_plans", "show_subscription", "subscribe", "watched_movies", "search_for_movies", "suggest_movies", "quit"** commands. These commands are available to authenticated customers only.

4 Tasks

You should find

```
#TODO: Implement this function
```

comments in the code and replace them with your own implementations.

4.1 Sign Up

```
>_ sign_up <email> <password> <first name> <last name> <plan id>
```

You need to implement **sign_up()** function inside **mp3.py**. This is the command to create a new customer. The anonymous customer enters customer information (email, password, first name, last name) and id of the plan that the new customer will be subscribed to (plan id). You should create a new customer with provided information in the database.

When you complete the implementation, the software should give output like this:

```
ANONYMOUS > sign_up wick@mp3.com pass123 John Wick 1
OK
```

If a customer with same e-mail address exists before, or the program gets any kind of exception during the execution, you should give an error message like this (assuming you have already a customer with e-mail wick@mp3.com):

```
ANONYMOUS > sign_up wick@mp3.com pass123 John Wick 1
ERROR: Can not execute the given command.
```

4.2 Sign In

```
>_ sign_in <email> <password>
```

You need to implement **sign_in()** function inside **mp3.py**. This is the command for a customer to sign in to the service. The customer types in customer credentials (email and password). You should implement required authentication and session management logic using your customer database.

Remember to increment *session_count* inside Customer table for the customer. Also check whether the customer is out of sessions or not by looking at *max_parallel_sessions* of the customer's plan.

Successful sign in operation should look like this:

```
ANONYMOUS > sign_in cj@mp3.com pass123
OK
Carl Johnson (cj@mp3.com) >
```

Failed sign in operation should look like this:

```
ANONYMOUS > sign_in cj@mp3.com pass123456
ERROR: E-mail or password is wrong.
ANONYMOUS >
```

If **session_count** >= **max_parallel_sessions**, then the output should look like this:

```
ANONYMOUS > sign_in cj@mp3.com pass123
ERROR: You are out of sessions for signing in.
ANONYMOUS >
```

***** For simulating multiple device connections, you can open multiple terminal windows and sign in from those terminals.**

4.3 Sign Out

```
>_ sign_out
```

You need to implement **sign_out()** function inside **mp3.py**. This is the command for an authenticated customer to sign out from the service. You should implement required authentication and session management logic using your customer database.

Remember to decrement *session_count* inside Customer table for the customer. Also check whether the customer's *session_count* can be at least 0.

Successful sign out operation should look like this:

```
Carl Johnson (cj@mp3.com) > sign_out
OK
ANONYMOUS >
```

Failed sign out operation should look like this:

```
Carl Johnson (cj@mp3.com) > sign_out
ERROR: Can not execute the given command.
Carl Johnson (cj@mp3.com) >
```

4.4 Show Plans

```
>_ show_plans
```

You need to implement **show_plans()** function inside **mp3.py**. This is the command to get list of all available plans to subscribe. This command does not have any parameters. You should print all columns of the plans in the database (e.g. plan id, name, etc.). Follow the pattern below while printing columns:

When there is an authenticated customer, show_plans operation should look like this:

```
Carl Johnson (cj@mp3.com) > show_plans
#|Name|Resolution|Max Sessions|Monthly Fee
1|Basic|720P|2|30
2|Advanced|1080P|4|50
3|Premium|4K|10|90
```

4.5 Show Subscription

```
>_ show_subscription
```

You need to implement **show_subscription()** function inside **mp3.py**. This is the command to get the details of the plan to which the authenticated customer is subscribed. This command does not have any parameters. Printing should be similar with the show_plans command.

When there is an authenticated customer, show_subscription operation should look like this:

```
Carl Johnson (cj@mp3.com) > show_subscription
#|Name|Resolution|Max Sessions|Monthly Fee
1|Basic|720P|2|30
```

4.6 Subscribe To A New Plan

```
>_ subscribe <new plan's id>
```

You need to implement **subscribe()** function inside **mp3.py**. This is the command for authenticated customer to subscribe to another plan. You must ensure that customers will not subscribe to a new plan with less parallel sessions allowed. You should update the subscription information for the authenticated customer on the customer database.

Assume that CJ has a plan with id=1 and max_parallel_sessions=2 and wants to subscribe to a plan with id=2 and max_parallel_sessions=4. This operation can be done since **old_max_parallel_sessions** <= **new_max_parallel_sessions**. If max_parallel_sessions values are same, you should also allow that operation too.

```
Carl Johnson (cj@mp3.com) > show_subscription
#|Name|Resolution|Max Sessions|Monthly Fee
1|Basic|720P|2|30
Carl Johnson (cj@mp3.com) > subscribe 2
OK
Carl Johnson (cj@mp3.com) > show_subscription
#|Name|Resolution|Max Sessions|Monthly Fee
2|Advanced|1080P|4|50
```

However, you must reject the command if **old_max_parallel_sessions** > **new_max_parallel_sessions**. When we try to change CJ's plan with the old plan id, following happens:

```
Carl Johnson (cj@mp3.com) > subscribe 1
ERROR: New plan's max parallel sessions must be greater than or equal to current
plan's max parallel sessions.
Carl Johnson (cj@mp3.com) > show_subscription
#|Name|Resolution|Max Sessions|Monthly Fee
2|Advanced|1080P|4|50
```

If there is no plan with given plan id, reject with not found message:

```
Carl Johnson (cj@mp3.com) > subscribe 99
ERROR: Plan is not found.
```

4.7 Watch Movies

```
>_ watched_movies <movie 1 id> <movie 2 id> <movie 3 id> ... <movie N id>
```

You need to implement **watched_movies()** function inside **mp3.py**. This is the command to save the fact that the authenticated customer has just watched a movie or movies. The customer types in ids of the movies (from the IMDB database) that the customer watched. You should save this information to **watched** table in the database.

Let's say CJ watched "The Dark Knight", "The Lord of the Rings: The Fellowship of the Ring", "Per qualche dollaro in piu". Below command is for saving this information to the database:

```
Carl Johnson (cj@mp3.com) > watched_movies "tt0468569" "tt0120737" "tt0059578"
OK
```

There are rules about this command:

1. You can send 1, 2, 3, ... movie ids as parameters to this command. Your implementation should handle multiple movie ids as arguments.
2. You should save **ALL** customer id-movie id relationships to the watched table, or **NONE** of them in case of an error during saving a customer id-movie id relationship to the watched table.
3. You should NOT add same customer id-movie id relationship to the watched table again, when the customer decides to watch a movie one more time. You don't need to do any updates in this case, you can consider it as successful operation.

An example for rule 2:

```
Carl Johnson (cj@mp3.com) > watched_movies "tt0468569" "tt0120737" "test"
ERROR: Can not execute the given command.
```

An example for rule 3, CJ watched "The Dark Knight", "The Lord of the Rings: The Fellowship of the Ring" again and watched "Good Will Hunting":

```
Carl Johnson (cj@mp3.com) > watched_movies "tt0468569" "tt0120737" "tt0119217"
OK
```

4.8 Search For Movies

```
>_ search_for_movies <keyword 1> <keyword 2> ... <keyword N>
```

You need to implement **search_for_movies()** function inside **mp3.py**. You should return list of movies whose title matches with provided string, **in CASE-INSENSITIVE manner**. **"search_text"** argument of search_for_movies() function is actually merged version of keywords you provided, so you don't need to deal with parsing words etc.

You should print each column of the movie and whether the movie was seen by the authenticated customer before or not. Movies should be ordered by their movie ids. You should follow the print format in the example below:

```
Carl Johnson (cj@mp3.com) > search_for_movies the dark knight
Id|Title|Year|Rating|Votes|Watched
"tt0147505"|"Sinbad: The Battle of the Dark Knights"|1998|2.2|149|0
"tt0468569"|"The Dark Knight"|2008|9.0|2021237|1
"tt1345836"|"The Dark Knight Rises"|2012|8.4|1362116|0
"tt3153806"|"Masterpiece: Frank Millers The Dark Knight Returns"|2013|7.8|28|0
"tt4430982"|"Batman: The Dark Knight Beyond"|0|0.0|0|0
"tt4494606"|"The Dark Knight: Not So Serious"|2009|0.0|0|0
"tt4498364"|"The Dark Knight: Knightfall - Part One"|2014|0.0|0|0
```

```
"tt4504426"|"The Dark Knight: Knightfall - Part Two"|2014|0.0|0|0
"tt4504908"|"The Dark Knight: Knightfall - Part Three"|2014|0.0|0|0
"tt4653714"|"The Dark Knight Falls"|2015|5.4|8|0
"tt6274696"|"The Dark Knight Returns: An Epic Fan Film"|2016|6.7|38|0
```

Search texts "the daRk knIght" or "THE DARK KNIGHT" should return the same result for this command.

4.9 Suggest Movies

```
>_ suggest_movies
```

You need to implement **suggest_movies()** function inside **mp3.py**. This is the command for authenticated customer to get movie suggestions from the service. This command does not have any parameters.

You will follow 3 steps to do recommendations. For each step, find movies to recommend and merge all these movies found in steps at the end. If there are movies recommended in multiple steps, it is OK, just remove duplications and return each movie only once. You should follow these steps to make suggestions:

1. **STEP 1:** Find genres of the movies that the authenticated customer watched. Let's call this genre group as G. For each genre in G, find movies (not watched by the authenticated customer) that has the most number of votes. You can get same movie for multiple genres, this is fine and recommend same movie.

   Let's assume that CJ watched

   - "The Dark Knight" (Crime, Action, Drama)
   - "The Lord of the Rings: The Fellowship of the Ring" (Adventure, Drama, Fantasy)
   - "The Shawshank Redemption" (Drama)
   - "I Am Legend" (Horror, Sci-Fi, Drama)

   Genres of CJ will be "Drama", "Action", "Fantasy", "Sci-Fi", "Horror", "Crime", "Adventure".

	genre_name	max
1	"Drama"	2,054,728
2	"Action"	2,021,237
3	"Fantasy"	1,479,525
4	"Sci-Fi"	1,800,667
5	"Horror"	763,646
6	"Crime"	2,021,237
7	"Adventure"	1,800,667

Figure 1: Max votes in above genres

	genre_name	max
1	"Drama"	1,644,366
2	"Action"	1,800,667
3	"Fantasy"	1,462,879
4	"Sci-Fi"	1,800,667
5	"Horror"	763,646
6	"Crime"	1,605,701
7	"Adventure"	1,800,667

Figure 2: Max votes in above genres,
when customer's watched movies are discarded

Then following movies should be recommended:

- "The Shining" (Horror)
- "Pulp Fiction" (Crime)
- "Fight Club" (Drama)
- "The Lord of the Rings: The Return of the King" (Fantasy)
- "Inception" (Action, Adventure, Sci-Fi)

2. **STEP 2:** Find **top 10** movies (not watched by the authenticated customer) with most number of votes and highest rating, directed after (>=) 2010. These movies should be sorted by votes in descending order and by rating in descending order.

	movie_id	title	movie_year	rating	votes
1	"tt1375666"	"Inception"	2,010	8.8	1,800,667
2	"tt1345836"	"The Dark Knight Rises"	2,012	8.4	1,362,116
3	"tt0816692"	"Interstellar"	2,014	8.6	1,261,057
4	"tt1853728"	"Django Unchained"	2,012	8.4	1,185,053
5	"tt0848228"	"The Avengers"	2,012	8.1	1,150,087
6	"tt0993846"	"The Wolf of Wall Street"	2,013	8.2	1,010,614
7	"tt1130884"	"Shutter Island"	2,010	8.1	984,509
8	"tt2015381"	"Guardians of the Galaxy"	2,014	8.1	910,209
9	"tt1431045"	"Deadpool"	2,016	8	797,470
10	"tt1392170"	"The Hunger Games"	2,012	7.2	786,382

Figure 3: Recommended movies - step 2

3. **STEP 3:** Find **top 10** movies (not watched by the authenticated customer) that have votes higher than the average number of votes of movies watched by the authenticated customer.

	movie_id	title	movie_year	rating	votes	genre_name
1	"tt0111161"	"The Shawshank Redemption"	1,994	9.3	2,054,728	"Drama"
2	"tt0468569"	"The Dark Knight"	2,008	9	2,021,237	"Crime"
3	"tt0468569"	"The Dark Knight"	2,008	9	2,021,237	"Action"
4	"tt0468569"	"The Dark Knight"	2,008	9	2,021,237	"Drama"
5	"tt0120737"	"The Lord of the Rings: The Fellowship of the Ring"	2,001	8.8	1,479,525	"Adventure"
6	"tt0120737"	"The Lord of the Rings: The Fellowship of the Ring"	2,001	8.8	1,479,525	"Drama"
7	"tt0120737"	"The Lord of the Rings: The Fellowship of the Ring"	2,001	8.8	1,479,525	"Fantasy"
8	"tt0480249"	"I Am Legend"	2,007	7.2	624,479	"Horror"
9	"tt0480249"	"I Am Legend"	2,007	7.2	624,479	"Sci-Fi"
10	"tt0480249"	"I Am Legend"	2,007	7.2	624,479	"Drama"

Figure 4: CJ's movies

The average is 1.544.992,25 votes. The movies with votes higher than this value are:

	movie_id	title	movie_year	rating	votes
1	"tt1375666"	"Inception"	2,010	8.8	1,800,667
2	"tt0137523"	"Fight Club"	1,999	8.8	1,644,366
3	"tt0110912"	"Pulp Fiction"	1,994	8.9	1,605,701
4	"tt0109830"	"Forrest Gump"	1,994	8.8	1,572,447

Figure 5: Recommended movies - step 3

FINAL RESULT: At the end, you need to merge movie lists found in each step and get rid of the duplications. You can do this within a single SQL query or in mp3.py with 3 different queries, it does not matter. Make sure that you follow the printing format below and order movies by their movie ids. Here is the result for suggest_movies operation:

```
Carl Johnson (cj@mp3.com) > suggest_movies
Id|Title|Year|Rating|Votes
"tt0081505"|"The Shining"|1980|8.4|763646
"tt0109830"|"Forrest Gump"|1994|8.8|1572447
"tt0110912"|"Pulp Fiction"|1994|8.9|1605701
"tt0137523"|"Fight Club"|1999|8.8|1644366
"tt0167260"|"The Lord of the Rings: The Return of the King"|2003|8.9|1462879
"tt0816692"|"Interstellar"|2014|8.6|1261057
"tt0848228"|"The Avengers"|2012|8.1|1150087
"tt0993846"|"The Wolf of Wall Street"|2013|8.2|1010614
"tt1130884"|"Shutter Island"|2010|8.1|984509
"tt1345836"|"The Dark Knight Rises"|2012|8.4|1362116
"tt1375666"|"Inception"|2010|8.8|1800667
"tt1392170"|"The Hunger Games"|2012|7.2|786382
"tt1431045"|"Deadpool"|2016|8.0|797470
"tt1853728"|"Django Unchained"|2012|8.4|1185053
"tt2015381"|"Guardians of the Galaxy"|2014|8.1|910209
```

4.10 Quit

```
>_ quit
```

You need to implement **quit()** function inside **mp3.py**. This is the command to quit the application. **Remember to sign out before quitting if there is an authenticated customer.**

5 Regulations

1. **Implementation:** Implement mp3.py only. If you think you need helper functions for the tasks, please write them to mp3.py. **You should NOT update any other files in the source directory.**
   This is for constructing a common language while asking questions, talking about solutions and providing black-box compatible implementations for grading. You may think you can find another better solutions when you read the implementation structure, it is possible, but we want you to follow the given structure for the reasons mentioned above.

2. **Response messages:** Check messages.py and comments on the functions for response messages to be used. **Different messages for responses are NOT ALLOWED.**

3. **Submission:** Submission will be done via ODTUClass. Please remember that **late submission is allowed 12 days for all projects.** You can distribute these 12 days to any project your want. You should put your **mp3.py** implementation inside a .zip file with following name:

   ```
   e1234567_project3.zip
       -> mp3.py
   ```

   Where you should replace "1234567" with your own student number. Please make sure that the .zip file doesn't contain any subdirectories, it should only contain mp3.py.

4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible clarifications on a daily basis.