Middle East Technical University

Department of Computer Engineering

CENG 352
Database Management Systems
Spring 2020
Project 2

1 Introduction

In this project you are asked to develop a simple database system using the Yelp database that you worked on in Project 1. The Yelp database contains information about businesses and user assessments about them. Now, assume that Yelp wants to extend their functionality by allowing different kinds of memberships for their users.

There will be three classes of users, Non Authorized users, Free users and Premium users. All not signed in users (i.e not registered to the system) is considered as 'Non Authorized' users. All existing users (in the the original dataset) are considered as 'Free' users. Premium users have subtypes (can be any number of type) which indicates the functionality that our system provides to the individuals. The Premium users can perform all the actions as Free users can do. Also, they can enjoy suggestions from the system and some coupons or discounts at certain locations.

Each user is subscribed to exactly one membership. For each user, there is a maximum number of parallel sessions allowed. This number is determined by the membership to which the user is subscribed. Once the number of sessions of a user reaches the maximum, you will deny following sign in requests (sign in command) until some sessions are terminated by signing out (sign out command). The details of this feature will be explained later. The reason behind usage of parallel sessions is there is a rewarding system on our system. Your system gives coupons and discounts using the combination of the time that user spends on the system and the number of interactions with the system (such as Tip or Review). This rewarding system will also be explained in the later sections.

2 Database

You will use the Yelp dataset that you've used in the Project 1. If you have not deleted the database, GOOD! :). If you have deleted, you should create the almost the same setup of Project 1 (we will not use Friend table). The database schema should look like this:
Business(business_id, business_name, address, state, is_open, stars)
Users(user_id, user_name, review_count, yelping_since, useful, funny, cool, fans, average_stars, session_count)
Review(review_id, user_id, business_id, stars, date, useful, funny, cool)

Tip(tip_id,business_id, user_id, date, tip_text, compliment_count)
Also, you need to add 2 more tables to your system to support new subscription system:

Membership(membership_id serial, membership_name text, max_parallel_sessions int, monthly_fee int)
Subscription(user_id, membership_id, time_spent real)

All existing users, should not have any subscription ('Free' type). If you want you can create a membership named 'Free' and add all existing users to subscription table with the membership_id of 'Free' type. However, this is **not necessary**, you may assume if the **user is not in the subscription table it is 'Free' user**. (Your own design choice).

There are unlimited Memberships. The meaning of them explained below. There will be at least 3 Premium memberships (at least 3 types should be in Membership table and a 'Free' type). You may assume new memberships will be added to the system using one of the commands (will be explained in the following sections).

To be clear about the new tables:

- **Membership**: represents available membership types for the users. "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 user who is using this membership can connect to Yelp with at most 2 devices at the same time. Every user should be stored in User table, if they are using a Premium membership, this value should be added to the Subscription table. Number of memberships are unlimited. The key difference between these memberships is allowed maximum parallel sessions (also fee is increasing as max sessions increases). Other than naming, fee and maximum parallel sessions all membership types are considered as Premium user. They can interact with the system in the same manner.
- **Subscription**: shows the premium users and their membership types. Also there is a value named time_spent which indicates how much time the user spent in the system.
- **Users**: There is a little change in the User table. You should add another column named session_count (int) to the table which defines current parallel sessions that the user is connected to the system. For all existing users, you should set the session_count to 0. If the "session count" is equal to "max parallel sessions" of that membership there should be no other connection by that user allowed. For 'Free' users (if user not in Membership table) "max parallel sessions" is equal to 1.
- **Note**: Since the User table is changed a little bit (added column), you need to figure out how to alter the table that already contains some data.

3 The Software

The application should be written in **Python3** and you should use **psycopg2** for database operations in PostgreSQL. Here there are some example links for psycopg2 and database operations using Python3:

- psycopg
- Python Transaction Management
- Transactions with psycopg
- PostgreSQL-Python Transaction

Moreover, we've provided you a source package which will be helpful for this project. The contents of the source package and their purposes are as follows:

- **main.py:** This file is organizing the inputs, requests and response messages. Generally communicating with users. The function is acting like a backend service. You should not modify this file.
  You can run the project using this file and the following command:
  >_ python3 main.py
- **trip.py:** This is the main file to handle your operations and you should modify **ONLY** this file (except your configurations). There are some functions given in the file for each task (see next section) and you should implement those functions.
- **validators.py:** This file is created for checking erroneous input. You don't need to worry about them. Each input taken from the user, first validated by this file then your functions are called.
- **user.py:** A user class implemented for simplicity. You should be able to create and return users on this class basis.
- **messages.py:** All warning or error messages is implemented here. You should only get the correct message for correct situation and return that. You can not add your own messages or return any other message than implemented messages in here. For example, this is a message in the messages.py:
  CMD_UNDEFINED = "Command is undefined. See available options with 'help'."
  The variables is guaranteed to be same, however, their values might change afterwards like: CMD_UNDEFINED = "Command is undefined... See help mode..."
  So, don't write messages directly as strings, use messages.py
- **database.cfg:** This file contains database configurations. Change them with your configurations to connect the database on your local machine.
- **NOTE:** Do not modify any other files than trip.py and database.cfg. ONLY work on trip.py

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

```
> python3 main.py
*** Please enter one of the following commands ***
> help
> sign_up <user_id> <first_name> <last_name>
> sign_in <user_id>
> quit
ANONYMOUS >
```

At first, there is no signed in user. User is shown as "ANONYMOUS" and you can only call "help", "sign up", "sign in", "quit" commands with anonymous user, so only listings (output of help command, directly called when the program starts) are only contains these operations. "help" command will remind you what are the available commands provided by this software (see above).
If you implement "sign in" command correctly and sign in with an existing user information, the look will change to authenticated user like below (assume that the user already exists id=UID12345 user_name = 'Osman Guloglu'):

```
ANONYMOUS > sign_in UID12345
OK
Osman Guloglu (UID12345) >
```

Authenticated users if they are 'Free' can use "**sign_out**", "**show_memberships**", "**show_subscription**", "**subscribe**", "**review**", "**search_business**", "**quit**" commands.
These commands are available to all signed in users.
Assume Ali Osmanoglu (UID98765) is a 'Free' user then (we already signed in at the previous step.):

```
Ali Osmanoglu (UID98765) > help
*** Please enter one of the following commands ***
> help
> sign_up <user_id> <first_name> <last_name>
> sign_in <user_id>
> sign_out
> show_memberships
> show_subscription
> subscribe <membership_id>
> review <review_id> <business_id> <stars>
> search_for_businesses <keyword1> <keyword2> <keyword3> ... <keywordN>
> quit
Ali Osmanoglu (UID12345) >
```

Premium users users can use all previous commands ("**sign_out**", "**show_memberships**", "**show_subscription**", "**subscribe**", "**review**", "**search_business**", "**quit**") and also "**suggest_business**" and "**get_coupon**" commands.
These two commands can be used by premium users ONLY.

Now if we sign out and sign in again with UID98765 "Osman Guloglu" which is a Premium User.

```
Ali Osmanoglu (UID98765) > sign_out
OK
ANONYMOUS > sign_in UID12345
OK
Osman Guloglu (UID12345) > help
*** Please enter one of the following commands ***
> help
> sign_up <user_id> <first_name> <last_name>
> sign_in <user_id>
> sign_out
> show_memberships
> show_subscription
> subscribe <membership_id>
> review <review_id> <business_id> <stars>
> search_for_businesses <keyword1> <keyword2> <keyword3> ... <keywordN>
> suggest_business
> get_coupon
> quit
Osman Guloglu (UID12345) >
```

4 Tasks

You should find

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

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

4.1 Sign Up

$sign\_up < user\_id >< first\_name >< last\_name >$
You need to implement sign_up() function inside trip.py. This is the command to create a new user and add him to the Users table. The anonymous user enters user information (id and name) to sign up to our system.
Each user is considered as Free user at the sign up stage until they subscribe (check 4.6) to a membership.
You don't need to merge first_name and last_name given as input, the function inside trip.py takes user_name as merged first and last name.

You should create a new user with provided information in the database. Since the User table has other fields, you should set them as follows:

- review_count: 0
- yelping_since: The current date of the system (you can use datetime module)
- useful, cool, fans, average_stars: 0
- session_count: 0

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

```
ANONYMOUS > sign_up UID22034 Yagmur Ceceli
OK
ANONYMOUS >
```

If a user with ID 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 user with id UID22034) :

```
ANONYMOUS > sign_up UID22034 Yagmur Ceceli
ERROR: Can not execute the given command.
```

4.2 Sign In

sign_in < user_id >
You need to implement sign_in() function inside trip.py. This is the command for a user to sign in to the service. The user types in user credentials (in our case we only have user_id). You should implement required authentication and session management logic using your user database. Remember to increment session count inside User table for the user. Also check whether the user is out of sessions or not by looking at max parallel sessions of the user's membership (for 'Free' users it is 1 by default, you need).
Finally, you need to start a timer that counts until user sign_outs from the system.
Successful sign in operation should look like this:

```
ANONYMOUS > sign_in UID12345
OK
Osman Guloglu (UID12345) >
```

Failed sign in operation should look like this:

```
ANONYMOUS > sign_in UID53123
ERROR: There is no user with such id.
ANONYMOUS >
```

If session count >= max parallel sessions, then the output should look like this:

```
ANONYMOUS > sign_in UID12345
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 trip.py. This is the command for an authenticated user 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 User table for the user. Also check whether the user's session count can be at least 0.
Finally, you need to stop the timer you've started in sign in operation and add the count in ms (2 decimal floating point) to that user's time_spent in the Subscription table. Successful sign out operation should look like this:

```
Osman Guloglu (UID12345) > sign_out
OK
ANONYMOUS >
```

Failed sign out operation should look like this (already not signed_in user or system error):

```
Osman Guloglu (UID12345) > sign_out
ERROR: Can not execute the given command.
Osman Guloglu (UID12345) >
```

4.4 Show Memberships

show_memberships
You need to implement show_memberships() function inside trip.py. This is the command to get list of all available memberships to subscribe. This command does not have any parameters. You should print all columns of the memberships in the database (e.g. membership_id, membership_name, etc.). Follow the pattern below while printing columns:
When there is an authenticated user, show memberships operation should look like this:

```
Osman Guloglu (UID12345) > show_memberships
#|Name|Max Sessions|Monthly Fee
1|Silver|2|30
2|Gold|4|50
3|Platinum|10|90
```

4.5 Show Subscription

show_subscription
You need to implement show_subscription() function inside trip.py. This is the command to get the details of the membership to which the authenticated user is subscribed. This command does not have any parameters. Printing should be similar with the show memberships command. When there is an authenticated user, show subscription operation should look like this:

```
Osman Guloglu (UID12345) > show_subscription
#|Name|Max Sessions|Monthly Fee
2|Gold|4|50
```

4.6 Subscribe

subscribe < membership_id >
You need to implement subscribe() function inside trip.py. This is the command for authenticated user to subscribe to another membership. Since each user is considered as 'Free' user at the sign up stage, they should use this method in order to become a Premium user. You must ensure that users will not subscribe to a new membership with less parallel sessions allowed. You should update the subscription information for the authenticated user on the user database.
Assume that Osman Guloglu has a membership with id=2 and max parallel sessions=4 and wants to subscribe to a membership with id=3 and max parallel sessions=10. 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.

```
Osman Guloglu (UID12345) > show_subscription
#|Name|Max Sessions|Monthly Fee
2|Gold|4|50
Osman Guloglu (UID12345) > subscribe 3
OK
Osman Guloglu (UID12345) > show_subscription
#|Name|Max Sessions|Monthly Fee
3|Platinum|10|90
```

However, you must reject the command if old max parallel sessions > new max parallel sessions.
When we try to change Osman's membership with the old membership_id, following happens:

```
Osman Guloglu (UID12345) > subscribe 2
ERROR: New membership's max parallel sessions must be greater than
or equal to current membership's max parallel sessions.
Osman Guloglu (UID12345) > show_subscription
#|Name|Max Sessions|Monthly Fee
3|Platinum|10|90
```

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

```
Osman Guloglu (UID12345) > subscribe 99
ERROR: Membership is not found.
Osman Guloglu (UID12345) > show_subscription
#|Name|Max Sessions|Monthly Fee
3|Platinum|10|90
```

4.7 Review

$review < review\_id >< business\_id >< stars >$
Now, for this task, you need to add a user review to the Review table. The function should be defined inside trip.py. The command is called with review_id (PK of the table), business_id indicates the business that the user wants to review and stars given to that business. You need to control the two values as follows:

- review_id: If there is a review with this id, operation should not be permited.
- business_id: If there is not a business with that id, operation should not be permitted.

Example (assume that review_id = 1 is already on the Review table and business_id = 100 is not referring any business).

```
Osman Guloglu (UID12345) > review 1 50 4
ERROR: Operation is not permitted.
Osman Guloglu (UID12345) > review 200 100 4
ERROR: Operation is not permitted.
```

If these conditions are satisfied, then you can add the review to the Review table. However, there are some missing values that are part of Review table, you **must** fill them as follows:

- date: The current date of the system (you can use datetime module)
- useful, funny, cool : 0

4.8 Search Business

$search\_for\_businesses < keyword1 >< keyword2 >< keyword3 > ... < keywordN >$
Now, you should be able to implement search business operation inside the trip.py. The input taken from the user word by word, however, the function takes them all as a sentence (merged words), so you don't need to consider merging them together. You should be able to find the businesses that somehow includes the search_keys in case insensitive form. You need to print the found businesses in following format:

```
Osman Guloglu (UID12345) > search_for_businesses Coffee Ankara
Id|Name|State|Is_open|Stars
1|A4 Coffee Ankara|ANK|1|4
2|Tetra N Caffeine Coffee Ankara|ANK|1|4
3|Grano Coffee Ankara|ANK|1|5
```

Note: Do not forget we are applying "CASE INSENSITIVE" search.
Note2: Order businesses by business_id

4.9 Suggest Business

suggest_business
For this task, you need to suggest businesses that the current user may like in the trip.py. You should follow 3 steps and merge the results coming from those steps.

Step 1 Gather the reviews of that user. From these reviews, find the top state by the reviewed business count. Then, from all **open** businesses find the businesses that is located in the found state. You should collect top 5 businesses by stars.

Step 2 Perform the same thing on the Tip table instead of Review table.

Step 3 Again check the review table to find the businesses get top stars from that user. Among them get the latest reviewed one. Now you need to find **open** top 3 businesses that is located in the same state and has the most stars (if there is an equality order by name and get top 3).

Final Merge these 3 results. Your suggestions should not include any duplicate businesses.

Print Format (is the same as search business):

```
Osman Guloglu (UID12345) > suggest_business
Id|Name|State|Is_open|Stars
1|A4 Coffee Ankara|ANK|1|4
2|Tetra N Caffeine Coffee Ankara|ANK|1|4
3|Grano Coffee Ankara|ANK|1|5
```

Note: Order businesses by business_id Note2: Only premium users are allowed to use this operation. If otherwise occurs, you should not permit the operation as follows:

```
Ali Osmanoglu (UID98765) > suggest_business
ERROR: You are not allowed to perform this operation
```

4.10 Get Coupon

get_coupon
For this task, you need to create coupons for the user inside the trip.py. There is a simple algorithm for creating coupons:
Score = time_spent + 10 * review_count
time_spent is the value in ms you should be able to calculate in the previous parts. review_count is the number of Reviews made by that user (count of this user_id in the Review table).
After you calculate the score, you need to find a percentage using threshold value (variable inside messages.py).
Actual Discount Percentage = $\frac{Score}{Threshold} * 100$

We are allowed to generate coupons at least for 25% percentage and maximum for 50% percentage. So, if the found percentage is lower than 25% you should not perform anything as follows:

```
Osman Guloglu (UID12345) > get_coupons
ERROR: You don't have enough score for coupons.
```

However, if the percentage is between 25 and 50 percent you should create a coupon for that user and print as follows:

```
\begin{verbatim}
Osman Guloglu (UID12345) > get_coupons
Creating X% discount coupon.
OK
```

After you create the coupon, you should set time_spent value of the User to 0.
Finally, if the found percentage is higher than 50 percent, you should create a 50% discount coupon and set the use the minimum amount of time from that user (ie. save remaining times of that user only reduce the times that are used in creating coupons).

```
Osman Guloglu (UID12345) > get_coupons
Creating 50% discount coupon.
OK
```

Note: Only premium users are allowed to use this operation. If otherwise occurs, you should not permit the operation as follows:

```
Ali Osmanoglu (UID98765) > get_coupons
ERROR: You are not allowed to perform this operation
```

4.11 Help

help
The help operation is already defined, however, it prints the same output independent from the user type. Change it in order to print correct outputs for each 3 situations, namely Non authorized, Free and Premium users.
You can check the section 3 for this operation.

4.12 Quit

quit
This operation quits the program. Most of the part is handled in main.py. However, you need to check whether there is a authenticated user or not. If there is such user, simply sign_out for that user and return True. If there is no authenticated user, simply return True. If any error occurs return False (check trip.py for details of return types).

5 Submission

Send a 'tar.gz' file with your 7 digit student id and starting with 'e' like 'e1234567.tar.gz' that contains only trip.py. You should submit 'tar.gz' file to Odtuclass before the deadline.