Ceng352 - Database Management Systems
Project 2 : JDBC, Transactions

Spring 2019

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 connect to two databases, one on PostgreSQL, the other on MySQL (IMDB database from the first project).

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. Your first task is to create a database of your customers. Next, 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 Task 1

Your first task is to design and create the customer database in PostgreSQL. Your database should have following entity sets:

- `Customer (customer_id:integer, email:text, password:text, first_name:text, last_name:text, session_count:integer)`

  This entity 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.

- `Plan (plan_id:integer, name:text, resolution:text, max_parallel_session:integer, monthly_fee:real)`

  This entity represents the available plans to subscribe for customers. You should insert at least three plans with different "max_parallel_session" values. You are free to invent your

own plans. A sample plan is "name : standard, resolution : hd, max_parallel_screen : 2, monthly_fee : 13".

- `Subscription (s_id:integer, customer_id:integer, plan_id:integer)`

  This entity represents the fact that a customer (with customer_id) is subscribed to a plan (with plan_id).

- `Watched (w_id:integer, movie_id:text, customer_id:integer, when:date)`

  This entity represents the fact that a movie (with movie_id) was watched by a customer (with customer_id). "movie_id" is from the IMDB database (imdb.Movies.tconst).

What to Turn in:

Create a text file called "setup_customer_database.sql" with CREATE TABLE statements and INSERT statements that populate each table with a few tuples (say 2-8 tuples). You will turn in this file. This file should load into PostgreSQL with the "\i" command. Test this file by loading it into PostgreSQL while connected to an empty customer database. Your Java program should then be ready to run. Other details of submission are given in last section.

3 Task 2

Write the Java application by completing the starter code. The application is a simple command-line Java program. Your Java application needs to connect to two databases: the IMDB database on MySQL, and your own customer database on your local machine running PostgreSQL.

Once the application is started, the user can select one of the following commands:

1. **sign_up <email><password><first_name><last_name><plan_id>**

   This is the command to create a new customer. The user 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 and a new subscription with provided information in your local customer database.

2. **sign_in <email><password>**

   This is the command for a customer to sign in to the service. The user types in customer credentials (email and password). You should implement required authentication and session management logic using your customer database.

3. **sign_out**

   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.

4. **show_plans**

   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.).

5. **show_subscription**

   This is the command to get the details of the plan to which the authenticated user is subscribed. This command does not have any parameters. Printing should be similar with the **show_plans** command.

6. **subscribe <plan_id>**

   This is the command for authenticated user to subscribe to another plan. The user types in id of the plan that the customer will be subscribed to (plan_id). 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 user on the customer database.

7. **watched_movie <movie_id>**

   This is the command to save the fact that the authenticated customer has just watched a movie. The user types in id of the movie (from the IMDB database) that the customer watched (movie_id). You should save this information to your customer database.

8. **search_for_movies <movie_title>**

   This is the command for authenticated user to search for movies. The user types in a string that is the keyword to search in titles of the movies (movie_title). You should return list of movies whose title matches with provided string. You should print id and title of the movie, names of its directors, names of its actors, and an indication of whether the movie was seen by the authenticated customer before or not.

9. **suggest_movies**

   This is the command for authenticated user to get movie suggestions from the service. This command does not have any parameters. You should follow these steps to make suggestions:

   (1) Find the movies watched by the authenticated user and take the one that was most recently watched.

   (2) Find the genres of that movie and take the one that is lexicographically greatest.

   (3) Find the movies that have the same genre and that were released this year. Order them by their average ratings and take 5 of them with the greatest average ratings.

   (4) Suggest the ones that was not watched by the authenticated user before.

   You should print ids, titles, and the average ratings of the movies.

10. **quit**

    This is the command to quit the application. For your own benefit, it is recommended to sign out before quitting if there is a signed in customer.

What to Turn in:

You will turn in all source files of your application. Other details of submission are given in last section.

4 Starter Code

The starter code is designed to give you a gentle introduction to embedding SQL into Java. The starter code is in the package named "ceng352_project_2.zip". To run the starter code, make following preparations:

- In order to connect to a database from Java, you need two database-specific JDBC drivers, one for PostgreSQL (can be downloaded from `https://jdbc.postgresql.org/download.html`) and one for MySQL (can be downloaded from `https://dev.mysql.com/downloads/connector/j/`). You should add these two drivers to your classpath before running the starter code.
- Start up the PostgreSQL server on your local computer and create a database named CUSTOMER. (Database name must be in all caps!)
- Edit the username and password for MySQL (use the credentials from the first project) and PostgreSQL on "dbconn.config".

Now, you can use following commands on terminal to compile and run the project:

```
javac Service.java
java Service
```

At this point you should see this in the terminal:

```
*** Please enter one of the following comands ***
> sign_up <email> <password> <first_name> <last_name> <plan_id>
> sign_in <email> <password>
> sign_out
> show_plans
> show_subscription
> subscribe <plan_id>
> watched_movie <movie_id>
> search_for_movies <movie_title>
> suggest_movies
> quit
```

There are only 4 files in starter code package: "Service.java", "Query.java", "Customer.java", and "dbconn.config". You are not allowed to modify "Service.java" file. You can add more files to your project if you need.

5 Notes

- You will need to write some of the commands as SQL transactions. Others are interactions with the database that do not require transactions.
- You need transactions only on the CUSTOMER database: you don't need transactions for IMDB, since this database is never updated.
- You need to enforce the following constraints:

  C1: Customers can't subscribe to a new plan with less parallel sessions.

C2: At any time a customer can have at most as many as the maximum number of parallel sessions allowed by his/her subscription.

To ensure constraint C1, when a customer requests to subscribe to a new plan with fewer allowed sessions, you may deny this request. Thus, you need to worry about C in ACID in designing your transactions. HINT: ROLLBACK is your friend.

To ensure constraint C2, you need to ensure that each instance of your application runs in isolation, i.e. the I in ACID. You need to worry about concurrency. A user may try to cheat and coerce your application to violate the maximum parallel sessions constraint by running two instances of your application in parallel, with the same customer_id. Depending on how you write your application and on race conditions, the malicious user may succeed in opening more sessions than he/she is allowed. You need to ensure that each instance of your application runs in isolation.

- Never include a user interaction inside a transaction. That is, don't begin a transaction then wait for the user to decide what she wants to do. Your transactions should not include any user interactions.
- Hint about transactions in Java: Below is a sample code to execute multi-statement transactions from Java.

```
Connection _db;
_db.setAutoCommit(false);

[... execute updates and queries.]

_db.commit();
OR
_db.rollback();
```

6 Submission

You are asked to submit a ".zip" package with all files from both tasks in it. You should name your file as "your_student_id.zip" using your 7-digits METU student ID. You should submit the ".zip" package to ODTÜClass before the deadline.