CENG 351

Data Management and File Structures

Fall 2017-2018

Programming Assignment I

Görkem Özer

(gorkem@ceng.metu.edu.tr)

Due date: 12.12.2017 23:59

1 Introduction

In this assignment, you are going to practice how to manage a MySQL database similar to IMDb [1] using JDBC [2]. You will manage actor, director, movie, user data and give statistics about them. To make you familiar with this topic, you are going to implement pre-defined functions that are defined in a Java interface [3]. When you implement the interface, you have to implement all functions of it. All necessary data files and source files will be provided. Last but not least, please read **"Hints"** and **"Compiling and Running"** sections before you start implementing the homework.

2 Input Files

- **actors.txt:** Input file that contains actor/actress information. Each line of it has actor id, actor full name which are separated by **semicolon (;)**. Each line is in below format and you can see examples:

  ```
  <actor_id>;<actor_name>

  8;Richard S. Castellano

  32;Edward Norton
  ```

- **directors.txt:** Input file that contains director information. Each line of it has director id, director full name which are separated by **semicolon (;).** Each line is in below format and you can see examples:

  ```
  <director_id>;<director_name>

  21;Alfred Hitchcock

  52;Vittorio De Sica
  ```

- **movies.txt:** Input file that contains movie information. Each line of it has movie id, movie title, movie year, movie genre, cast, director name, rating which are separated by **semicolon character (;).** In cast section, there are actor names separated by **comma (,)**. **It is GUARANTEED that there are 4 actors and 1 director in a movie.** Each line is in below format and you can see the example:

  ```
  <movie_id>;<movie_name>;<year>;<genre>;<actor1,actor2,actor3,actor4>;<director>;<rating>

  105;Heat;1995;Action;Al Pacino, Robert De Niro, Val Kilmer, Jon Voight;Michael Mann;8.2
  ```

- **users.txt** Input file that contains user information. Each line of it has user id, user name, user age and id of movies that user watched; which are separated by **semicolon character (;).** In movie ids section, movie ids are separated by **comma (,)**. **It is GUARANTEED that a user has watched 10 movies. Therefore number of movie ids is 10.** Each line is in below format and you can see the example:

  ```
  <user_id>;<user_name>;<age>;<movie_id1, movie_id2, movie_id3, ...>

  52;Naomi Wood;34;65, 17, 194, 19, 148, 23, 25, 27, 11, 44
  ```

3 Objectives

This assignment aims to help you get familiar with

- Java programming language basics,
- Object oriented programming concepts,
- Connecting and querying to MySQL Server using JDBC.

4 Database Schema

You will use (strictly) the schema given below in the scope of this assignment. Underlined columns are **primary keys** for tables.

- **actor** (aid: int, name: varchar(50))
- **director** (did: int, name: varchar(50))
- **movie** (mid: int, title: varchar(100), genre: varchar(50), year: int, did: int, rating: float). **Notice did is the director id.**
- **user** (uid: int, name: varchar(50), age: int)
- **casted_in** (aid: int, mid: int). **Notice aid is the actor id and mid is the movie id.**
- **watched** (uid: int, mid: int). **Notice uid is the user id and mid is the movie id.**

5 Tasks

- Create tables mentioned in **Database Schema** section. (10 points)
- Read and parse input files by implementing `ParseFilesAndInsertData()` method. Insert parsed data into related tables. Return type should be `void`. (10 points)
- Insert parsed data into tables created before by implementing `InsertActor()`, `InsertDirector()`, `InsertMovie()`, `InsertUser()`, `InsertCastedIn()`, `InsertWatched()` methods.
  Return type should be `void` for all these methods. (3 points each)
- Retrieve movies that director with given id has directed by implementing
  `GetMoviesOfDirector(int did)` method. Return type should be `List<Movie>` type. (3 points)
- Retrieve movies that actor with given id is casted in by implementing
  `GetMoviesOfActor(int aid)` method. Return type should be `List<Movie>` type. (3 points)
- Retrieve movies that user with given id has watched by implementing
  `GetUserWatchlist(int uid)` method. Return type should be `List<Movie>` type. (3 points)
- Retrieve movies in which given two actors/actresses casted in together by implementing
  `GetMoviesWithTwoActors(String actorName1, String actorName2)` method. Return type should be `List<Movie>` type. (5 points)
- Retrieve movies that have higher rating than given value by implementing
  `GetMoviesAboveRating(float rating)` method. Return type should be `List<Movie>` type. (5 points)
- Update ratings of movies with given pattern by implementing
  `ChangeRatingsOfMoviesLike(String pattern, float value)` method. Ratings of movies that match pattern will be updated with given value. Return type should be `void`. (5 points)
- Delete director with given did and delete user with given uid by implementing
  `DeleteDirectorWithId(int did)`, `DeleteUserWithId(int uid)` methods. Return type should be `void`. (5 points each)

- Print how many times each movie is watched, with movie details by implementing `PrintViewStatsByMovie()` method. Return type should be `void`. For each movie; print movie id, movie title and view count separated by **pipe character ("|").** (8 points)

  `Output format: movie_id|movie_title|view_count`

- Print how many times movies from each genre is watched by implementing `PrintViewStatsByGenre()` method. Return type should be `void`. For each genre; print genre and view count separated by **pipe character ("|").** (8 points)

  `Output format: genre|view_count`

- Print average rating of movies that directors directed by implementing `PrintAverageMovieRatingOfDirectors()` method. Return type should be `void`. For each director; print director name, average rating and view count separated by **pipe character ("|").** (8 points)

  `Output format: director_name|average_rating|view_count`

- Drop all tables by implementing `DropTables()` method. Return type should be `void`. (4 points)

6 Hints

- **It is GUARANTEED that input files** (actors.txt, directors.txt, movies.txt, users.txt) **are correctly formatted** and sample data will be given to you. There will be no surprises about the data, similar (and larger) data will be used while evaluating homeworks.
- Functions that you have to implement are defined in `IMovieManager` interface. There is also a class named `MovieManager`, which implements `IMovieManager` interface in source files. All you need to do is fill empty function bodies inside `MovieManager` class.
- It may be beneficial to use `InsertActor()`, `InsertDirector()`, `InsertMovie()`, `InsertUser()`, `InsertCastedIn()`, `InsertWatched()` methods inside `ParseFilesAndInsertData()` method.
- `InitializeConnection()` method is beneficial for you, to make you avoid creating database connection every time before an operation is done. It is recommended to use private `_connection` object before every database operation.
- `databaseConfiguration.txt` file helps you to change settings for your environment. You can set server, username, password, port settings from here, instead of setting them inside the source code. **Please, use the template main function given to you, for doing database configuration and initializing database connection.**
- Designing a database is an important part in software engineering. Better database design should lower the burden you handle while implementing the homework.
  For methods `DeleteDirectorWithId(int did)` and `DeleteUserWithId(int uid)`, keep this advice in mind. Provide referential integrity in the database.
  When a director is deleted, make sure that related films are deleted. When these films are deleted, make sure that related user-movie relations are deleted. Also when a user is deleted, make sure that user-movie relations are deleted. How you achieve these goals is up to you.
- When implementing methods for getting statistics (`PrintViewStatsByMovie()`, `PrintViewStatsByGenre()`, `PrintAverageMovieRatingOfDirectors()`), you **HAVE TO** follow output formats provided as comments in `IMovieManager` interface. Print each field with a **pipe character ("|")** among them, with no spaces before/after pipes.
- You don't have to send your main function in implementation. Only implementations of functions defined in the `IMovieManager` matter.

7 Regulations

- **Programming Language:** Java.
- **Database:** You will connect to local MySQL server on your machine. Use `databaseConfiguration.txt` file to change database connection settings. Detailed explanation for setting up a local MySQL server will be given to you, in homework files.
- **Attachments:** Necessary source files and JDBC driver will be provided.
- **Late Submission:** Late submission policy is stated in the course syllabus.
- **Cheating:** We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations.
- **Newsgroup:** You must follow the newsgroup [6] for discussions and possible updates on a daily basis.
- **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. Please, be noticed that you have to achieve the tasks only within your SQL queries not with any other java programming facilities.

8 Submission

Submission will be done via COW. Create a compressed file named `eXXXXXXX.zip` that contains `Source` directory which contains `MovieManager` class implemented by you. **You will not submit mysql.jar file, interface file, class files and input files provided to you. Only submit MovieManager.java implementation.** So, make sure that you do not modify classes other than `MovieManager.java` during implementation. Because evaluation will be held with unmodified versions of them. In short, you should put your MovieManager implementation into directory named `Source`, and compress this directory as .zip file and name it `<eXXXXXXX.zip>`, which should actually be **your student id.**

9 Compiling and Running

For Ubuntu:

```
>_ cd Source
>_ javac -cp .:mysql.jar *.java
>_ java -cp .:mysql.jar MovieManager
```

For Windows:

```
>_ cd Source
>_ javac -cp .;mysql.jar *.java
>_ java -cp .;mysql.jar MovieManager
```

To boost you about installing MySQL server and working with Java in an IDE software, you should look at tutorials about XAMPP and NetBeans IDE. Detailed guidelines are explained inside these tutorials for both Ubuntu and Windows. You can do MySQL installation manually and you can do development in Java by just editing the files. However, these tutorials are **STRONGLY** recommended, to speed you up. **You can find these tutorials in homework page.** Also, you can check links in [7].

References

[1] **IMDb (Internet Movie Database)**
https://www.imdb.com/

[2] **Java Database Connectivity and Tutorials**
https://en.0wikipedia.org/wiki/Java_Database_Connectivity
http://www.mkyong.com/tutorials/jdbc-tutorials/
https://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm

[3] **What is interface in Java?**
https://www.tutorialspoint.com/java/java_interfaces.htm

These links may help you while installing Java 8 and MySQL:

[4] **Java 8 Installation for Ubuntu**
https://tecadmin.net/install-oracle-java-8-ubuntu-via-ppa/

[5] **MySQL Installation for Ubuntu**
https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-16-04

[6] **Newsgroup**
https://cow.ceng.metu.edu.tr/News/

[7] **Helpful Video Tutorials**
https://www.youtube.com/watch?v=kCsFkNYahP0 (Mustafa Murat Coşkun - Turkish)
https://www.youtube.com/watch?v=dV3JjLhi4Jk (English)
https://www.youtube.com/watch?v=UWZQJrAtEu4 (English)