CS5200: Homework 4: Interoperability via JDBC

Fall 2024

1 Overview

In this assignment, you will write Java classes that allow an application to interact with a running MySQL instance. In doing so, you will practice the following skills:

- Designing and implementing model classes to represent database records
- Working with SQL connections, the Java abstraction for communication between an application and a database
- Defining and executing parameterized prepared SQL statements
- Retrieving results from an executed statement.

PLEASE NOTE: this assignment requires significantly more time than most others. I strongly encourage you to start on this early!

Use JDBC to create the data access layer for a database that tracks applications to a university. I have provided applications.sql,[1] which defines the schema and the tables that you are to use. Please note that the `LetterWriter`, `Reviewer`, and `Applicant` tables represent subtypes of the `User` table, and your class definitions should reflect this.

For each table in this schema, you must create two Java classes: a model class (similar to BlogUsers.java from the in-class example) and a data-access object (DAO) class (similar to BlogUsersDao.java). Use the JDBC data access layer example we had in class for the blog application as a starting point; you can find that example here: BlogApplication.zip.[2]

We recommend the use of Eclipse, as demonstrated in class, for this project. You may choose to use other development environments if you prefer, but if you do so it is *your* responsibility to configure the project and get all of the various components (JDBC, etc.) working together well. The course staff is not able to support IDEs other than Eclipse.

You must use Java for this assignment; other programming languages are not permitted.

All submitted code must use the techniques demonstrated in lecture to guard against SQL injection attacks. We will impose a penalty of up to 50 points for insecure code.

You should upload a single .zip file containing .java files for all of the classes described below. (If you use the same project structure as the sample application, you can just construct a zip file out of your project's src/main/java directory.)

You should implement the following classes, with the specified methods:

- Model classes for each of the tables in the schema above: `Applicant`, `Degree`, `RecLetter`, and so on.
  - You are not only permitted but encouraged to use your IDE's support for automatically generating getter and setter methods!
- ConnectionManager

[1] That link should work, but in case it doesn't, the file is available in Canvas: Files > homework-files > hw-04-files > applications.sql.

[2] This is the same file that we downloaded during project setup this week. If the link doesn't work, it's available in Canvas in the same location as applications.sql.

  - Re-use the file from the blog application example, and update with your MySQL instance properties.

- UserDao
  - `public User create(User user)`
  - `public User getUserByUserID(int userID)`
  - `public User delete(User user)`
- ApplicantDao
  - `public Applicant create(Applicant applicant)`
  - `public Applicant getApplicantByUserID(int userID)`
  - `public List<Applicant> getApplicantsByProgram(Applicant.Program)`
    - You should define `Program` as a Java enum inside the `Applicant` model class. Use `BlogUsers.StatusLevel` in the sample application as a model.
  - `public Applicant delete(Applicant applicant)`
- ReviewerDao
  - `public Reviewer create(Reviewer reviewer)`
  - `public Reviewer getReviewerByUserID(int userID)`
- LetterWriterDao
  - `public LetterWriter create(LetterWriter letterWriter)`
  - `public LetterWriter getLetterWriterByUserID(int userID)`
- DegreeDao
  - `public Degree create(Degree degree)`
  - `public List<Degree> getDegreesByApplicant(Applicant applicant)`
- RecLetterDao
  - `public RecLetter create(RecLetter letter)`
  - `public RecLetter getRecLetterByApplicantAndAuthor(Applicant applicant, LetterWriter writer)`
  - `public List<RecLetter> getRecLettersByApplicant(Applicant applicant)`
- RatingDao
  - `public Rating create(Rating rating)`
  - `public List<Rating> getRatingByApplicant(Applicant applicant)`
  - `public Rating updateRating(Rating rating, int newRating)`
- Inserter
  - For each data-access class above: exercise the create, read, update, and delete operations (if the method exists) using the data access object.

For the `get...` methods in the data-access classes: if the method returns a single record from the database, it should return `null` if the specified record is not present. If, on the other hand, the method returns a list of records, then it must return an empty list (which is *not* the same thing as `null`) if no records match the search criteria.