CIS 331
University Management System
Group Project Part 4: Database Integration

The client was pleased with the prototype JavaFX based System you have developed! Now they would like to see one more prototype feature: Database integration. Your PM would like you to use your Oracle Express Edition DB skills and integrate communication with the DB into your application. This way, data is not destroyed/lost when the application closes.

To Do FIRST:

- First, make a **backup copy** of your entire Part 3 NetBeans JavaFX Project Folder. Store it on the cloud or in another safe location so that you can revert back if something goes wrong.

Part 4 Requirements:

- All functionality from Part 3 should be included (or finally completed) for Part 4. You can make small tweaks and changes as needed or in response to Part 3 feedback.
- **Your team will need to use your ERD from Part 1 as a starting point to design and implement your DB tables for Part 4. Create your entity tables for Student, Faculty, Course, and Semester. Their fields should roughly match your Classes. Create associative/bridge tables for Schedule and Enrollment. Test things out in SQL first before diving deep in to Java integration.**
- Think about the JOIN queries you'll need to write to drive your reports (potentially unless you keep your app logic for printing reports from instance objects).
- **All tables should have dedicated primary keys. Use foreign keys in your bridge/associative tables.**
- Make any changes you need to your Java Classes to make your DB life easier. **BE CAREFUL**: Any changes to your classes may break existing application code that will need to be fixed!
- Your team should think about and choose either a "real-time" nature to the database interaction or an "backup-and-load" nature.
    - "Real Time" Approach: Your team will code your JavaFX application so that data is written out to the DB in real-time as it is created in the application. When the application first loads, it will load all the data from the DB tables, (possibly) create instance objects from them, and display them in the GUI.
    - "Backup-and-load" Approach: As the user interacts with the application, instance objects are created in memory. No database writing occurs until the user closes the application window. When this occurs, JavaFX automatically will attempt to call a "stop()" method in your application. If you override the Application class' stop() method in your App class, it will be invoked. Code for writing everything out to the DB can occur there. When the user opens the application, it will load all data from the DB tables, (possibly) create instance objects from them, and display them in the GUI.

Things to Keep in Mind:

- Do not print anything to the output console. All interaction with the client should occur through your JavaFX form (messages and input).
- **Take a close look** at the Ch. 12 Supplemental PDF on Canvas, as it walks you through DB interaction for a JavaFX Grocery Store application.
- **Information on the `stop()` method:**
    - https://docs.oracle.com/javase/8/javafx/api/javafx/application/Application.html#stop--
    - **https://www.tutorialspoint.com/javafx/javafx_application.htm**

Deliverable Requirements:

- **You will include SQL in a text file with your submission, BOTH create and insert statements, so that I can recreate your DB and your application will run and talk to the DB. Your SQL should CREATE your tables and then include INSERT statements to put test data into the tables prior to the application starting.**
- **Copy your entire Part 4 NetBeans project folder into another folder. In the same top level, place a copy of your SQL text file. Include an additional text file IF you need to leave me any instructions / things I should know. Then, ZIP up that overall containing folder.**
- Choose **one** team member to submit.
- **Make sure** all team member names are listed in the comment header of your App.java JavaFX application file.
- **Make sure** all team member names are listed in the Canvas Comments of the submission.

Grading and Extra Credit:

- **Normal Grade:** Your application runs without any errors. Your application reads/writes data from just the Entity Tables (Student, Course, Faculty, and Semester). Your application uses Instance Objects in memory after the data is loaded, and reads from them when writing to the DB.
- **Normal Grade + 6 Exam Bonus Points on your Lowest Exam Score:** In addition to the "Normal Grade" functionality above, your application writes to and reads from bridge/associative tables in the DB and does so without error. For example, if the user enrolls a Student in a Course in a Semester, an Enrollment instance object is in memory and then (eventually) a bridge table record is written out to your DB for that Enrollment. When the application launches, the enrollment record is read from the bridge table and represented visually in the app and with Instance Objects in memory. (Either the "real time" or "backup-and-load" approaches will qualify for this).