CSCE 156 – Computer Science II

Lab 10.5 - Java Persistence API (JPA)

Dr. Chris Bourke

Prior to Lab

1. Review the lecture notes and examples of JPA
2. Read the following tutorial on JPA: `http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html`

Lab Objectives & Topics

Following the lab, you should be able to:

- Understand the basics of JPA
- Use JPA to annotate Java classes to facilitate loading data
- Use JPA to persist data to a database

Peer Programming Pair-Up

To encourage collaboration and a team environment, labs will be structured in a *pair programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

Java Persistence API

TODO

Activities

Clone the starter code for this lab from GitHub using the following url: `https://github.com/cbourke/TODO`.

Configuration

- Be sure that you have the albums database installed on your CSE database
- Open the `config/META-INF/persistence.xml` file. This is the primary configuration file that will allow you to define "persistence units," which are essentially database profiles.
- We have already defined a persistence unit named `"album_database"`. Change the `url`, `username` and `password` values to reflect your database credentials.

Troubleshooting

Open the `Part1` class and run the code. There are several problems that you will need to address before it works properly.

1. The first time you run it, the exception will be self-explanatory, fix the issue and rerun it.
2. The second time you run it, there will be a different error. Observe that in the `DataLoader` class, the `EntityManager` is closed before each album's band can

be loaded from the data base. Focus on the annotations responsible for the `Album` → `Band` relation and fix the problem.

Annotating a Class

Open the `Part2` class and run the code. It should print album information, but it does not print the songs on each album. This is because we have not annotated the `AlbumSong` "join class" and associated the entity with the `List<AlbumSong> songs` member variable in the `Album` class.

1. First, remove the `@Transient` annotation on the `songs` variable in the `Album` class[1] and replace it with the following annotations:

   ```
   @OneToMany(fetch=FetchType.EAGER)
   @JoinColumn(name="albumId", nullable=false)
   @OrderBy("trackNumber ASC")
   ```

   Run the demo again and observe the results.

2. Now annotate the `AlbumSong` class to make it an entity.

3. Contrast the way we associated the `Album` and `Song` classes through the `AlbumSong` "join class" and the way we associated the `Band` and `Musician` classes. Both are many-to-many relationships with join tables, but one has a join class and the other does not. Discuss with your partner the consequences of these design choices.

Querying Using JPQL

You will now get some practice writing JPA code to load data from a database using JPQL. Open the `Part3` class.

1. Implement the `getBands()` and `getBandById(int bandId)` methods in the `DataLoader` class. Use the other methods in that class as examples.

2. Run the `Part3` class to troubleshoot your implementations.

Persisting Data

You will now use JPA to persist (save) data to the database by implementing the `addAlbum()` method in the `DataPersister` class. To do this you will need to:

[1]We only placed this annotation on the class to get the first demo to work.

1. Create an `EntityManager` object
2. Get the `EntityManager`'s transaction and begin it.
3. Persist the given `Album` object
4. Commit the transaction
5. Catch and handle any exceptions (rolling back if necessary) and clean up your resources.

Test your code using the `Part4` class.

Advanced Activity

Modify the albums web app (labs 9 and 10) to use your JPA classes. To get this to work you will need to:

- Make sure that all the required JAR files are in the `WebContent/WEB-INF/lib` folder (JAR files in subdirectories will not be found).
- Place your `persistence.xml` file in the `WebContent/WEB-INF/classes/META-INF` folder (you may need to manually create this.