CSCE 156 – Lab: Using JDBC in a Web Application II

Handout

0. Prior to the Laboratory

1. Review the laboratory handout
2. Make sure that the Albums database is installed and available in your mysql instance on CSE
3. Be sure to have completed and understood Lab 8 as a prerequisite to this lab.

1. Lab Objectives & Topics

Upon completion of this lab you should be able to understand how to:

- Write SQL queries to insert into a database using JDBC
- Handle data integrity problems arising from bad user input and database integrity constraints (foreign key constraints)
- Have further exposure to a multi-tiered application using Java Servlets and the HTTP request/response protocol.

2. Problem Statement

In this lab you will continue to work with the Java Database Connectivity API (JDBC) by finishing some new functionality to the Album web application from the prior lab.

The application has been updated to include a link (in the `albumList.jsp` page) to another page where a user can add an album by providing the Title, Band (by name), Year, and Album Number (in their catalog).

Again, the focus will be on the JDBC functionality for inserting these records, but you are invited to look into how the HTML web form posts to an XML configured (`web.xml`) Java Servlet (`AlbumAdderServlet.java`) which hands over the business logic of inserting the album to the database using another Plain Old Java Object (POJO), `AlbumAdder.java`.

Your task is to make changes to the `AlbumAdder` class to take the four pieces of information provided by the user and insert records in the Album database (specifically the `Albums` and the `Bands` tables). You will put your JDBC code in the `AddAlbumToDatabase()` method (the `AlbumAdder` class is instantiated with the four pieces of data that you'll need). Keep in mind however:

1. The POST and GET methods in the HTTP protocol only transmit String data. Consequently the Servlet that handles the request and the `AlbumAdder` class that it builds only deals with

strings. Your database expects integers for some fields. You will need to handle the validation somehow.

2. The user should not be expected to know the internal keys to a database—thus the input for asks them to provide a band *name*. To preserve the integrity of data, you should not insert duplicate band names. Therefore, you should check to see if the Band record already exists (according to its name). If it does, use that foreign key; if it doesn't, then you'll also need to insert the Band record.

Hints & Tips

Relations & Keys

You cannot insert an album record into the Albums table before you have inserted a band record into the Bands table. This is because the Albums table has a foreign key reference to a band. This is good database design it ensures good data integrity.

Unfortunately, it presents a problem: we can insert a band record, but we need to immediately pull the auto-generated key of the inserted record so we can use it in the album record. There are many mechanisms for dealing with this, unfortunately a lot of them are vendor-specific (they work with one database, but not others).

One mechanism in mysql is the `LAST_INSERT_ID()` function which returns the ID of the last inserted record with an auto generated key. You can query the database using JDBC to call this function as follows:

```
PreparedStatement ps = conn.prepareStatement("INSERT INTO Table (…) VALUES (…)");
ps.set…
ps.executeUpdate();
ps = conn.prepareStatement("SELECT LAST_INSERT_ID()");
ResultSet rs = ps.executeQuery();
rs.next();
foreignKeyId = rs.getInt("LAST_INSERT_ID()");
```

Debugging

Recall from last lab that we provided you a main method that you could use to test your code prior to deploying it to the glassfish server. You may consider adding a similar testing method(s) in this lab before you deploy your project.

3. Instructions

For this lab, you will need to use the JEE (Java Enterprise Edition) version of Eclipse, *not* the JSE (Java Standard Edition). In Windows, click the start menu and enter "Eclipse", the "Java EE Eclipse" should show up, select this version. You may use the same workspace as with the JSE (Java Standard Edition) version of Eclipse.

Importing Your Project

1. Be sure that the Albums database from the prior lab is still installed and available on your mysql database on cse.
2. Download the zip archive file from blackboard and import it into Eclipse:
    a. File -> Import -> General -> Existing Projects into Workspace
    b. Select "Select archive file" and select the zip file you downloaded
3. The java code is located under the "Java Resources -> src" directory
4. The JSP and web files are located under the "WebContent" directory

Modifying Your Application

1. You will first need to make changes to the `unl.cse.music.DatabaseInfo` java source file:
    a. Change instances of `YOURLOGIN` to your cse login
    b. Change instances of `YOURSQLPASSWORD` to your mysql password (if you have misplaced it, it can be reset by going to http://ponca.unl.edu)
2. All the JSP and most of the Java code is complete except for the `AddAlbumToDatabase()` method in the `AlbumAdder` class. You can make any modifications to the other classes, JSP or HTML files that you feel will help you, but modifying code in one part may break functionality in another.

Deploying Your Application

Follow the instructions in the previous lab to deploy your application as a WAR file either locally or to the csce server.

4. Completing Your Lab

1. Complete the worksheet and have your lab instructor sign off on it.
2. Before you leave, make sure you "undeploy" your web app by deleting the WAR file from the glassfish autodeploy directory:
```
rm /usr/local/glassfishv3/glassfish/domains/domain1/autodeploy/loginLab09.war
```

Advanced Activities (Optional)

1. Every piece of data coming from a webform is treated as a string. Your `AlbumAdder` had to do the necessary conversion and handle the situation where invalid (non-numeric) data was provided. We could have also done the validation at the client tier to prevent bad data from being sent to the server. Add some Javascript to your page to validate the data to prevent the user from clicking the "Add" button unless the year and album number are valid. For some pointers on Javascript and form validation, see: http://www.w3schools.com/js/js_form_validation.asp
2. A Cross-Site Scripting (XSS) attack is an attack that is similar to an SQL injection attack that involves an attacker who injects HTML into a webpage by submitting it as legitimate input but

the application fails to properly sanitize the HTML when it gets served back to the user. We will demonstrate that our webapp is vulnerable to this attack by hacking it.

a. Modify your Album database and extend the length of the Band.BandName column by opening your mysql client and executing the following SQL:
```
ALTER TABLE Bands CHANGE COLUMN BandName BandName
VARCHAR(2048) NOT NULL DEFAULT '';
```
b. Navigate to the page where you enter albums/bands and enter the following:
   i. Album Title: "Yeah Ghost"
   ii. Band Name (exactly as shown, no line breaks):
```
Zero 7</a></td><td>2009</td></tr></table><p>For more,
click <a href=
"http://cse.unl.edu/~cbourke/cse156/passwordStealer.ht
ml">HERE</a><!--
```
   iii. Album Year: 2009
   iv. Album Number: 4
c. Submit the new Album and view the list—what has happened? Click the new link at the bottom and see what happens.
d. This attack was not that sophisticated. Imagine injecting Javascript which operates in silence and in the background and sends all browser data to another site and you can get an idea of how severe the problem can be.
e. Read more on how to prevent these kinds of attacks:
   http://en.wikipedia.org/wiki/Cross-site_scripting
   https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
   Modify your project to protect against these sort of attacks.