Extra Credit Update Lab

Lab Summary: In this lab, you will add an update column to two of your data display pages (i.e., your web_users table, and your other table), similar to what is shown below:

<table>
<thead>
<tr>
<th>Id</th>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
<th>Password</th>
<th>Credit Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>174</td>
<td>James</td>
<td>Bond</td>
<td><a href="mailto:007@spies.com">007@spies.com</a></td>
<td>jb</td>
<td></td>
</tr>
<tr>
<td>279</td>
<td>Matt</td>
<td>C</td>
<td>pc@email.com2</td>
<td>oo</td>
<td></td>
</tr>
</tbody>
</table>

If you hover your mouse over the first “update” link, you’ll see this link address:

update_form.jsp?custId=174

Once you click on that link, you should see a page like this. All of all the fields are pre-populated by reading the record with the specified id (e.g., custId=174) from the database on first rendering.

The code of the page above will be very similar to the code you already wrote for the register/insert lab. One difference (between the update page and your insert/register page) is that the update page has to check a field other than customer Id to know if it is postback or not (since customer id is specified even on first rendering when linked from the data display page).
**Functional Requirements:**

Your user display and “other” display pages data shall provide an update link for each record on the page.

- When the user clicks on the update link for a particular record, they shall see a form that has all of the fields of that record pre-populated into textboxes (on first rendering).
- Instead of linking the word “update”, your display pages shall link using an update icon. You can either use the icon provided in the sample code or you can find/create your own icon (you’ll need a png with a transparent background).
- The update form shall have field level validation messages. If all fields pass validation, the record shall be updated and a confirmation message provided. If there was any other error (e.g., database not available, duplicate record, sql syntax error), that message shall be displayed on the page.
- The update form shall have a link back to the display page that shows all the records of that database table.

**Design Specifications:**

In addition to the above functional requirements, here are design specifications that you must also follow.

- The code in your new/modified data display JSP pages shall be short, pushing as much code as possible to reusable java classes.
- Currently, you should have an Insert class inside each of your “model” packages, e.g., “modelWebUser” or “model<<your other table name>>”. Create an Update class that is patterned after the Insert class.
  - If you are unsure of how to specify a legal SQL update statement for your table, use MySql Workbench to help you. Edit a record from your table and click “apply” then take note of the SQL that MySql Workbench shows you.
  - If you want to work on your update_form.jsp page before getting the enhanced list page to work, you can test out the update form by doing “URL tampering”. For example, run update_form.jsp but add “?custId=174” at the right of the URL.
- You have a view method that returns HTML table code – this is the method that you called in your “display data” lab. Instead of modifying this view method (to add update functionality to it), create a new method that is similar to the first one, that supports the update functionality.
  - Your new view method (that returns an HTML table and supports update functionality) shall be in the same class as your old view method.
  - Note that the sample code does not have a viewForUpdate method. You’ll have to move code out of list_for_update.jsp into the new view method that you are being asked to write.
  - This new view method shall not make any assumptions about what is or is not on the JSP page. (Think of the method as a worker and the JSP page as the boss. The worker has no say in what the boss does.) This means that the JSP page needs to pass the following information to the new view method:
    - The name of the update icon image (that the view method will need to use).
    - The name of the JSP page that shall be invoked when the user clicks on an update icon.
Suggestions About How to work:

(1) **Back up** your project (as always) before starting anything.

(2) **Download the Update sample code** from the zip file and install it on your PC in its own project. Test to be sure that this project works properly (you'll need to be tunnelled in unless you are running from a lab PC). Leave this project untouched so it continues to function properly and so you can reference it for help.

(3) **Download the Register (Insert) sample code** from last week and install it also into its own project (unless you already have the untouched Register sample code on your computer).

(4) Since you are new to programming, it is very important that you **make just a few changes** between running and testing your code each iteration.

(5) **Blend the Update sample code into the Register sample code** (and take note of what you had to do because you'll have to do the same things to your web app to blend in Update functionality). I'll try to provide you with a list of things I needed to do:
   a. Copy the Search and Update classes (.java files) into modelCustomer of the insert project. Search and Update should compiler errors because they reference two new data members of modelCustomer.StringData (errorMsg and an id field). Add these data members to modelCustomer.StringData. The Search class should now be error free (save all).
   b. The Update class references dbUtils.PrepStatement. Copy PrepStatement from (the dbUtils package of) the Update project into the dbUtils package of the Insert project.
   c. Copy elements from the Web Pages folder of the Update project (into the Insert Project): the two JSP pages and the icons folder.

(6) At this point, you should have no compiler errors and should be able to **run list_for_update.jsp** and make sure it works.

(7) Now, repeat the process above: **blend the Update Sample code** into your actual CIS 1056 web application project and modify it to be appropriate for your database tables (webUser and “other”).

(8) Note that you are asked to do a few things that go **beyond the sample code**, including:
   a. Use a link icon instead of linking the word “update” for each row of data.
   b. Move the view code out of list_for_update.jsp and into a method within the view package.

Suggestions About How to Debug

c. **HTML/CSS**: render the page in firefox and view source and check to see that there are no red syntax errors identified. Sometimes a page can look good even though there are HTML or CSS syntax errors in it. Netbeans can help you identify HTML/CSS syntax errors if you add a new HTML page to your project, then paste the “view source” of the page you just rendered, then do a source format and look for error messages. (You might even copy/paste the code from an external style sheet into an internal <style> tag in your <head>.) Remember that it is no good to fix the errors in this HTML page, you have to fix the problem where it originally came from, e.g., the JSP page or from an include file or a style sheet.

d. **SQL**: in this lab, your java code will build a SQL statement that updates a record. The SQL must be syntactically correct and it must reference table names and field names exactly as you defined them in your database. If your SQL is not correct, your java/jsp program will throw an exception when you try to run it. You just have to be sure to display the exception error message, e.getMessage(), either on the JSP page (out.print) or in the glassfish log (System.out.print). Another technique is that you can type your proposed SQL command right into the query area at the top of MySqlWorkbench and try to execute it. MySqlWorkbench provides errors at the bottom, if there are any.
e. **JSP:** thankfully, the NetBeans editor gives you warning about any JSP syntax errors that you may be typing in. However, you can still get unexpected results or runtime errors. As mentioned before, your best option is to add `out.print` statements into your JSP pages – this will help you determine which code is being run and the values of certain variables. A couple of things to check in your JSP pages:

i. Check that your page posts to itself (action attribute of form references name of jsp page) – especially if you just changed the name of the jsp page.

ii. Check that when you do `request.getParameter("name_of_input_tag")`, you spell the name of the input tag exactly as you have it spelled below inside the `<form>` section of your HTML. If you misspelled the name of the input tag that you are checking for postback, your page will always act like first rendering.

iii. If you have a weird java compilation error that confuses you, click on your first left brace { and check to see it’s matching right brace }. Chances are that you are missing a brace or have an extra one. The weird error you are getting is actually in the servlet that is created from your JSP page.

f. **Java:** If you are debugging a java class, you cannot use `out.print()`, but you can use `System.out.print()`. I usually precede the message with stars so I can find the messages which appear at the bottom of the glassfish log in the output pane of NetBeans. For example,

```
System.out.println("exception in …, msg is: " + e.getMessage());
```

When you no longer need to debug some code, I recommend that you comment out the debug statement rather than delete it (who knows, you might need to debug again in the same place).