CIS 3308 (Kyvernitis) Update Lab

Overview:

In this lab, you will modify each of your three list pages (users.jsp, other.jsp, assoc.jsp) so that every row has an edit icon that links to an update form (passing in the id of the record to be updated). You will also create three new pages: updateOther.jsp, updateUser.jsp, and updateAssoc.jsp. These three pages will be similar to your insert pages.

Functional Requirements:

other.jsp, users.jsp, and assoc.jsp shall:

1. be modified such that they have an "edit" icon for each data row. The edit icon shall link to the related update page, passing in the id of the row. For example,

```
<tr><td><a href='updateUser.jsp?id=5'><img src='edit_Icon.png'></a></td> … other fields … </tr>
<tr><td><a href='updateUser.jsp?id=12'><img src='edit_Icon.png'></a></td> … other fields … </tr>
```

updateOther.jsp, updateUser.jsp, and **updateAssoc.jsp** shall:

1. Label and provide one text box for each non-key field in the related database table.
2. Label and provide an HTML select tag for every foreign key (such as user.role_id and the two foreign keys keys of your associative table). This select tag options shall be generated from the database (not hard coded). User selected values shall persist when the user clicks submit. "null" shall never appear on the page.
3. On first rendering, the text boxes shall be "pre-filled" with the values extracted from the database record that matches the given id (that was passed in the URL) and any select tags (that represent a foreign key value) shall have the correct option selected (e.g., selected='selected').

   Note: request.getParameter("id") should never be null. On first rendering, it should have been populated from the URL of the link from the list page. On first rendering, you use request.getParameter("id") to find the record from the database so you can pre-populate the fields on the form. On postback, the value of the (finally hidden) textbox named "id" should be persisted just like all other inputs. If request.getParameter("id") is ever null, then
   - there was a programmer error (between your list page linking to your update page, for first rendering), or
   - you forgot to persist it (for postback), or
   - someone is trying to hack your page by URL tampering.

4. When the user clicks submit,
   - All user entered values shall persist, even the id that was passed in on first rendering, and the selected options (from select tags).
   - Field level validation messages shall be displayed (validation messages that check for data type, length, and required vs. optional, as per your database design).
   - A form level message shall indicate:
     - "Please try again" (if there were any field level validation errors), OR
     - The a database error (user friendly preceding the technical message for known problems like duplicate value for unique constraint, or database unavailable - if there is any database error), OR
     - A message that the update was successful.

Handling nullable fields: Your database was required to have some nullable non-character fields (double check that you have met these requirements from the database labs).

- Just like the insert lab, if the user enters nothing for a nullable non-character field, the web application shall encode null into that field of the database record, in the insert/update statement.
- View code shall test for null and replace that with empty string (when building an HTML table from the result set from the database).
- We will test specifically for appropriate handling of nullable non character fields and deduct points if your database lacks the required nullable non-character fields and/or your web app is not handling these fields appropriately.

Design Specifications

1. **updateOther.jsp, updateUser.jsp,** and **updateAssoc.jsp** shall (as mentioned in the "Requirements for all Labs and Project"):
    a. Declare a database connection object, pass that to classes/methods as needed, and then close the database connection object when it is no longer needed.
    b. Have as little code as possible (delegating functionality, wherever possible, to reusable java classes). Note: a JSP page is the only place where you can use JSP implicit objects, so the JSP page must handle input (request.getParameter), output (out.print), session (get/put logon information), and security/redirect (response.sendRedirect).
2. Your web application shall have a **model package** with 3 sub-packages, such that each sub-package named the same as one of your database tables (not assoc, not other).
    a. Within **each model sub-package**, there is a StringData class and a DbMods class. (This should already be the case from your previous lab.)
3. The **StringData class** shall contain
    a. one String property (can be public) for each field in the associated database table - even fields that are not of type VarChar in the database, including primary key and foreign key fields.
        - In the last lab, it may not have mattered if your StringData class had a String property for the auto-increment id of your table, but in this lab, you will definitely need to be user that you have a field, e.g., String webUserId = "";
    b. one additional String property named something like "errorMsg" – to hold form level (or record level) validation error messages.
4. The **DbMods class** shall have an **update method** that
    a. accepts a StringData object, validates it and (if it passes validation) inserts it into the database.
    b. returns a StringData object that provides field level error messages and a form level message (error or success) – to the JSP page.
    c. invokes the same validation code as was called by insert functionality (last lab) – for good code reuse.

Labs Page (Blog)

Your labs page shall include a blog that explains what you learned and links to your work.

Programming Style

Adhere to the requirements listed in the section entitled "Requirements for all Labs and Project" on the class labs page.

Homework Submission

- After getting your code to work locally, publish it to cis-linux2 and test it.
- Submit a zip file of your whole project to blackboard.
- Make sure that you have a link from your labs page to a screen capture of your data model (that you created using mySqlWorkbench) – should be there from your database labs.

Suggested Approach

- If you have not already done the update activity, do that now by clicking here: http://cis-linux2.temple.edu/~sallyk/cis3308/08_update/updateActivity_pg.html

  Once you have completed the update activity, your other.jsp page would provide update icons for each "other" row. When clicked, the update icons redirect you to updateOther.jsp where the page (which looks very similar to insertOther.jsp) would have all its fields pre-populated with data of the clicked row. Your updateOther.jsp page should also persist all the data on the page upon submit (including the id field).

- Next create an update method, similar to your insert method, except for the SQL update statement which would look like this:

```
String sql = "UPDATE customer set email_address=?, pwd=?, first_name=?, "
        + " last_name=?, credit_limit=? WHERE customer_id=?";
```

- After getting other.jsp and updateOther.jsp to work, work on users.jsp and insertUser.jsp. For this, you would need to:
  - Blend in the <select> tag persistence code (from search and insert labs).
  - Enhance (or make an enhanced copy of) the method that spits out the HTML <table> tag.
  - Create a search method (returns populated StringData object when passed in an id).
  - Create an update method.
- Then, do the same for assoc.jsp and insertAssoc.jsp
- Remember to **use System.out.print() to debug.** Look for this output in the glassfish server log (in the output area at the bottom of the screen). If you have a MAC, you have to click on "Services" (instead of projects, upper right pane), then "Servers", then right click on "Glassfish" and select "View Domain Server Log".