

## CIS 3308 Lab: JSP Search

**Overview:** In this lab, you will create a page (named search.jsp) that shows records from your associative table (joined with your user table and your other table) – filtered by user entered search criteria. Your page will dynamically generate pick lists (<select> tags) based on current data as pulled from your database. User entered values (text boxes and pick lists) shall persist when the user clicks submit.

### Functional Requirements:

1. Your search page shall have at least two dynamically generated pick lists:
  - one pick list showing the "identifying column" (e.g., product description or name, unique) from your "other table".
  - another pick list showing the "identifying column" from your user table (e.g., user\_email, or other unique column you have in that table, not user id). If you wish, you can have a user nick name concatenated with the user\_email – this could be more helpful while still being guaranteed unique.

"Dynamically generated" means that if someone adds a new record to your other table or your user table, your generated pick list will have one more element in it. Your dynamically generated pick lists will also persist user selected values.
2. Your search pages shall have at least one more criteria from your associative table (in the example below, I put a date range, but you may do whatever is appropriate for your web application).
3. Once the user clicks the submit button, your page shall display only those records (from your associative table joined with your user table and your other table) that meet the user entered selection criteria. For any user selection criteria field where the user did NOT enter anything, your page shall do no filtering. So, if the user just clicks submit without entering anything into the form, the user will see all records (associative joined with your user and other table).

4. When your page displays the selected data, it shall show all the (non-key) columns from the associative table, as well as at least one non-key field from your user table and at least one non-key field from your other table ("key" meaning PK as well as FK). This data shall be displayed in a useful (per the application) and appealing fashion. Your page can display the data using the HTML <table> tag or it can use other HTML tags. If you choose to display the data in an HTML table and it is too wide, think of ways to combine data into a single HTML <td> (table data/cell), so that your users are not constantly being shown a horizontal scroll bar.

For example, each row (from your joined result set) could be broken down into two parts: lineHeader and lineDetail and you could use a bit of simple jQuery (see below) to allow the user to open up only the lines they want to see. This is what I have done in my 3308 labs page – only the lab titles show initially and you click on the lab title to open/close the lab details (View Source on the labs page to see how this works). If you do this (and so that it is easy for us to grade), the lineHeader area shall include the data that demonstrate that the search filters are working properly.

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
  $(document).ready(function () {
    $(".lineDetail").hide();
    $(".lineHeader").click(function () {
      $(this).next().toggle(); // hide if showing, show if hiding.
    });
  });
</script>
```

5. User entered values (text boxes and pick lists) shall persist when the user clicks submit. Here is an example of how your search page might look:

## SEARCH

Select Customer:

▼(customer pick list)

Select Product:

▼(product pick list)

Date Range (between):

low date

and

high date

submit

After submit, show your associative table joined with your user and other table here. If user picks nothing, show ALL records.

2

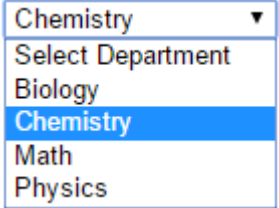
## Design Specifications:

1. Your **dbUtils** package shall have a new **class** (perhaps called "MakeTags") that has a **static method** that shall take the following **input parameters**:
  - a. DbConn object (open database connection)
  - b. Name you want the select tag to have (e.g., "dept" in the example below).
  - c. SQL select statement (String) that selects two columns from some table, the first column being an id and the second column being a description or name. Order the result set by the second column. Example: "SELECT dept\_id, dept\_name from department ORDER BY dept\_name". Note that you can extract data from a ResultSet using numbers instead of column names: **results.getString(1) returns 1<sup>st</sup> column.**
  - d. A string which contains the id of the element that should show as selected in the generated HTML code – this is the <option> that will have selected="selected" as an attribute. In the code example below, Chemistry (option value is "4") would show up in the pick list (before the user does anything) – because Chemistry is the option that has selected="selected" as an attribute.
  - e. Additional parameter(s) that indicate an <option> to be added before the rest of the entries that are taken from the database. We need to be able to identify when the user did not select a value from a pick list. In the HTML code below, we are talking about needing to add the first <option>:

`<option value="0" >Select Department</option>`

and **return** a String that contains the HTML code for a <select> tag.

Here is valid HTML code for a <select> tag (also called "pick list") and how it renders in the browser.

|   |  |
|---|--|
| <pre>&lt;select name="dept"&gt;   &lt;option value="0" &gt;Select Department&lt;/option&gt;   &lt;option value="3"&gt;Biology&lt;/option&gt;   &lt;option value="4" selected="selected" &gt;Chemistry&lt;/option&gt;   &lt;option value="2"&gt;Math&lt;/option&gt;   &lt;option value="1"&gt;Physics&lt;/option&gt; &lt;/select&gt;</pre> |  |
|---|--|

So, your MakeSelectTag might have a method header that looks like this:

```
public static String MakeSelectTag (DbConn dbc, String selectTagName, String selectedValue, String preOption)
```

and the JSP page might call MakeSelectTag like this (first rendering):

```
String selectTag = "";
// ...
selectTag = MakeSelectTag(dbc, "dept", "", "<option value='0'>Select Department</option>");
```

or like this (postback, when Chemistry is supposed to be pre-selected):

```
selectTag = MakeSelectTag(dbc, "dept", "4", "<option value='0'>Select Department</option>");
```

2. In your **view package**, your class that is named the same as your associative database table shall have a new **static method** named **search** that shall take the following **input**. (If the classes in your view package are not named the according to your database table names, rename /refactor them now and check that your 3 data display JSP pages still work.) This new search method shall have the following input **parameters**:
- DbConn object (open database connection)
  - As many other parameters as you have user criteria on your search form (all typed String)

and **return** a String that is HTML code (probably a <table> tag) that contains all the data selected from your associative table (joined with user and other, filtered by the user's criteria, as described above). If you prefer not to use the HTML <table> tag, that is fine, you can use something that's more creative. In any event, make sure that your layout is not so wide as to require horizontal scroll bar most of the time. To avoid the horizontal scroll bar with the <table>, combine some of your data into <td>s (table data or cell). For example, you could have something like this:

```
<td>address<br/>city, state, zip</td>
```

Here is an example of a SELECT STATEMENT (in your search method) that would work if all user criteria were filled in (for my example, using my database and the search form provided above):

```
SELECT * FROM web_user, purchase, product
WHERE web_user.web_user_id = purchase.web_user_id
AND purchase.product_id = product.product_id
AND web_user.web_user_id = ? AND product.product_id = ?
AND purchase.p_date > ? AND purchase.p_date < ?
```

Your code shall create the SQL statement with a variable number of parameters (?s) in a “linear fashion” and then replace these parameters also in a linear fashion. By saying “linear fashion”, we mean that if a form has 6 user criteria, the code shall have 6 if statements, one after the other – not a complicated set of nested if/else code that has 6 factorial blocks in it (representing all the combinations of criteria filled in or empty).

Since you do not know how many criteria you will find, initialize your SQL Select statement to have that part of the SELECT statement that is needed regardless of how many user criteria are filled in or not filled in (the blue part of the SELECT statement, above) -- something like this:

```
String sql = "SELECT * FROM web_user, purchase, product "+
" WHERE web_user.web_user_id = purchase.web_user_id " +
" AND purchase.product_id = product.product_id ";
```

Then, your code shall append the additional conditions to your WHERE clause, depending on which user criteria have been specified - in a “linear fashion”. Once the SQL SELECT statement is complete (with all ?s for all possible user criteria), your code shall create the PreparedStatement.

```
if (userId.length() != 0) {
    sql += " AND web_user.user_Id = ? ";
}
if (prodId.length() != 0) {
    sql += " AND product.product_Id = ? ";
}
//... other criteria ...
```

Then, your code shall “replace the ?s”, also in a “linear fashion”. It will need a counter to keep track of which question mark it is replacing – the second set of "linear if statements" (e.g., that replace the parameters) need to be presented in the same order as the first set of "linear if statements" (above).

```
int count=1;
if (userId.length() != 0) {
    stmt.setString(count,userId);
    count++;
}
if (prodId.length() != 0) {
    stmt.setString(count,userId);
    count++;
}
//... other criteria ...
```

### **Navigation Bar:**

- Your navigation bar shall include a link for "Search" that references your new search.jsp page.

### **Programming Style:**

For every lab, follow the requirements listed in the "For All Labs and Project" entry in the labs page. Important requirements include: data model requirements, code reuse, no database connection leaks, user friendly database error messages, error handling, separation of concerns, MVC, self-documenting names, code indentation/comments/white space, no unused code, labs page blog, and submission of zip file into blackboard.

### **Homework submission:**

- Add a blog entry on your labs page that tells what you did, what you learned, and links to the relevant page.
- Make sure that you have a link from your labs page to a screen capture of your data model (that you created using mySqlWorkbench). (Your web application proposed functionality should be clear from your home page.)
- 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.

## Suggested Approach:

1. Begin by making search.jsp, probably by “File -- SaveAs” from your assoc.jsp so that you start out with your site’s layout.
2. Make sure that your <body> has a <form> that posts to itself. Inside the form, put the textboxes for user criteria and a submit button. We will add the select tags later. Get the basic functionality working – just persisting the user entered input and never showing “null” on the form.

```
<form action="search.jsp" method="get">
    Date range between <input type="text" name="lowDate" value="<%=strLowDate%>"/> and
    <input type="text" name="highDate" value="<%=strHighDate%>"/><br/>
    <input type="submit" value="Click to Search" />
</form>
<% out.print(msg);%>
```

3. Next, work on your dynamically generated <select> tags. At first, you will be happy to just get a well-formed select tag that shows data that was selected from your database. If you do not see a pick list from the browser, compare what you see in your View Source to the example code above. Even if the select tag “looks OK”, View Source to be sure there are no syntax errors.

```
<%
String sql = "SELECT cust_id, cust_lname from customer ORDER BY cust_lname";
String custSelect = MakeSelectTag.makeSelect(dbc, sql, ...);
//...
%>
<form action="search.jsp" method="get">
    Customer: <%=custSelect%>"/> <br/>
    Date range between <input type="text" name="lowDate" value="<%=strLowDate%>"/> and
    <input type="text" name="highDate" value="<%=strHighDate%>"/> <br/>
    <input type="submit" value="Click to Search" />
</form>
```

This is not the final method, but it is a start and it does output a valid HTML <select> tag that you can view in your browser. Note: the “\n” adds new line that you can see only in “View Source” from the browser (helps with debugging).

```
public static String makeSelect(DbConn dbc) {

    String sql = "SELECT web__user__id, user__email FROM web__user ORDER BY user__email";
    String tagName = "user";
    String out = "\n\n<select name = '"+tagName+"'>\n";
    try {
        PreparedStatement stmt = dbc.getConn().prepareStatement(sql);
        ResultSet results = stmt.executeQuery();
        while (results.next()) {
            out += "    <option value='"+results.getInt("web__user__id")+"'>";
            out += results.getString("user__email")+"</option>\n";
        }
        out += "</select>\n\n";

        return out;
    } catch (Exception e) {
        return "Exception in MakeSelectTag.makeSelect(). Partial output: " + out
            + ". Error: " + e.getMessage();
    }
}
```

4. Next, work on the persistence of your <select> tags. For this, you will have to add input parameters to your MakeSelectTag method. The idea is that when you click submit, whatever you selected from the pick list remains as you selected it – it does not revert to the top option from the list.
5. Now you are ready to work on your search method of the class for your associative table in your view package. You can pattern your search method after the method you have from your data display lab (the method that outputs a HTML table full of data from your associative table joined with your user table and your other table). Starting out with this code, start adding input parameters (that represent the user’s criteria selections). You can start out with just one or two user criteria being passed in. You can start out with the criteria always being required. But eventually, you will need to implement the full functionality as described in the functional requirements and design specifications above.