CIS 3308 (Kyvernitis) Delete Homework

Overview:
In this lab, you will modify your web application so that it provides delete functionality wherever it lists your data. In other words, the user will be able to delete from your delete web user table, your “other” table, and your associative table.

Functional Requirements:
- Wherever data is listed (web_user, other, associative), your web application shall add an extra column to the HTML table. This extra column shall include an icon that, when clicked, calls a JS function to (1) invoke a delete API and (2) if successful, remove the clicked row from the HTML table.

![HTML Table]

- **This is not a functional requirement** but I wanted to show it next to the image above... Since the HTML table is built dynamically (by your JavaScript), the only place you can actually check to see if your <table> HTML is correct is by clicking on the Elements tab from the Chrome debugger (F12). Open the body (from the elements tab) and drill down into one of the data rows of the HTML table. For each row <tr>, you should see a cell <td> that has an <img> that has a click event that calls a JS function. The JS function call passes in (1) the id of the record to be deleted (for each row, this parameter should match the id in the first column) and (2) a reference to the image that was clicked (from the image, the JS function gets the clicked <tr> and deletes it).
Functional Requirements (continued):

- When the user clicks on one of the delete icons, JavaScript code shall ask the user for confirmation that they really want to delete. If so, the record should be both deleted from the database (accomplished by calling the appropriate delete API). If the database delete was successful, then the clicked row <tr> shall be deleted from the HTML table in the User Interface. In addition, the user interface shall be updated with a confirmation message like “Web User 23 was deleted” or “...some database error...” or “... the API call failed ...”, etc.

- Since no records point to your associative table, there should never be any FK constraint that kicks in preventing any delete. However, this is not the case for web_user records and records from your “other” table. If any associative record points to a web_user record or to an “other” record, the database is supposed to reject the deletion of the web_user or “other” record. Your web app shall provide user friendly error messages whenever a FK constraint prevents deletion. Message should be like “you cannot delete this user because s/he has purchases” or “you cannot delete this product because it has associated purchase records”. Your code shall intercept the database exception message (searching for substrings that indicate the error is related to a particular FK constraint) and replace the technical constraint error text with a user friendly message.

Design Specifications

1. In the webAPIs folder shall be three delete APIs
   a. deleteUserAPI.jsp (deletes one record from web_user, provided in sample code),
   b. deleteOther.jsp (deletes one record from whatever you named your “other” database table), and
   c. deleteAssoc.jsp (deletes one record from your associative table).

   Each WebAPI shall **extract the id of the record to be deleted** (from URL using request.getParameter) and then invoke the appropriate delete method that’s in a java class (see item below).

   Each Web API shall **output an error message** or “” (empty string) if all went ok. By sending empty string, we allow the HTML page decide how they want to let the user know that everything went fine.

   Each Web API (jsp page) shall **close the DbConn object** so that there are no database connection leaks.

2. Add a **delete method** to the DbMods class in each of your 3 packages model.webUser, model.xxx, & model.yyy where xxx and yyy represent your “other” table and your associative table. These methods shall be called by the associated Delete Web API. Each delete method shall take as input a DbConn object as well as the id of the record to be deleted. Each delete methods shall return an error message (String) or empty string “” if the delete happened correctly.

3. JavaScript files. Store all JS files in the js folder and follow this naming convention:

   **Js files shall be named the same as the single global function (or object) that is defined within the js file.**

4. Keep all the CRUD code for each database table in the same js file. See userCRUD.js in the sample code. The js file creates a global userCRUD object then invokes an IIFE, an Immediately Invoked (anonymous) Function Execution. This IIFE enhances that object by adding to it all the global functions you need for web_user CRUD, like list, startInsert, insertSave, etc. Just to complete this assignment, you could do it without having an IIFE (you
could just add the functions to the object without it, but if you add an IIFE (like the sample code), you’ll be set up nicely for the update homework (coming later) and you’ll be able to have local private functions that can be shared by update code that is similar to insert code.

5. Remove unneeded code: for example, 1_deleteTester.html was provided to help you accomplish the coding for this week’s homework, but it is not part of your web application, so it should be deleted before you publish and submit to Canvas.

Labs Page (Blog)

- Your labs page shall always include a blog for each lab, explaining what was easy, what was hard, and what were the important concepts that you learned.

Programming Style

- Follow what is listed under "Requirements for All Labs and Project" (on the 3308 labs page).
- Any JSP page that accesses your database will need one DbConn object. That JSP page should instantiate the object (constructor opens the connection), use it, and then be sure to close it (for EVERY CODE path). This will prevent your code from having any database connection leaks.

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).

Suggested Approach

Finish off User delete functionality

1. Tunnel in.

2. Run your list users Web API so you can see the data from your users table- keep this open in one tab so that you can refresh it as needed to verify whether or not your delete functionality is working.

3. The index page from your sample code looks like it works (it deletes the row from the HTML table), but it does not actually call the delete user API, so if you would refresh the page, all the deleted rows were never deleted. Finish off the code so that once the user confirms they want to delete, you call the delete user API and (only if it worked) then allow the code to run that deletes the row from the HTML table. Refresh the page after deleting, and make sure that the record does not come back to life.

4. Find the try/catch block where the database throws an exception when the user tries to delete a web_user record that is related to associative records. Try to delete a web_user record that has related associative records and note the Database exception message. Use some java code (search for substring within exception message string) – when you know it’s a FK constraint violation, replace the exception message with a user friendly message like “you can’t delete this customer because they have made purchases”, etc.

5. Once your web_user delete is working, you can close out the list User Web API tab.
**Associative delete:**

Next, work on the Delete Web API for your associative table. Since no database tables have FKs that point to your associative table, you should not have to worry about any FK constraint preventing any records from being deleted from your associative table.

1. Run your associative List API (JSP page) that lists all the records from your associative table – keep it open in its own tab and refresh it to verify if your delete associative functionality is working.

2. Copy from the user delete API to the associative delete API. Change the imported classes to reference your associative package instead of the webUser package.

3. Create a delete method in the DbMds class of your associative package. Link up your associative delete API with this delete method and start testing by using URL tampering on the associative Delete API.

4. Make a copy of 1_deleteTester.html (it will be temporary, call it what you like, perhaps deleteAssoc.html). Try to get this page to correctly invoke your associative delete API.

5. Make associative delete from your index page, similar to the user sample code. As you work on this, test to be sure that records deleted from the HTML table are actually deleted from the database (re-run the list functionality).

**Other delete:**

You should know how to work on the “other” delete, but I’ll summarize.

1. Start by creating the other Delete Web API and testing it using URL tampering in one tab and having another tab open that lists all the other records (to verify what’s been deleted).

2. Create a dummy HTML page (like 1_deleteTester.html) that deletes using client side code (HTML/javascript).

3. Once that works, incorporate your client side code so that it works from your index.html page.

4. Remember to keep all your other CRUD code in a single js file (probably named otherCRUD.js).

Don’t forget to add a blog for this week’s HW to your blog page.

After regression testing (all the code from all previous HWs), publish and regression test again from what is published.

Then submit your zip file into canvas.