

CS 328 - Homework 10

Deadline

Presentation of Problem 2 is due **during lab on Friday, April 22**;
the remainder of this homework is due by **11:59 pm on Sunday, April 24, 2016**

How to submit

Submit your files for this homework using `~st10/328submit` on nrs-projects, with a hw number of 10.

Purpose

To practice a bit with unobtrusive-style client-side JavaScript for form validation, to get practice with GUI design and to consider usability and accessibility in interface designs, to start a 4-screen bookstore application using the PL/SQL `sell_book` function, and to write a PL/SQL subroutine of use for your "additional" database.

Important notes

- Be sure to follow the unobtrusive-style client-side JavaScript coding guidelines discussed in class.
- **NOTE:** you are welcome to use `require_once` and `include_once` in your PHP documents! BUT when you use them in homework problems' documents, be sure to also SUBMIT copies of all files that you are requiring/including!
- Also note: I hope to have you present versions of some of these documents to the class at some point.
- Remember to follow the **CS 328 SQL and PL/SQL Coding Standards** as given in the CS 328 Homework 2 handout and the public course web site for all SQL and PL/SQL code.
- Make sure that you have executed the scripts `create-bks.sql` and `pop-bks.sql`.
- Remember to follow the **CS 328 HTML5 Coding Standards** as discussed in class and as given on the public course web site for all HTML5 documents.
- Remember to follow the **CS 328 CSS3 Coding Standards** as discussed in class and as given on the public course web site for all CSS3 that you write, also.
- Remember to follow the **CS 328 PHP Coding Standards**, as discussed in class and as given on the public course web site for all PHP documents.

Problem 1

For a *little* JavaScript-for-form-validation and modifying-the-DOM practice:

Make a **new** copy of your latest external CSS3 file `bks.css` in a **different** directory for Homework 10, since you *might* be modifying it for this Homework 10 and you don't want to accidentally change how it styles any previous homeworks' files.

Consider Homework 8 - Problem 3's `hand-insert-order-needed-8.php`. Copy it into a new file

`hand-insert-order-needed-10.php` in your Homework 10 directory (and change the URL in its opening comment to what it needs to be for me to run this new version).

Create, and have `hand-insert-order-needed-10.php` use, an external JavaScript `hw10Prob1.js` that:

- uses unobtrusive-style JavaScript to modify the `window` object only after the page has been loaded (see `postLab12.js` example posted with the Week 12 Lab examples)
- includes a function that validates the form on the 2nd screen in which the user enters the ISBN and the desired order quantity -- since the ISBN here is a drop-down, all it needs to do is verify that a non-zero, positive integer has been entered into the desired order quantity textfield.
 - what if the form doesn't validate? It should return the proper result to keep the form from being submitted, AND
 - ...it should modify the DOM in a clear, readable fashion of your choice to let the user know what they have done wrong.

Submit your resulting `hw10Prob1.js`, `hand-insert-order-needed-10.php`, and any other files needed for `hand-insert-order-needed-10.php` to work.

Problem 2

You will be presenting your work on this problem in lab on Friday, April 22. (Note that this presentation will also count as *part* of Week 13's lab exercise.)

Now, for something a little different: a **GUI design exercise**:

This assignment was adapted from a user interfaces course at New York University.

The following describes a number of components for carrying out a single task.

- Lay out the components in forms within **two or more** HTML screens, implemented with the help of PHP-based sessions from a `.php` document:
 - including `pizza` somewhere in the `.php` file's name,
 - including the URL for running your design in its opening comment,
 - making appropriate use of an external CSS for consistency between screens,
 - and somehow appropriately using sessions between the screens;
 - (The sessions aspect could be as simple as a greeting to the user on a subsequent screen, or something more intricate (perhaps summarizing one's pizza order on a final screen?))
- **Your goal**: to design these following the user interface design guidelines discussed in class.

Here are a few more ground rules:

- So as to not provide any hints regarding task flow, the components are arranged in alphabetical order.
- You cannot change the provided components or boilerplate,
 - ...but you are free to add graphical elements such as lines and borders to the layout, and to play with font attributes like bold and italic.
- You are also permitted to add punctuation to the boilerplate text as desired, if you think it would be

helpful;

– you may also add additional boilerplate, if you wish.

- Finally, you may add additional components for navigation between HTML screens.

Do not worry about whether you agree with the components I've chosen; just think of this as a **design** exercise.

The task: Ordering a pizza

- Boilerplate "Address" and text fields (for street address, city, and zip)
- A button labelled "Cancel"
- Boilerplate "Credit Card" and a text field (with room for the text "0000 0000 0000 0000")
- Boilerplate "Crust" and at least two radio buttons; include at least two crust options of your choice (e.g., "White" or "Whole Wheat", "Thin" or "Thick", "Crispy" or "Chewy", etc.). (Do not include more than five crust options.)
- Boilerplate "Expiration" and two drop-down lists, one for month (with names of months spelled out in full, e.g., "November") and one for years (e.g. "2014").
- Boilerplate "Last Name" and one text field
- A button labelled "Order"
- Boilerplate "Phone Number" and from one to three text fields, your choice (with room for the text (XXX) XXX-XXXX)
- Boilerplate "Price" and a text field (with room for the text "\$XX.XX") with the X's displaying the price of the currently selected pizza. \$00.00 or \$0.00 should be the initially-displayed price.
- Boilerplate "Quantity" and one text field
- Boilerplate "Size" and at least three radio buttons; include at least three pizza size options of your choice (e.g., "6 inch", "8 inch", or "12 inch", "18 inch", "21 inch", or "24 inch", etc.) (Do not include more than five size options.)
- Boilerplate "Total" and a text field (with room for the text "\$XX.XX") with the X's displaying the total cost of the order. \$00.00 or \$0.00 should be the initially-displayed total.
- Boilerplate "Sauce" and at least two radio buttons; include at least two pizza sauce options of your choice (e.g., "Red (Tomato Sauce)" or "White (Four Cheese)", "Marinara" or "Clam", etc.) (Do not include more than five sauce options.)
- Boilerplate "Toppings" and at least nine checkboxes; include at least nine pizza topping options of your choice (e.g., "Anchovies", "Bacon", "Broccoli", "Extra Cheese", "Mushroom", "Onion", "Pepperoni", "Peppers", "Sausage", "Tofu", "Zucchini", etc.) (Do not include more than fifteen toppings options.) You may also have subgroups of toppings within these if you wish (with appropriate subgroup headings).
- Note: you can only order **one kind** of pizza with this application, but you can order **any number** of this one kind of pie.

Submit all of your files implementing the above by the files deadline, and be prepared to demonstrate your design by the beginning of your lab on Friday, April 22.

Problem 3

You will be working on a slightly-larger application making use of the PL/SQL `sell_book` function over the next two homeworks. (You will be **demonstrating** this during the Week 14 Lab both as part of Homework 11 and as part of the Week 14 Lab Exercise.)

By **this** homework's deadline, you are expected to at least **complete** Screens 1 and 2 (they should be implemented and functional, and should work correctly), and Screen 3 should at least display (although it may display as little as "SCREEN 3 GOES HERE" by Homework 10's deadline).

Two additional requirements:

- include `sell_book` or `SellBook` somewhere in the filename of your "master" `.php` file
- ALSO include the URL for running your application-so-far in the opening comment of your "master" `.php` file.

General Requirements

- you are required to use HTML5 on the client-side, and some JavaScript as well
- you are required to make appropriate use of CSS to maintain consistency between the different screens; make your screens as attractive and easy to use as possible
- you will use PHP for the application tier, and input validation is required, of course.
- you are required to use sessions to pass information between the four screens -- and you are required to invalidate/terminate your sessions appropriately
- you are required to make appropriate use of PL/SQL stored function `sell_book` and possibly others we have developed previously for this scenario
 - you are permitted to write and use additional PL/SQL stored procedures/stored functions as you wish.
- your **boilerplate** (that's just a term for "hard-coded", non-dynamic text) should be spelled correctly.
- currency values should be displayed to two fractional places

Screen 1:

- include an appropriate title for your bookstore
- include a way for the user to enter an Oracle username and password (the password field should be of type "password"), using the `required` attribute appropriately to ensure that the form will not be submitted unless the user has entered something for the username and password
- include a submit button with the label "Log in"
- logging in should lead to SCREEN 2.

Screen 2:

- If the user types in an **invalid** username/password, **decide** which of these options you would prefer:
 - redirect back to SCREEN 1
 - OR (slightly-more-informative, but requires an extra click by the users to retry):

- show SCREEN 1A, which simply prints a "friendly" "that username/password combination didn't work" message of your choice with a "Back" button or link that takes you back to SCREEN 1 when it is pressed. This screen would be designed to be readable and user-friendly.
OR (more advanced, I think):
- redirect back to SCREEN 1 -- but that now also includes an eye-catching "friendly" "that username/password combination didn't work" message of your choice
- include an appropriate title for this 2nd screen for your bookstore.
- populate a `<select>` (drop-down box) element with ordered ISBN-and-their-title combinations from the bookstore database. Have it show at least 3 ISBN-and-their-title combinations at a time (that is, make use of the `<select>` element `size` attribute, which allows you to specify this).
- include a "Quantity sold" label and textfield, with contents initially 1.
- include a "Proceed" submit button, which leads to SCREEN 3, and an "Exit" submit button, which leads back to SCREEN 1.
- include unobtrusive JavaScript to ensure that the quantity entered is a non-zero positive integer before allowing this form to be submitted, modifying the DOM in a clear, readable fashion of your choice to let the user know what they have done wrong if this is not the case.

Screen 3:

- what must be the case, if you have reached this screen appropriately? If you determine that any of that is NOT the case, decide which of these options you would prefer:
 - redirect back to SCREEN 1
OR (slightly-more-informative, but requires an extra click by the users to retry):
 - show SCREEN 1A, which simply prints a "friendly" message of your choice describing the problem along with a "Back" button or link that takes you back to SCREEN 1 when it is pressed. This screen would be designed to be readable and user-friendly.
OR (more advanced, I think):
 - redirect back to SCREEN 1 -- but that now also includes an eye-catching "friendly" message of your choice describing the problem
- include an appropriate title for this 3rd screen for your bookstore.
- include elements populated with the ISBN selected from SCREEN 2, its Publisher name, its Quantity sold (as selected from SCREEN 2), its Title, its Author, its Price, the computed Subtotal for a sale of this quantity, the computed Tax for a sale of this quantity (use a reasonable non-zero tax rate of your choice), and the computed Total for a sale of this quantity, including tax. These must be attractively formatted.
 - the user should be able to modify **only** the Quantity to be sold at this point; they should not be able to modify any of the other values
 - (if the user wants to change which book is being sold, they need to use the Cancel button to return to SCREEN 2 (see below))
- the Price, Subtotal, Tax, and Total should all be displayed in Currency format to 2 fractional places
- include a Complete submit button that updates the database appropriately, using the `sell_book` PL/SQL

stored function, and then leads to SCREEN 4.

- include a Cancel submit button that leads back to SCREEN 2 without updating the database.
- include unobtrusive JavaScript to ensure that the quantity entered is a non-zero positive integer before allowing this form to be submitted, modifying the DOM in a clear, readable fashion of your choice to let the user know what they have done wrong if this is not the case.

Screen 4:

- what must be the case, if you have reached this screen appropriately? If you determine that any of that is NOT the case, decide which of these options you would prefer:
 - redirect back to SCREEN 1
 - OR (slightly-more-informative, but requires an extra click by the users to retry):
 - show SCREEN 1A, which simply prints a "friendly" message of your choice describing the problem along with a "Back" button or link that takes you back to SCREEN 1 when it is pressed. This screen would be designed to be readable and user-friendly.
 - OR (more advanced, I think):
 - redirect back to SCREEN 1 -- but that now also includes an eye-catching "friendly" message of your choice describing the problem
- include an appropriate title for this 4th screen for your bookstore.
- include something displaying a confirmation message of how many copies of the selected ISBN have been successfully sold.
- include an OK submit button that leads back to SCREEN 2.

Submit all of your files for your application-so-far by this homework's deadline.

Problem 4

Consider your "additional" database. As part of your CS 328 portfolio, it should have at least one PL/SQL stored function or stored procedure!

Decide on a PL/SQL stored function or stored procedure (your choice!) that you think would be a useful one for users (or application programmers!) using your "additional" database. You get to choose -- it can be as simple or as complex as you would like. However, NOTE that you will be adding an option to CALL this function to an extension of Homework 9 Problem 7's `custom-session2.php` as part of Homework 11.

(Maybe it returns a frequently-requested value, or a frequently-requested computation -- maybe it makes related changes to a couple of tables -- maybe it is a useful predicate (a function that returns true or false, like `is_on_order` or `pending_order_needed`). There are many possibilities!)

- In a file `328hw10.sql`, include an opening comment including at least your name, CS 328 - Homework 10, and the last modified date.
- Include a SQL*Plus `spool` command to spool the results of running this SQL script to a file named `328hw10-out.txt` (and don't forget to `spool off` at the END of `328hw10.sql`)
- Then include the definition for your PL/SQL stored function or stored procedure.

- Finally, include at least TWO demonstrations or tests that show that your stored function or stored procedure works -- be sure to include `prompt` commands that print out what the user SHOULD see if the demo-or-test has worked, before the script outputs the demo-or-test's result.

Submit your files `328hw10.sql` and `328hw10-out.txt`.

(By the way -- if you have updated any files related to your "additional" database -- especially `325design.sql` or `325populate.sql` -- please be sure to submit those updated files as well!)