

CS 328 - Homework 11

Deadline

Problem 4 (presenting something operational from Problem 3) is due **during lab on Friday, April 29**; the remainder of this homework is due by **11:59 pm on Sunday, May 1, 2016**

How to submit

Submit your files for this homework using `~st10/328submit` on nrs-projects, with a hw number of 11.

Purpose

To practice a little bit with XML and JSON, to complete and demonstrate your 4-screen bookstore application using the PL/SQL `sell_book` function, and to add an option to call Homework 10's Problem 4 PL/SQL subroutine and a little unobtrusive client-side JavaScript to your "additional" database application from Homework 9.

Important notes

- **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!
- Follow the **unobtrusive client-side JavaScript class coding standards** as discussed in class.
- Remember to follow the **CS 328 PHP Coding Standards**, as discussed in class and as given on the public course web site.
- 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.

Problem 1

Recall: We mentioned in class that PHP has a number of XML-related packages for dealing with XML, including one named `SimpleXML`. (There's way more about this than you need to know at <http://php.net/simplexml>.)

We saw that, if a file `play.xml` contains XML, you can obtain a PHP object version of that file's XML contents with:

```
$myXml = simplexml_load_file("play.xml");
```

Then you can obtain the contents of any element `elem` within `play.xml` with the expression :

```
$myXml->elem
```

And, if you want the first of several instances of an element, you'd use array notation to indicate which you want, and so on.

(And you can see an example of `SimpleXML` in action along with the Week 13 Lecture 2 posted examples.)

ASIDE REMINDER:

- also remember that PHP has a function `trim` that expects a string, and returns that string with any leading and trailing blanks removed.
- This is useful, here, because this package considers EVERY character, even white space, between the opening and closing tags of an element to be part of that element's content (!!).

1 part a

Create a well-formed XML document in a file whose name includes `hw11-1` in it somewhere and has the suffix `.xml`. The contents must be different from any of the in-class examples or the course textbook -- you decide the theme of your document, and also meet the following requirements:

- it should be well-formed XML
- it should include your name in it somewhere
- include at least one element with simple content
- include at least one element with element content
- include at least one element with mixed content
- include at least one element with empty content
- include at least one element with an attribute

1 part b

Now create a simple PHP document whose name includes `hw11-1` in it somewhere and has the suffix `.php` that:

- visibly includes your name somewhere in its body
- uses `SimpleXML` to load 1 part a's `.xml` file into a variable
- uses PHP expressions using this variable (as demonstrated above, or in a more complex way if you prefer) to display the contents of at least three elements from your `.xml` file from 1 part a in a tasteful way

Submit your resulting `.xml` and `.php` documents.

Problem 2

And, recall: We mentioned in class that PHP has a JSON extension. (There's more about this than you need to know at <http://php.net/manual/en/book.json.php>.)

We saw that, if you have a JSON string:

```
$myJSON = '{"sound": "moo", "volume": 13}';
```

...then you can get the value encoded in the JSON decoded into an appropriate PHP object using `json_decode`:

```
$phpVersion = json_decode($myJSON);
```

and now:

```
($phpVersion->{'sound'}) === "moo"
```

```
($phpVersion->{'volume'}) === 13
```

(And you can see an example of PHP's JSON extension in action along with the Week 13 Lecture 2 posted examples.)

2 part a

Consider your XML document from Problem 1 part a. Create a syntactically-correct JSON version of this same information in a file whose name includes `hw11-2` somewhere and has the suffix `.json`.

(I am not sure if there is just one correct way to do this -- I am curious to see how you decide to handle the different types of XML content in JSON notation.)

NOTE!!: PHP's `json_decode` function does NOT look kindly on comments included with JSON -- I've removed the ones I originally put at the beginning of Week 13 Lecture 2's example JSON files...

2 part b

Fun fact: PHP has a function `file_get_contents` whose simplest variant expects a string containing the name of a file, has the side effect of reading that file's contents, and returns them as a string.

Now create a simple PHP document whose name includes `hw11-2` in it somewhere and has the suffix `.php` that:

- visibly includes your name somewhere in its body
- uses PHP's `file_get_contents` function to read 2 part a's JSON into a PHP variable (as a string)
- uses PHP's JSON extension to decode this JSON string into a corresponding PHP object
- uses PHP expressions using this result (as demonstrated above, or in a more complex way if you prefer) to display the contents of at least three elements from your `.json` file from 1 part a in a tasteful way

Submit your resulting `.json` and `.php` documents.

Problems 3 and 4

Problem 3 is completing the 4-screen BOOKSTORE application you started in Homework 10 - Problem 3 (and whose overall description is also included here).

Problem 4 is displaying something operational for this problem in-lab on Friday, April 29; your Week 14 Lab Exercise grade will also be partially determined by this demonstration. During this presentation, you are expected to:

- speak clearly, not too quickly, projecting toward the audience
- demonstrate a successful sale of 2 copies of a book of your choice, explaining what you are doing as you do so

- also pointing out something you think is interesting about your application along the way (for example, this could be how you approached something, or a design choice you made)

Submit **all** of your files for your completed application.

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 completed application by this homework's deadline.

Problem 5

(Note: You will be **demonstrating** your final version of this problem's results during the Week 15 Lab both as part of Homework 12 and as part of the Week 15 Lab Exercise.)

First: consider your external CSS3 file `custom.css` from Homework 9. Make a **new** copy of this in a **different** directory, since you *might* be modifying it for this Homework 11 and you don't want to accidentally change how it styles Homework 9's problem using it (since that could affect your Homework 9 grade!)

NEW ADDITION 1:

Now, you will modify PHP document `custom-session2.php`, styled using `custom.css`, into `custom-session3.php`, whose second screen, giving the user a way to choose which action they would like to do, now **also** includes an option that will somehow lead to calling your PL/SQL stored procedure or stored function from Homework 10 Problem 4.

- This new action, once done, should still include allowing the user the options of either quitting (end the session, return to the login form) or to return to the choose-which-action screen.
 - and, optionally, you may have additional options (if appropriate/desired for your particular newly-added option).

NEW ADDITION 2:

Also: select at least one of your (non-login) forms in your application, and add unobtrusive client-side JavaScript to ensure that this form's user-entered value(s) are reasonable before allowing the 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.

- (it is fine if you wish to do so for all of your forms -- but it is only *required* that you do so for at least one of your forms)

Your application should continue to meet these specifications:

- This document should be one of those that can generate all of this application's pages, depending on the

keys that exist in the `$_POST` and `$_SESSION` arrays when it is reached.

- Each page that this document generates should include, in its body element, your name, CS 328, and the name of your additional database's scenario.
- Its initial form should require that the user enter JUST an Oracle username and password.
- USE external PHP helper functions for each page of your application, as demonstrated in `try-trio.php` and `dept-fun.php`! (and additional helper functions/require files are encouraged, also!) (and of course submit all of those files also!)

Submit **all** of your files for your completed application by this homework's deadline.