CS 318 - Homework 4

Deadline:

Due by 11:59 pm on Wednesday, February 27, 2013

How to submit:

Submit your files for this homework using `~st10/318submit` on nrs-projects, with a homework number of `4`

Purpose:

To practice some more with PL/SQL, to using CSS3 to lay out/format HTML5 forms, and to practice a little with unobtrusive JavaScript.

Important notes:

- In your JavaScript code, you are expected to indent the contents of all { }'s by at least 3 spaces, and each { and } should be on its own line, even with the preceding line (as seen in posted class examples).
  - Also, all JavaScript functions are expected to start with a comment that at least gives its name, and a purpose statement which explicitly describes what the function expects and what it does and/or returns.
  - We will also avoid the use of JavaScript within the body element of a page, in an attempt at unobtrusive-style JavaScript.
  - And, any HTML5 page that uses JavaScript is expected to include a noscript element within its body element that warns a user displaying it within a browser that does not support JavaScript.
- Remember to follow the **CS 318 SQL and PL/SQL Style Standards** as given in the CS 318 Homework 1 handout for all SQL and PL/SQL code.
- Make sure that you have executed the scripts `create-bks.sql` and `pop-bks.sql`, and that the bookstore tables are successfully created and populated.
- Unless explicitly indicated otherwise, for the entire semester, all web pages submitted are expected to use "strict" style HTML5, as discussed in class and in the course textbook.
- Likewise, unless explicitly indicated otherwise, all web pages submitted are expected to include the link to the W3C experimental HTML5 validator as well as the link to http://lint.brihten.com/html/ as shown in example page `html5-template.html`, and all must validate/pass the tests from both. Each page that does not will cause a loss of points on the problem involved.
  - If a page also uses CSS, it must include the image-link to the W3C CSS3 validator as shown in the now-updated `html5-template.html`, and it must validate as valid CSS level 3. Each page using CSS that does not will cause a loss of points on the homework problem involved.

- I'm not requiring specific indentation for HTML5 yet - I reserve the right to do so, however, if necessary. In the meantime, find a readable way of indenting it, and consistently do so...
- However, for **CSS rules**, you are **expected** to indent the contents of all { }'s by at least 3 spaces, and each { and } should be on its own line, lined up with the beginning of the selector (as seen in posted class examples).

The Problems:

Problem 1

Create a SQL script `318hw4.sql`, and start it off with comments including your name, `CS 318 - Homework 4`, and the last-modified date.

Next, add the command to run the `pop-bks.sql` script each time this script is run, so that you have "fresh", original versions of these tables. (Their contents are mucked with below, so it is important that these are "reset" here.)

Include the command to `set serveroutput on`, followed by a SQL*Plus `spool` command to spool the results of running this SQL script to a file named `318hw4-out.txt` . Then write a SQL*Plus prompt command that says `problem 1` . (You may add additional `prompt` commands around this to make it more visible, if you would like.)

Now, for a more interesting PL/SQL stored function that involves some exception handling: design and write a PL/SQL stored function `sell_book` that will represent the sales transaction of selling one or more copies of a particular single book. So, you will not be shocked to hear that `sell_book` expects two parameters (in this order): an ISBN representing the book being sold, and an integer representing the quantity being sold.

Then `sell_book` returns an integer representing a results code, letting the caller know if the sales transaction for this book was successfully completed. We'll describe its possible values further below.

`sell_book`'s purpose is to manage the database fields relating to the inventory of this ISBN. Here are its tasks (assume they are based on this scenario's "business rules"):

- reduce the `qty_on_hand` field of the `title` table for this ISBN by the number of copies being sold
- determine if we need to note that an order is now needed for this ISBN (because of this sale):
  - when is it needed?
  - It is NOT needed yet if the `qty_on_hand` for this title is larger than that title's `order_point`; the stock is not low enough, yet.
  - It is NOT needed at this point if the `qty_on_hand` for this title is less than or equal to that title's `order_point`, but it is already on-order.
  - And it is NOT needed if the `qty_on_hand` for this title is less than or equal to that title's `order_point`, it is NOT on order yet, but it DOES have a pending `order_needed` row already.
  - ...so, it is ONLY needed if the `qty_on_hand` for this title is less than or equal to that title's order_point, it is NOT on order yet, and it does NOT have a pending `order_needed` row

already...! (whew!)

  - be sure to make appropriate use of `is_on_order` (from Homework 1, Problem 7, whose example solution is available on the course Moodle site if you need it...) and `pending_order_needed` (from Homework 2, Problem 2) in determining this;

- only if it IS needed, then, should `sell_book` call `insert_order_needed` (from Homework 2, Problem 1) appropriately to make an entry into the `order_needed` table.
  - Use the ISBN from the ongoing transaction;
  - use the `order_qty` from the `title` table for this ISBN as the value of the `order_qty` attribute of the `order_needed` table.

BUT, of course, there's always the chance that `sell_book` might receive inappropriate arguments. It should protect against these problems:

- an ISBN that doesn't exist in the title table. (Let the *system* raise this `NO_DATA_FOUND` exception; your procedure should merely be able to handle it.)
  - `sell_book` should return a results code of -1 in this case.
  - Make sure any changes made up to this point by this procedure get un-done. (This is a transaction, after all...)
- a value for the number of copies being sold that is not greater than zero. (Raise this exception yourself: a user-defined exception.)
  - `sell_book` should return a results code of -2 in this case;
  - again, you should make sure any changes made up to this point get un-done.
- a value for the number of copies being sold that is greater than the current `qty_on_hand` for this ISBN. (Raise this exception yourself, also: another user-defined exception.)
  - `sell_book` should return a results code of -3 in this case;
  - any changes made by `sell_book` up to this point should be un-done.
- handle any other exceptions that occur, returning a results code of -4 in this case, and un-doing any changes made by `sell_book` up to this point. (This is purely defensive coding; such an exception will probably not actually be raised.)
- If **no** exceptions are raised, return a results code of 0. The caller can look at the returned results code value to see if his/her book sale transaction succeeded or not.

This is a transaction -- don't forget to commit your database updates. Remember that we have an atomic transaction here, so the collection of database updates should be committed all together or not at all. It would be wise then, I think, to start this function with a `commit`. And then this function should have another `commit` statement after (if!) all the database interactions have *successfully* completed. (And what statement does this suggest should be included in each exception handler within the exception section??)

To vigorously test this takes quite a bit of testing code; you'll find the code for testing this in a posted file `prob1-test-code` accompanying this homework handout. Paste this code after your code for the above procedures, and be sure to inspect your `318hw4-out.txt` file results carefully to see if the tests

passed.

And, as always, you may add additional testing calls if you would like.

Follow all of this with a spool off command; submit your files `318hw4.sql` and `318hw4-out.txt`

Problem 2

Consider your HTML5 page `bks-splash.html` and the external CSS3 file `bks.css` from Homework 3, Problem 7.

Make new copies of these files in a **different** directory, since you will be modifying them and you don't want to change Homework 2's or Homework 3's versions.

For this homework's version, the following changes should be made:

- Modify the HTML5 comment in `bks-splash.html` containing the URL I can use to view your `bks-splash.html` from your nrs-projects account to reflect this version's new directory. (Note that, for full credit, this URL must successfully display Homework 4's version of this page when I paste it into a browser.)
- If necessary (it might not be, depending on how you wrote it), modify the link element for `bks.css` so that the new version is used.
- `bks-splash.html`'s link to `req-order-status.html` should lead to Homework 4's version of that file (modified in Problem 3 below).
- `bks-splash.html`'s link to `insert-o-needed.html` should lead to Homework 4's version of that file (modified in Problem 4 below).
- Add rules to `bks.css` to nicely lay-out the form on `bks-splash.html` (and make changes as needed to `bks-splash.html` to use the new rules).

Your resulting `bks-splash.html` and `bks.css` files are not quite ready to submit yet (unless you want to submit "partial" versions early on).

Problem 3

Consider your HTML5 page `req-order-status.html` from Homework 3, Problem 7.

Make a new copy of this page in this homework's directory, so you don't change Homework 3's version.

For this homework's version, the following changes should be made:

- Modify the HTML5 comment containing the URL I can use to view your `req-order-status.html` from your nrs-projects account to reflect this version's new directory. (Note that, for full credit, this URL must successfully display Homework 4's version of this page when I paste it into a browser.)
- If necessary, modify the link to `bks.css` so that the new version is used.
- Do your modifications to `bks.css` from Problem 2 "work" for laying out `req-order-status.html`'s form? If not, add or modify `bks.css` to nicely lay-out the form on this page (and make changes as needed to `req-order-status.html` to use the new rules).

Your resulting `req-order-status.html` file is probably ready to submit, although the `bks.css` file is not quite ready to submit yet (unless you want to submit a "partial" version early on).

Problem 4

Consider your HTML5 page `insert-o-needed.html` from Homework 3, Problem 7.

Make a new copy of this page in this homework's directory, so you don't change Homework 3's versions.

For this homework's version, the following changes should be made:

- Modify the HTML5 comment containing the URL I can use to view your `insert-o-needed.html` from your nrs-projects account to reflect this version's new directory. (Note that, for full credit, this URL must successfully display Homework 4's version of this page when I paste it into a browser.)
- If necessary, modify the link to `bks.css` so that the new version is used.
- Do your modifications to `bks.css` from Problems 2 and 3 "work" for laying out `insert-o-needed.html`'s form? If not, add or modify `bks.css` to nicely lay-out the form on this page (and make changes as needed to `insert-o-needed.html` to use the new rules).

Your resulting `insert-o-needed.html` file is probably ready to submit, although the `bks.css` file is not quite ready to submit yet (unless you want to submit a "partial" version early on).

Problem 5

Consider your HTML5 page `order-info.html` from Homework 3, Problem 7.

Make a new copy of this page in this homework's directory, so you don't change Homework 3's versions.

For this homework's version, the following changes should be made:

- Modify the HTML5 comment containing the URL I can use to view your `order-info.html` from your nrs-projects account to reflect this version's new directory. (Note that, for full credit, this URL must successfully display Homework 4's version of this page when I paste it into a browser.)
- If necessary, modify the link to `bks.css` so that the new version is used.
- Add or modify the rules in `bks.css` so that this page's table is formatted and laid-out nicely and readably.

Your resulting `order-info.html` file is probably ready to submit, and now probably your `bks.css` file is also ready to submit.

Problem 6

Finally -- back to `bks-splash.html`! You will now practice a little with JavaScript.

- Write an external JavaScript `ck-login.js` that contains a JavaScript function that expects nothing and returns `true` if there is something contained within both username and password fields whose `id` attributes have the value that those fields just happen to have in `bks-splash.html`, and returns `false` otherwise.
- Modify `bks-splash.html` so that, on submit, its form's data is only submitted to the web server if

that JavaScript function returns `true` at that point.

  - how should your JavaScript let the user know that something is awry if that JavaScript function returns `false`? You can either use an alert popup to let him/her know, or if you want to find some other means of doing so (such as inserting a paragraph or header element into your page, for example), you may.

- For full credit, use unobtrusive-style JavaScript for this.

Now both `bks-splash.html` as well as `ck-login.js` should be ready to submit.