

CS 328 - Homework 5

Deadline

Due by 11:59 pm on **Sunday, March 6, 2016**

How to submit

Submit your files for this homework using `~st10/328submit` on nrs-projects, with a homework number of 5.

Purpose

To set up a PL/SQL stored function for future use, and to practice formatting & laying out forms using CSS3.

Important notes

- Note: you *may* be presenting versions of some of these HTML5 pages to the class at some point.
- None of the problems below happens to involve styling your `index.html` on nrs-projects with an external CSS - but you may certainly do so, if you'd like! It would be good additional practice.
- 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`, and that the bookstore tables are successfully created and populated.
- 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.
- And, now, also 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

Create a file `328hw5.sql`.

(Make sure that you have a copy of `pop-bks.sql` in the same directory as `328hw5.sql` -- it is called in this problem's testing script, to make sure the tests are run on "fresh", original versions of these tables, before the tests muck with them. Note that this script happens to already end with a `commit;` command.)

Start your `328hw5.sql` file with the following:

- comments containing at least your name, CS 328 - Homework 5, and the last-modified date
- 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 `328hw5-out.txt`
- followed by a `prompt` command including your name

- followed by a prompt command that says `problem 1.` (You may add additional prompt commands around this to make it more visible, if you would like.)

Be sure to `spool off` at the end of this script (after your statements for this problem).

Now, for a PL/SQL stored function that makes use of several earlier stored subroutines and 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 3, Problem 2, whose example solution is available on the course Canvas site if you need it...) and `pending_order_needed` (from Homework 4, Problem 2, whose example solution is likewise available) in determining this;
- only if it IS needed, then, should `sell_book` call `insert_order_needed` (from Homework 4, Problem 1, whose example solution is likewise available) appropriately to make an entry into the `order_needed` table.
 - Use the ISBN from the ongoing transaction;
 - use the `auto_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 function 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 function get un-done. (This is a transaction, after

all...)

- a value for the number of copies being sold that is not greater than zero. (FOR SOME ADDITIONAL EXCEPTION HANDLING PRACTICE, raise this exception yourself, as 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. (AGAIN, FOR SOME ADDITIONAL EXCEPTION HANDLING PRACTICE, 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??)

Separately (outside of `328hw5.sql`) run the posted testing script `sell_book_test.sql` posted along with this homework handout, make sure the tests all pass, and submit the file it spools with the test results, `sell_book_test_out.txt`, along with your files for this homework.

You may also add additional testing calls along with your `sell_book` code in your `328hw5.sql` script if you would like.

Make sure that you submit your `328hw5.sql` and `328hw5-out.txt` files as well as the `sell_book_test_out.txt` file.

Problem 2

Consider a rather typical plain HTML5 form, in `328hw05-before.html`, adapted from: <http://srv13.fountainheadcollege.com/mustafa.eminoglu/ws201/registration.html>

Copy this example into a file `328hw05-after.html` and modify it as you like with an external style sheet named `hw5-after.css` (whose link element should **follow** that for `normalize.css`). Meet the following specifications:

- Include comment(s) including at least your name and the last-modified date in **BOTH** your `328hw05-after.html` and `hw5-after.css`.
- Add your name within the `body` element of `328hw05-after.html` in some **visible** fashion of your choice.
- Use CSS3 to tastefully, attractively, and readably format and layout this document, including its form.

- Remember that your course textbook describes many interesting properties in Chapters 3 and 4.
 - You may add appropriate attributes to elements in `328hw05-after.html`.
 - You **may** remove some of the `
` elements currently within this form -- try to avoid adding any additional `
` elements.
 - Do **not** remove any of the existing form controls; do **not** change any of existing visible text.
 - At least one rule should style one or more aspects of an element's font in some visible way.
 - At least one rule should style one or more aspects of some element's color in some visible way.
 - At least one rule should use the `float` property in a visible and attractive way.
 - At least one rule should set at least one of an element's padding values in a visible and attractive way.
 - At least one rule should set at least one of an element's margin values in a visible and attractive way.
 - At least one rule should add an attractive border to an element.
- When you are satisfied with them (**AFTER** debugging and validating!!),
submit your resulting `328hw05-after.html` and `hw5-after.css` files,
and **then "hide" them** by changing their permissions to make them **NOT** world-readable:

```
chmod 600 328hw05-after.html
```

```
chmod 600 hw5-after.css
```

Problem 3

Consider your HTML5 document `bks-splash.html` and your external CSS3 file `bks.css` from the end of Homework 4.

Make **new** copies of these in a **different** directory, since you will be modifying them and you don't want to change Homework 4's version of them (since that could affect your Homework 4 grade!)

Add a link from your `index.html` on nrs-projects to this new version of `bks-splash.html` (making clear somehow this is **Homework 5's** version, and NOT removing the earlier links to Homeworks 3's and 4's versions!!)

Now consider these new versions of `bks-splash.html` and `bks.css`. Add rules to `bks.css` to "nicely" (tastefully, attractively, and readably) lay out and format `bks-splash.html`'s login form, of course also making any needed changes to `bks-splash.html` along the way. Feel free to also add additional formatting to this document as desired (especially formatting making use of the CSS box model, but you are not limited to that!).

Submit your file `bks-splash.html` and, eventually, `bks.css` (but this `bks.css` is not quite ready to submit yet, unless you want to submit a "partial" version early on).

Problem 4

Consider your HTML5 document `insert-ord-needed.html` from the end of Homework 4.

Make a **new** copy of `insert-ord-needed.html`, also, in the same directory as your new `bks-splash.html` from Problem 3, so that you again will not change Homework 4's version of `insert-`

`ord-needed.html`.

Now make whatever additions and modifications are needed to `bks.css` and `insert-ord-needed.html` to "nicely" (tastefully, attractively, and readably) format `insert-ord-needed.html`'s form. Feel free to also add additional formatting for this document as desired (especially formatting making use of the CSS box model, but you are not limited to that!).

Submit your files `insert-ord-needed.html` and `bks.css`.

Problem 5

Consider your HTML5 document `hw4-play.html` and your external CSS3 file `hw4-play.css` from Homework 4.

Make **new** copies of these named `hw5-play.html` and `hw5-play.css`.

Add rules to `hw5-play.css` to "nicely" (tastefully, attractively, and readably) lay out and format `hw5-play.html`'s table AND form, of course also making any needed changes to `hw5-play.html` along the way. Feel free to also add additional formatting to this document as desired (especially formatting making use of the CSS box model, but you are not limited to that!).

Submit your files `hw5-play.html` and `hw5-play.css`.