CS 328 - Homework 4

Deadline

11:59 pm on **Sunday, February 18, 2024**

Purpose

To get more practice with HTML form widgets, and to practice more with PL/SQL procedures, also adding in parameters, basic `if` statements, and its version of a `while` loop.

How to submit

Each time you wish to submit, submit your files using `~st10/328submit` on nrs-projects, with a homework number of `4`.

Important note: It is quite likely that your PL/SQL files will be in a different directory than your HTML files. That's fine, and preferable!

- Just remember that you need to run `~st10/328submit` from **EACH** directory with files to be submitted for Homework 4.

Homework 4 Requirements/Set-up

- For this homework's problems, do **not** include any CSS **except** for:
  - the external CSS `normalize.css` included in `html-template.html`
  - (optionally) Week 3 Lab Exercise's minimal external CSS mostly for `table` formatting, `lab3-table.css`
    - To use this, place this element at the END of your `head` element, right before the `head` element's closing tag, right AFTER the `link` element for `normalize.css`:

```
<link href="https://nrs-projects.humboldt.edu/~st10/styles/lab3-table.css"
      type="text/css" rel="stylesheet" />
```

- For an `img` element, note that it needs to validate as strict-style HTML. If its URL does not do so, make a copy of the image in your nrs-projects account (if you can legally do so) or use a service such as such as tinyurl to avoid problematic characters.
- Make a sub-directory in your `public_html` directory for Homework 4's HTML document. And, in this case, **you** choose the name for this sub-directory.

```
cd ~/public_html          # make sure you are in your public_html
mkdir name-you-choose     # make a directory within public_html
chmod 711 name-you-choose # make it world-executable
cd name-you-choose        # go to that new subdirectory
```

Remember that a world-readable file *my-doc.html* in the `public_html` subdirectory *name-you-choose* would have the URL:

`https://nrs-projects.humboldt.edu/~`*`your_user_name`*`/`*`name-you-choose`*`/`*`my-doc.html`*

- (Note: it is also perfectly fine if you choose to put your Homework 4 files in a "deeper" sub-directory within `public_html`.)

- You get to choose where your PL/SQL files go on nrs-projects -- but make sure these files are not readable by anyone but you. (That is, give your `328hw4.sql` file permissions of `600` (`rw-------`):

```
chmod 600 328hw4.sql
```

Problem 1

The purpose of this problem is to give you yet another chance to practice with form widget elements.

Starting from the **`html-template.html`** posted on the course public site and along with this homework handout, create a strict-style HTML document that meets the class style standards as well as the following requirements:

- Include **`prob1`** somewhere in its file name, and give its file name the suffix **`.html`** .
- Fill in the opening comment block as specified, putting in your **name**, the last modified **date**, and the **URL** that can be used to run your document.
  - (You *will* lose some credit if this URL does not work when I or the grader paste it into a browser!)
- You get to choose the theme(s)/topic(s) for your form, but your form's content may not be identical to that from in-class examples or your Week 4 Lab Exercise. (The idea here is to get *additional* practice.)
- Give the **`title`** element appropriate descriptive content.
- Include an appropriate **`h1`** element.
- Include a **`form`** element that meets the CS 328 class style standards, that also meets the following requirements:
  - It should have an **`action`** attribute whose value is a "real" URL of your choice (because we haven't gotten to writing an actual application program to handle this form yet).
  - It should have a **`method`** attribute whose value is **`"get"`**.
  - It should contain a top-level **`fieldset`** element that contains an appropriate **`legend`** element of your choice -- and within this top-level `fieldset` should be the following (not necessarily in this order):
    - (you may include additional nested `fieldset` elements around parts of the content below as you would like)
    - an **`input`** element of **`type="number"`** that is required to be completed (using attribute **`required="required"`**), with a logically-related **`label`** element
    - at least **3 checkboxes**, each with a logically-related **`label`** element
    - at least **3 logically-grouped radio buttons**, exactly **one** of which is initially selected, each

with a logically-related **`label`** element

  - at least one **`select`** element containing at least **five** options, set up so the user can only select one item at a time, with a logically-related **`label`** element
  - at least one **`textarea`** element, with a logically-related **`label`** element
  - at least **one other form widge**t discussed in zyBooks Chapter 2 but **not** yet required in a homework or lab exercise, with a logically-related **`label`** element
  - (you may add additional form widgets if you would like)
  - the last form widget in your form's fieldset should be an **`input`** element with **`type="submit"`** (which does *not* need a logically-related `label` element).

- Include **your last name** within a **`p`** element that you add to the **`footer`** element.

Reminder: for this homework, you may not use any CSS to style this form, and we'll never use the `table` element to format a form element, either. However, it appears that you can use `fieldset` elements, `p` elements, and instances of the void element `br` and still have it successfully validate as strict-style HTML.

Try filling out and submitted your form, guessing what name=value pairs should appear at the end of your `action` attribute's URL when you submit it, and see if they do.

Make sure an `.xhtml` copy of your document validates as strict-style HTML, and submit your resulting **`.html`** document. (Highly recommended: validate your form-in-progress frequently as you are creating it, and do **not** wait until you have completed attempts at all of its parts. Likewise, submit partial in-progress versions of your `.html` file through the week.)

Set-up for PL/SQL Problems 2-4

Create a file **`328hw4.sql`**. Give this file permissions of **`600`** (`rw-------`) by typing this at the nrs-projects prompt:

```
chmod 600 328hw4.sql
```

Start this file with the following:

- comments containing at least your name, **`CS 328 - Homework 4 - Problems 2-4`**, 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 **`328hw4-out.txt`**
- followed by a **`prompt`** command including your name

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

ASIDE: Basic PL/SQL Parameters

The most basic PL/SQL **parameters** are declared similarly to those in C++/C/Java, except:

- As for PL/SQL local variable declarations, you write the parameter *name* and **THEN** the parameter *type*.
- The parameter type must be **unconstrained** -- this means that, if the PL/SQL type can be followed by a set of parentheses with info within constraining the type, you may NOT put those parentheses.

For example, if you wanted a PL/SQL stored procedure that expected an item's name and quantity, returns nothing, and queries its price and prints to the screen the cost for that many of that item, that procedure heading could be written like this:

```
create or replace procedure print_cost(item_name varchar2,
    quantity integer) as
```

(Note, then, that while a *local variable* of type `char` or `varchar2` typically *NEEDS* to have a size specified, a *parameter variable* of those types must *NOT* have a size specified! And the same will be the case with the `number` and `decimal` types as well.)

Problem 2

As a warm-up to try out a PL/SQL parameter, in your PL/SQL script `328hw4.sql`, write a PL/SQL stored procedure **`num_pub_titles`** that:

- expects the name of a publisher,
- prints to the screen a tasteful message including both the name of the publisher and the number of titles the bookstore carries from that publisher
- returns nothing (since it is a procedure!)

(Note: by "number of titles", I mean the number of different titles, not how many copies of those titles. That is, if the bookstore carried just the titles "How to Moo" and "How to Baa" published by Tuttle Press, then the bookstore carries 2 titles from Tuttle Press, no matter how many copies of each it currently has in stock.)

Here are additional requirements for this problem:

- Look in the posted SQL script **`hello.sql`** at the version of the stored procedure **`hello_world`** that we created during class on Wednesday -- after class, I added an opening comment block for that procedure.
  - Create an opening comment block for your procedure that has a **`procedure:`** part and **`purpose:`** part in the same style that you see here. (You don't have to give an `examples:` part, but you can if you wish.)
  - Follow that with the PL/SQL code creating your procedure.
- Remember to follow your PL/SQL procedure with:

  `/`

  `show errors`
- Then put a comment saying you are about to **test** your procedure `num_pub_titles`.
- Follow that with at least **TWO** tests of `num_pub_titles`, **EACH** including:

- **`prompt`** command(s) stating that you are about to test `num_pub_titles` and **describe** what you should see if it is working properly.
  - (Your description should be specific enough that someone looking just at the spooled output can tell if the test passed or not.)
- Then write a SQL*Plus command calling `num_pub_titles`.

ASIDE: Basic PL/SQL `if` statement

Here's PL/SQL's basic `if` statement:

```
IF bool_expr THEN
    statement;
    ...
    statement;
ELSE
    statement;
    ...
    statement;
END IF;
```

Note that:

- You must put a keyword **`then`**!
- You **don't** have to put parentheses around `if`'s boolean expression (although you can if you wish!).
- It **must** end with an **`end if;`**

We'll talk in class about the oddness that is `ELSIF` -- but, you won't need that for this homework's procedures.

Problem 3

Sad fact: SQL*Plus does not handle PL/SQL `boolean` values gracefully!

So, both to practice writing a PL/SQL `if` statement and to make a little procedure for use in Homework 5, in your PL/SQL script `328hw4.sql`, write a PL/SQL stored procedure **`print_test`** that:

- expects a testing message and a `boolean` expression
- prints to the screen:
  - the given testing message,
  - followed by a colon and space,
  - followed by `TRUE` if the given `boolean` expression's value is `true`,

    or `FALSE` if the given `boolean` expression's value is `false`.
- returns nothing (since it is a procedure!)

So, for example:

```
exec print_test('Should see TRUE', true)
```

...should cause this to be printed to the screen:

```
Should see TRUE: TRUE
```

Here are additional requirements for this problem:

- Create an opening comment block for your procedure that has a **`procedure:`** part and **`purpose:`** part. (You don't have to give an `examples:` part, but you can if you wish.)
  - Follow that with the PL/SQL code creating your procedure.
- Remember to follow your PL/SQL procedure with:

  `/`

  `show errors`
- Then, put a comment saying you are about to **test** your procedure `print_test`.
- And, put in `prompt` command(s) saying that you are about to test `print_test`.
- But -- to then provide at least TWO tests of `print_test`, you should be able to just call it twice, being sure to use first arguments that describe what should be seen if your procedure is working!

Problem 4

Consider Homework 3, Problem 5, parts b and c.

For more practice with PL/SQL parameters and an `if` statement, in your PL/SQL script `328hw4.sql`, write a PL/SQL stored procedure **`title_info`** that:

- expects an ISBN,
- if a title with that ISBN exists in the title table, it prints to the screen a tasteful message including the title name, its quantity on hand, its order point, and its auto order quantity,

  otherwise it prints a tasteful message saying that the bookstore has no title with that ISBN
- returns nothing (since it is a procedure!)

Here are additional requirements for this problem:

- Create an opening comment block for your procedure that has a **`procedure:`** part and **`purpose:`** part. (You don't have to give an `examples:` part, but you can if you wish.)
  - Follow that with the PL/SQL code creating your procedure.
- Remember to follow your PL/SQL procedure with:

  `/`

  `show errors`
- Then put a comment saying you are about to **test** your procedure `title_info`.
- Follow that with at least **TWO** tests of `title_info`, (including at least one for an ISBN that **does** exist in your `title` table, and at least one for an ISBN that does **not**), **EACH** including:

- **`prompt`** command(s) stating that you are about to test `title_info` and **describe** what you should see if it is working properly.
  - (Your description should be specific enough that someone looking just at the spooled output can tell if the test passed or not.)
- Then write a SQL*Plus command calling `title_info`.

Make sure you turned spooling off at the end of `328hw4.sql`, and submit your **`328hw4.sql`** and **`328hw4-out.txt`**.