CS 328 - Homework 4

Deadline

11:59 pm on **Friday, February 20, 2026**

Purpose

To check your understanding of some PL/SQL basics, to get more practice with HTML form widgets, and to practice more with PL/SQL stored procedures, also adding in parameters and basic **`if`** statements.

How to submit

You complete **Problems 1 and 2** on the course Canvas site (short-answer questions on PL/SQL), so that you can see if you are on the right track.

Each time you wish to submit files for **Problems 3 onward**, 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.

Problem 1 - on Canvas - 8 points

Problem 1 is correctly answering the "HW 4 - Problem 1 - PL/SQL basics" on the course Canvas site.

Problem 2 - on Canvas - 6 points

Problem 2 is correctly answering the "HW 4 - Problem 2 - compiling and running PL/SQL" on the course Canvas site.

Requirements/Set-up for HTML Problems 3 - 5

- For this homework's problems, the **only** CSS permitted is the external CSS `normalize.css` included in `html-template.html`.
- For an **`img`** element, note that it needs to validate as strict-style HTML.
  - If its URL does **not** validate as strict-style HTML when used as your **`img`** element's **`src`** attribute,

    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, and use that URL instead.
- You are expected to put your HTML documents for Homework 4 into one or more **sub-directories**

with permissions of **711** in your `public_html` directory on nrs-projects.

In this case, I am asking **you** to choose the name for these sub-directories.

```
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 "deeper" sub-directories 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 3 - more practice with HTML form widget elements

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 **`prob3`** somewhere in its file name, and give its file name the suffix **`.html`** .
  - For example:

    `cp ~st10/html-template.html` *your*-**`prob3`**-*file-name*.**`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.)
- In its **`head`** element, give the **`title`** element appropriate descriptive content.

Within its **`body`** element:

- 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 **`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 top-level `fieldset` element should be an **`input`** element with **`type="submit"`** (which does *not* need a logically-related `label` element).
- Within its **`footer`** element, add a **`p`** element whose content includes **your name**.

Reminder: for this homework, you may not use any CSS to format or layout 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, **`div`** 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 files *your*-**`prob3`**-*file-name*`.html` and *your*-**`prob3`**-*file-name*`.xhtml`.

(**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` and `.xhtml` files **throughout** the week.)

Problem 4 - a form for selecting an ISBN

Consider the the small bookstore database created and initially-populated by **`create-bks.sql`** and **`pop-bks.sql`**, which you described in relation-structure form in **`describe-bks.txt`** as part of Homework 1, and which you decided on a theme for and described in Homework 2's **`about-bks.html`**.

In this problem, you are going to create a first version of a form for an eventual small application built atop your version of the bookstore database. For this small application, a form allowing the user to

select a title's ISBN will be useful.

(Since ISBNs are not very human-friendly, the user will be **shown** ISBN-title name pairs to choose from, but their selection will cause a name=value pair whose **value** is *just* the ISBN of their choice.)

Starting from the CS 328 course **`html-template.html`**, create a strict-style HTML document **`bks-isbn-choice.html`** that meets the class style standards as well as the following requirements:

- 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!)
- In its **`head`** element, give the **`title`** element appropriate descriptive content.

Within its **`body`** element:

- Include an appropriate **`h1`** element that includes the **name** for your bookstore from your **`about-bks.html`**.
- Include an appropriate block-level element that includes an **`a`** element that references your **`about-bks.html`** from Homework 2. (I think there are several reasonable choices for the containing block-level element!)
  - It is **your choice** whether you reference Homework 2's copy of **`about-bks.html`**, or create a copy of **`about-bks.html`** in your Homework 4 directory and reference that copy.
- Include a **`form`** element that meets the CS 328 class style standards and the following requirements:
  - For Homework 4, this should have an **`action`** attribute whose value is any "real" URL of your choice (because we haven't gotten to writing an actual application program to handle this form yet).
    - On a *future* homework, we'll replace this with a URL that will actually attempt to process this form.
  - It should have a **`method`** attribute whose value is **`"post"`** (although while you are debugging you can use `"get"`, as long as you replace it with `"post"` for the version that you submit).
  - It should contain a top-level **`fieldset`** element.
  - Within the top-level **`fieldset`** element, there is a **`select`** element, set up so the user can only select one ISBN-title name at a time, with a logically-related **`label`** element
    - Later, you will *dynamically*-generate the **`option`** elements for the ISBNs. For **now**, just **hard-code three or four** ISBN-title name pairs of your choice.
    - Set up this **`select`**'s **`option`** elements so that the user **sees**, in the **`select`**/drop-down box, ISBN-title name pairs, **but**...
    - ...when a particular option is selected, the **value** in the resulting name=value pair for this option is **JUST** the ISBN for the user's choice.
    - (for example, if the user selects an option that is displayed as:

`9780131103627 - The C Programming Language`

... the form will submit a name=value pair whose value is just `9780131103627`)

- **ASK ME** if you are not sure what I am asking for here.

- Also within the top-level **`fieldset`** element, there is an **`input`** element with **`type="submit"`** (which does *not* need a logically-related `label` element)

- Within its **`footer`** element, add a **`p`** element whose content includes **your name**.

Reminder: for this homework, you may not use any CSS to format or layout 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, **`div`** elements, and instances of the void element **`br`** and still have it successfully validate as strict-style HTML.

Make sure an **`.xhtml`** copy of your document validates as strict-style HTML, and submit your resulting files **`bks-isbn-choice.html`** and **`bks-isbn-choice.xhtml`.**

Problem 5 - a form for entering something "second"-db related

Consider your "second" database, the one you specified as part of **Homework 3 - Problem 1**.

Within your "second" database's scenario, imagine a question that an expected user might want to ask, that could be answered by a SQL `select` statement that has a `where` clause specifying something entered by the user.

(For the bookstore scenario, for example, a user might like to ask how many copies of a specified title are currently available.)

Starting from the CS 328 course **`html-template.html`**, create a strict-style HTML document **`custom-choice.html`** that meets the class style standards as well as the following requirements:

- 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!)
- In its **`head`** element, give the **`title`** element appropriate descriptive content.

Within its **`body`** element:

- Include an appropriate **`h1`** element.
- Include a **`form`** element that meets the CS 328 class style standards and the following requirements:
  - For Homework 4, this should have an **`action`** attribute whose value is any "real" URL of your choice (because we haven't gotten to writing an actual application program to handle this form yet).
    - On a *future* homework, we'll replace this with a URL that will actually attempt to process this form.
  - It should have a **`method`** attribute whose value is **`"post"`** (although while you are debugging you can use `"get"`, as long as you replace it with `"post"` for the version that you submit).

  - It should contain a top-level **`fieldset`** element.
  - Within the top-level **`fieldset`** element, there is an appropriate form widget element/set of elements so that the user can make the desired choice.
    - Later, you will *dynamically*-generate the different choices based on the current contents of your "second" database. For **now**, just **hard-code three or four** appropriate choices.
    - Give this/these appropriate logically-related **`label`** element(s).
  - Also within the top-level **`fieldset`** element, there is an **`input`** element with **`type="submit"`** (which does *not* need a logically-related `label` element)
- Within its **`footer`** element, add a **`p`** element whose content includes **your name**.

Reminder: for this homework, you may not use any CSS to format or layout 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, **`div`** elements, and instances of the void element **`br`** and still have it successfully validate as strict-style HTML.

Make sure an **`.xhtml`** copy of your document validates as strict-style HTML, and submit your resulting files **`custom-choice.html`** and **`custom-choice.xhtml.`**

Requirements/Set-up for PL/SQL Problems 6 - 8

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 6-8`**, and the last-modified date.
- The command: **`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 6 - PL/SQL stored procedure num_pub_titles

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 different titles the bookstore carries from that publisher
- returns nothing (since it is a procedure!)

(Note: by "number of different titles", I mean the number of distinct 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 **328lect04-2.sql** at the version of the stored procedure **hello_world** that we created during class -- 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 **describing** 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.)
  - ...followed by a SQL*Plus command calling **num_pub_titles** for that test.

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 of **`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 7 - PL/SQL stored procedure `print_test`

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 **two** arguments: 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.

- For full credit, appropriately use an **`if`** statement in this 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 8 - PL/SQL stored procedure `title_info`

Consider **Homework 3 - Problem 5, parts a and b**.

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.

- For full credit, appropriately use an **`if`** statement in this 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 **describing** 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.)

- ...followed by a SQL*Plus command calling **`title_info`** for that test.

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