CS 328 - Week 4 Lab Exercise - 2025-02-13/14

Deadline

Due by the end of lab.

Purpose

To practice with some more HTML form widgets, and to practice writing and running a PL/SQL subroutine.

How to submit

Submit your files for this lab using **`~st10/328submit`** on nrs-projects, each time entering a lab number of **84**.

- **Important note:** It is quite likely that your SQL files will be in a different directory than your HTML files, especially for the navigator. That's fine, and preferable!
  - Just remember that you need to run **`~st10/328submit`** from **EACH** directory with files to be submitted for this lab exercise.

Requirements

- You are required to work in **pairs** for this lab exercise.
  - This means **two** people working at **ONE** computer, one typing ("driving"), one saying what to type ("navigating"),

    while **BOTH** are looking at the **shared** computer screen and **discussing** concepts/issues along the way.
- Make sure **BOTH** of your names appear in each file submitted.
- When you are done, before you leave lab, **BOTH** of you should submit appropriate versions of these files using **`~st10/328submit`** on nrs-projects, with a lab number of **84**.
- For this lab exercise's problems, the **only** CSS permitted is the external CSS `normalize.css` included in `html-template.html`.

Set-up for HTML Problems 1-4

Consider before you start:

You can approach today's lab exercise Problems 1-4 in a number of different ways.

You are going to be building a form, trying out several more kinds of form widgets, and seeing what name=value pairs are submitted by those form widgets when a form containing them is submitted.

More-Scattershot approach:

You can just have different "themes" for each example form widget you are practicing with in this problem.

More-Thematic approach:

You can decide on an overall "theme" for your form at the beginning, and add form widgets appropriate for that "theme" for each example form widget you are practicing with in this lab.

...and there are certainly possibilities between these two extremes, also.

If you are interested in being more thematic, note that the elements you will be practicing with are good for the following situations:

- **asking for EXACTLY one choice from a relatively-small number of exact choices** (do you want your sandwich on white, wheat, or gluten-free bread?)
- **asking for EXACTLY one choice from a possibly-larger-number of exact choices** (choose from exactly one of 7-10 sides to be included with your sandwich)
- **asking for possibly-multiple lines of text** (enter your opinion of your sandwich order)

You might want to consider (briefly) before starting your form, then: what is a setting where you might want to ask users for information such as the above?

Problem 1 - start a form

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

- Include **`lab4`** somewhere in its file name, and give its file name the suffix **`.html`** .
  - Reminder: assuming the navigator is already in their desired **`public_html`** subdirectory for today's lab, this works for starting off this file:

    `cp ~st10/html-template.html` *your*-**`lab4`**-*file-name*.**`html`**
- Fill in the opening comment block as specified, putting in your **names**, 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!)
- Give the **`head`** element's **`title`** element appropriate descriptive content.
- Within the **`body`** element, add an appropriate **`h1`** element indicating this document is for the **CS 328 Week 4 Lab Exercise**.
- Within the **`footer`** element, add a **`p`** element whose content includes **your last names**.

Add the following after the **`h1`** element you added above:

- START a **`form`** element.
  - set its **`action`** attribute to a "real" URL of your choice (because we haven't gotten to writing an actual application program to handle this form yet)
  - set its **`method`** attribute to **`"get"`**
- In the content of this **`form`** element, add:
  - a top-level **`fieldset`** element that, to start, contains:
    - an appropriate **`legend`** element of your choice
    - a submit button (an **`input`** element with a **`type`** attribute whose value is **`"submit"`**)
- Remember that, for this lab exercise, you may **not** use CSS to format or layout this form, although you **can** tastefully use **`br`** void elements, **`p`** elements, or **`div`** elements if you wish.

validate your document-so-far

Before you go on...

- Create an **`.xhtml`** copy of your document-so-far:

  `cp` *your*-**`lab4`**-*file-name*`.html` *your*-**`lab4`**-*file-name*`.xhtml`

- Go to either https://validator.w3.org/nu or https://html5.validator.nu

  and paste in the absolute URL of the *your*-**`lab4`**-*file-name*`.xhtml` version of your file

  and click the "**Validate**" button.

  - Correct any issues that are shown before going on to Problem 2.

Problem 2 - add a `fieldset` with logically-grouped radio buttons

Reminder - Radio buttons

- **Radio buttons** are another type of **`input`** element;
- This can be a good choice when the user is to select **EXACTLY ONE** of a "SMALLISH" number of specific options.
  - Selecting one of these causes the previous selection to be UN-selected.
- IMPORTANT TO KNOW:
  - An **`input`** element with **`type="radio"`** is a radio button.
  - Browsers typically format a radio button as a small circle -- clicking it fills in that circle and "un-fills" the circle for the previously-selected radio button in its logical group.
  - Each should have a **`name`** attribute. To logically group those radio buttons representing the options for a single choice, give each the SAME value for its **`name`** attribute.
    - [One way to think about this: all together, the radio buttons with the same value for their **`name`** attribute represent a "single" logical element.]
  - Each should have a **`value`** attribute.
    - When the form is submitted, one name=value pair will be sent for the whole logical group of radio buttons;
    - the name will be the value of that logical group's **`name`** attribute,
    - and the value will be the **`value`** attribute's value for the currently-selected radio button in that logical group of radio buttons.
  - Add attribute **`checked="checked"`** to the SINGLE radio button you'd like to be INITIALLY selected when the form is created.
    - What if you don't? Then, when the form is submitted, if the user never clicked one of the radio buttons, NO name=value pair will be submitted for that logical set of radio buttons, which may be confusing on the application tier when handling this form.
  - Add a logically-related **`label`** element along with each radio button instance to logically associate that radio button's label text with that radio button.

Problem 2 requirements:

Within your **`form`** element, *within* its top-level **`fieldset`**, *before* its submit button:

- Add *another* **`fieldset`** element containing **at least three logically-grouped radio buttons**, **exactly one** of which is initially shown as checked.
  - Be sure to include an appropriate logically-related **`label`** element for *each* radio button.
- Try selecting each radio button, and:
  - Notice what happens when you click on the label for a radio button.
  - Tip: if you can select more than one of your radio buttons at the same time, you have an error!
  - Try submitting your form, and look over the name=value pair that appears at the end of your form's action URL as a result.
  - **Think about**: can you explain why that is the name=value pair submitted?

Validate an `.xhtml` copy of your document-so-far, correcting any issues that are shown, before you go on to Problem 3.

Problem 3 - add a `select` element

Drop-down box - `select` element

- This one actually IS a separate element, and **NOT** a type of **`input`** element!
- This can work better than radio buttons when user is selecting **EXACTLY ONE** from a ***larger*** set of specific options.

  (although you can add an attribute to allow the user to select more than one from the set of options)
  - By default, it is select-just-one, and selecting another unselects the previous selection
  - Can use attribute `multiple="multiple"` to allow user to allow multiple values at a time
- BASIC syntax:

```
<select name="desired-name">
    <option value="whatever1"> shows in drop-down </option>
    <option value="whatever2"> shows in drop-down </option>
    <option value="whatever3" selected="selected"> shows initially
        </option>
</select>
```

- IMPORTANT TO KNOW:
  - **`select`** is an inline element, and it CAN have content
  - Use attribute **`selected="selected"`** for the SINGLE **`option`** element whose content you'd like to be INITIALLY selected/showing when the form is created.
  - Be sure to give the **`select`** element a **`name`** attribute,

    AND to give each **`option`** element nested within it a **`value`** attribute.
  - Note that the **`option`** element's content is what the user will **see**,

    and the value of the **`option`** element's **`value`** attribute is what will be sent as the value for this element's name=value pair if that option is selected when the form is submitted.

    (and the name in the name=value pair will be the **`name`** attribute's value in the **`select`** element's

start tag)

- Add a logically-related **`label`** element along with the **`select`** element to logically associate that select/drop-down element's label text with that select/drop-down element.

Problem 3 requirements:

Within your **`form`** element, within its ***top-level*** **`fieldset`**, ***after*** Problem 2's **`fieldset`** but ***before*** the form's submit button:

- **ADD** a **`select`** element containing **at least five** options, **explicitly** specifying which is initially shown.
  - Be sure to include an appropriate logically-related **`label`** element for your **`select`** element.
  - In this case, write it so that the user can only submit a **single** value at a time (do *not* include the attribute `multiple`).
- Try selecting an option from your **`select`** element, and:
  - Notice what happens when you click on the label for the `select` element.
  - Look over the name=value pairs that appears at the end of your form's action URL as a result when you submit your form.
  - **Think about**: can you explain why that is the name=value pair submitted for your **`select`** element?

Validate an `.xhtml` copy of your document-so-far, correcting any issues that are shown, before you go on to Problem 4.

Problem 4 - add a `textarea` element

`textarea` element

- This one also is a separate element, and **NOT** a type of **`input`** element!
- This can be a good choice when the user may be entering **multiple** lines of freeform content/text.
- IMPORTANT TO KNOW:
  - **`textarea`** is an inline element, and it CAN have content
  - Be sure to give the **`textarea`** element a **`name`** attribute.

    The value of its **`name`** attribute will be the name for the name=value pair sent for this **`textarea`** when its form is submitted.
  - Its optional **`rows`** attribute specifies how many rows are displayed (height!).
  - Its optional **`cols`** attribute specifies roughly how many "characters" are displayed (width!), but note that this is approximate, especially if the display font is not monospaced (a W is wider than an i !)
  - If this element happens to include content, that content will be the *initial* text showing within the **`textarea`** element when the form is displayed.
    - Fun fact: if there are blanks or newline in this content, it will be included within the textarea when the form is displayed!
    - Using a **`placeholder`** attribute along with empty content can be convenient, but note that:

      1. This may not be read by screen reader software, so do not use this in place of a logically-associated label.

2. This **`placeholder`** attribute's value will **not** be sent as the value for the textarea if the user has not entered anything in this textarea when the form is submitted.

3. If the **`textarea`** element has ANY content -- even a single blank! -- it overrides the **`placeholder`** attribute and you won't see the placeholder attribute's value in your textarea. So, when you have a **`placeholder`** attribute, put the **`textarea`**'s end tag RIGHT after its start tag.

- Whatever (non-`placeholder`) content is appearing within the **`textarea`** when the form is submitted will become the value for the name=value pair sent for this textarea.
  - And, if there are blanks or newline in this content, it will be converted to URL-friendly character equivalents and included in the value submitted!
- Add a logically-related **`label`** element along with the **`textarea`** element to logically associate that **`textarea`** element's label text with that **`textarea`**.

Problem 4 requirements:

Within your **`form`** element, *within* its ***top-level*** **`fieldset`**, ***after*** Problem 3's **`select`** element but ***before*** the form's submit button:

- Add a **`textarea`** element, with **at least three rows** displayed.
  - Include a **`placeholder`** attribute with an appropriate value.
  - Be sure to include an appropriate **`label`** element for your **`textarea`** element.
- Try entering more than one line of text in your **`textarea`** and then submit.
  - Notice the name=value submitted for this: you should see special characters in the value part representing the newline character and any spaces you entered.
  - **Think about**: can you explain why that is the name=value pair submitted?

Validate an `.xhtml` copy of your document-so-far, correcting any issues that are shown, before you go on to Problem 5.

Lab set-up for PL/SQL Problems 5-6

- On nrs-projects, if the driver has not previously executed **`set-up-ex-tbls.sql`** in their Oracle account, they should do so, so that they have the tables `empl`, `dept`, and `customer` in their database.
  - If needed, they can get a copy of this script using:

    ```
    cp ~st10/set-up-ex-tbls.sql .   # don't forget the blank and period!
    ```

- In a SQL script **`lab4.sql`**:
  - In opening comment(s), FIRST put the script file's name, both of your names, and today's date/last modified date.
  - Put in the SQL*Plus command:

    ```
    set serveroutput on
    ```

    ...so that you will see output from **`dbms_output.put_line`** statements you put into today's PL/SQL subroutine.
  - Start spooling to a file **`lab4-out.txt`**:

    ```
    spool lab4-out.txt
    ```

...(and make sure you **`spool off`** at the script's end!)

- Put both of your names in a **`prompt`** command.

Problem 5 - write a PL/SQL stored procedure `empl_overview`

Consider this version of PL/SQL stored procedure **`hello_world`** (slightly added to after class, including what will be the CS-328-required opening comment block):

```
/*===
    procedure: hello_world: void -> void
    purpose: expects nothing, returns nothing, and has the
        side-effect of (IF serveroutput is on) printing to
        the screen a cheery greeting, the current date,
        and the current value of local variable quantity
===*/

create or replace procedure hello_world as
    curr_date date;
    quantity integer := 1;
begin
    select sysdate
    into curr_date
    from dual;

    quantity := quantity + 1;

    dbms_output.put_line('Hello, world! on ' || curr_date);
    dbms_output.put_line('quantity is: ' || quantity);
end;
/
show errors
```

(You can see an overly-comment version of this that includes testing calls posted with the Week 4 Lecture 2 class examples.)

To get some practice writing a PL/SQL stored procedure, write a stored procedure **`empl_overview`** that meets the following requirements:

- It expects nothing (that is, it expects no arguments).
- It returns nothing, but has the side-effect (IF **`serveroutput`** is set to **`on`**) of printing to the screen the following (in reasonable messages):
  - the current **`sysdate`**
  - the current number of employees
  - the average employee salary for the current employees
- Create an opening comment block for your procedure that has a **`procedure:`** part and **`purpose:`** part using the same style that you see for procedure **`hello_world`** above.
- 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 **`empl_overview`**.
  - test it? Yes; and since it is a procedure, we'll kluge tests by printing to the screen what we SHOULD see if the procedure is working, followed by the desired testing call.
  - For example, here are the tests I added for hello_world after class:

```
prompt ============
prompt TESTING hello_world, should see Hello, world! on the current date
prompt (using: exec hello_world() )
prompt

exec hello_world()

prompt ============
prompt TESTING hello_world, should see Hello, world! on the current date
prompt (using: exec hello_world )
prompt

exec hello_world
```

- SO: in your **`lab4.sql`**, follow your **`show errors`** command with a blank line and then **`prompt`** command(s) stating that you are about to test **`empl_overview`** 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 give a blank line, followed by a SQL*Plus command calling your procedure.
- Then, after a blank line, include this **`delete`** command:

```
delete from empl
where job_title = 'Clerk';
```

  - Follow this with a blank line and then **`prompt`** command(s) stating that you are about to test **`empl_overview`** on the case in which all the clerks were removed, and describe what you should see if it is working properly.
  - Then give a blank line, followed by a SQL*Plus command calling your procedure again.
  - Then give a blank line, followed by calling the command:

    `rollback;`

    ...to undo the deletion of the clerks!
- Finally, end your script by turning off spooling:

```
spool off
```

If successful, your resulting **`lab4-out.txt`** should show that your procedure successfully compiled, and that its tests passed.

Problem 6 - demo that `empl_overview` IS now stored in the database

If, in Problem 5, your procedure **`empl_overview`** compiled without errors, this PL/SQL stored procedure is now **stored in your Oracle database** along with your tables, views, etc.

You do **not** have to recompile or recreate it next time you log into **`sqlplus`** -- it will persist!

To demonstrate this:

- Exit **sqlplus**, and then restart it.
- Type these commands in **sqlplus**:

```
set serveroutput on
spool prob6-demo.txt
prompt include both of your names here
exec empl_overview()
spool off
```

- Exit **sqlplus**.

Your resulting file **prob6-demo.txt** should contain both of your names, followed by the result of successfully running your stored procedure **empl_overview**.

BEFORE you leave lab:

Make sure that you **both** have copies of the files:

- *your*-**lab4**-*file-name*.**html**

  your-**lab4**-*file-name*.**xhtml**

  - that include a correct URL in a comment for successfully executing this document from one of your nrs-projects account(s)
  - (that is, I will again leave it up to the navigator to decide if they would like to UPDATE their *your*-**lab4**-*file-name*.**html** and your-**lab4**-*file-name*.**xhtml** so its opening comment includes the URL to *their* copy, or if they want to leave the URL for the driver's copy)

- **lab4.sql** and **lab4-out.txt**
- **prob6-demo.txt**

...and you BOTH submit these **five** files using **~st10/328submit** on nrs-projects, with a lab number of **84**.

How the navigator can get Problem 5-6's files:

(for a driver with username *dr12*, and a navigator with username *na89* - replace these with your *actual* usernames when you actually do this)

These may be in a directory that is harder for the navigator to make a copy from.

For example -- they might be in a directory `328lab04` that is **not** a sub-directory of `public_html`., but is instead a subdirectory of *dr12*'s home directory.

Here is an approach for this:

- The **driver** should ***temporarily*** make the directory with these files world-executable, and these files world-readable -- assuming the driver is currently in their directory `328lab04`:

```
chmod 711 .   # notice the space and the period!
chmod 644 lab4.sql lab4-out.txt prob6-demo.txt
```

- Now the **navigator** can copy these into a directory of their choice -- assuming they are within the directory they want to copy into:

  cp ~*dr12*/328lab04/lab4.sql . `# notice the space and the period!`

  cp ~*dr12*/328lab04/lab4-out.txt .

  cp ~*dr12*/328lab04/prob6-demo.txt .

- After the copies are successfully made, the **driver** and **navigator** should **BOTH protect** these files:

  chmod 600 lab4.sql lab4-out.txt prob6-demo.txt

- ...and **both** can now submit these using **~st10/328submit** from their directory containing these files.

Reminder: how the navigator can get a copy of the .html file

(for a driver with username *dr12*, and a navigator with username *na89* - replace these with your *actual* usernames when you actually do this)

- Because this is a file the nrs-projects web server has to be able to reach, the navigator should be able to get a copy of this file from the driver using an approach like this:
  - Assume the driver has, in their `public_html` directory, a sub-directory `328lab04` containing *your*-**lab4**-*file-name*.**html**. (Adapt the following accordingly based on your driver's actual *your*-**lab4**-*file-name*.**html** location.)
  - The NAVIGATOR *na89* can now:
    - log in to THEIR (*na89*'s) nrs-projects account, and run these commands:

      cd public_html

      mkdir 328lab04 `# or` **other** `name they choose`

      chmod 711 328lab04

      cd 328lab04

      cp ~*dr12*/public_html/328lab04/*your*-lab4-*file-name*.html . # note space & period!
  - Now the navigator *na89* should have their own copy of *your*-**lab4**-*file-name*.**html**
- Either repeat for *your*-**lab4**-*file-name*.**xhtml**,

  or (if desired) change the URL within the navigator's *your*-**lab4**-*file-name*.**html** to refer to the navigator's copy, and then make the **.xhtml** copy (and double-check that it still validates):

  cp *your*-**lab4**-*file-name*.**html** *your*-**lab4**-*file-name*.**xhtml**