CS 328 - Homework 3

Deadline

11:59 pm on **Friday, February 14, 2025**

Purpose

To let me know your "second" database you will be building several application upon, to practice more with writing and (at-least-partially) validating more strict-style HTML, including more practice with **`table`** and **`form`** elements, and to write some SQL statements both for review and to help get more familiar with the bookstore tables before using them with PL/SQL on future course assignments.

How to submit

Problem 1's files should be submitted with the special homework number of **33**, as described below.

Then, each time you wish to submit your work for **Problems 2 onward**, submit those files using `~st10/328submit` on nrs-projects, with a homework number of **3**.

Important note: It is quite likely that your 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 3.

Homework 3 Requirements/Set-up

- For this homework's problems, the **only** CSS permitted is **EITHER**:
  - **Just** the external CSS `normalize.css` included in `html-template.html`, **OR**
  - The external CSS `normalize.css` included in `html-template.html`, **after** which you **also** put:

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

    (and this would be right BEFORE the **`head`** elements closing tag)

    ...which adds a *little* mostly-table-related formatting.
- 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 3 into a **sub-directory** with

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

In this case, I am asking **you** to 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 3 files in a "deeper" sub-directory within `public_html`.)

Problem 1 - let me know your "second" database

If you recall, Homework 1 - Problem 2 noted that you will be building application pieces atop **another**, **second** database in addition to the bookstore database, and you started considering what that database might be as part of Homework 1 - Problem 2's submission `second-db.txt`.

Homework 2's handout, in the section 'NEXT STEPS for your "second" database', provided more information about this, including letting you know about some posted available shred databases and giving you a heads-up about the files you would need to provide for your chosen second database.

So, as noted in that Homework 2 handout section, now it is time for you to let me know your "second" database.

Remember: several databases that several CS 325/CS 328 students have given me permission to share have been posted -- you can see them at the course Canvas site, under "Modules", scrolling down to the section "**Available Shared Databases**".

- If you do decide to use one of these, you will be required to leave the original creator's name(s) in those files -- you may modify your copies of them, adding **additional** comments of "modified by:" with your name and the date last modified if you do so.
- You will also be required to include a "database designed by" credit including their name(s) at the beginning of each application you build atop this database.
- (Understand that these have been shared **as-is**, so you might find errors that need to be fixed, or aspects you would like to change, etc.! Each time you do change them, you will be expected to submit the revised database file(s) along with the first homework that uses/depends on those revisions.)

For this homework problem, (**for Homework 3 - Problem 1**), you need to submit the following **FIVE** files for your selected "second" database:

- **NOTE**: I am considering writing scripts that search the homework submission for files with names that end *precisely* as noted below, so if yours do not, you may lose some credit.
- Your selected database' **business rules** file, **in PDF format**, in a file whose name ends in **`biz-rules.pdf`**

- Your selected database' **ER model** file, **in PDF format**, in a file whose name ends in **`model.pdf`**
  - If you are creating a new database: note that this model should have at least five distinct entity classes, with at least four relationships, at least one of which is 1:N.
  - (Where does the "at least 7 relations" part come in, then? That is based on assuming some combination of N:M relationships, multi-valued attributes, and supertype/subtype entity classes in your model will result in at least two more relations when this model is converted to a design.)
- Your selected database' **design given in relation-structure form** (the form you used to express the bookstore database in Homework 1 - Problem 3's file `design-bks.txt`), **in plain-text format**, in a file whose name ends in **`design-rs.txt`**
- The **SQL script** implementing your selected database' **design**, in a file whose name ends in **`design.sql`**
  - The grader or I may need to create a version of your database at some point, so you may lose credit if this SQL script does not successfully create empty versions of your selected database's tables if we try it.
- The **SQL script** for an initial **population** of your selected database' design, in a file whose name ends in **`populate.sql`**
  - The grader or I may need to initially-populate a copy of your database at some point, so you may lose credit if this SQL script does not successfully initially-populate the tables created in your design script if we try it.

These files should be submitted using the **special** homework number of of **33** so that I'll know you've chosen your "second" database and this is it.

- If you already submitted this using this special homework number as part of Homework 2, then you have already completed this problem, Problem 1, for Homework 3!
- By the way, it is perfectly fine if you improve any of these files later -- just re-submit them using the homework number of **33** and I'll then know that's an improved updated version of the files included.

Problem 2

As you read the zyBooks Chapter 2 - "More HTML", you will see that there are numerous form widgets implemented using the HTML **`input`** element besides submit buttons and classic textfields!

Read about and consider these -- there are at least **14** more of them, in addition to **`type="submit"`** and **`type="text"`** -- demonstrated and/or described in that chapter!

Select **five** of these other **`input`** element types -- whose **`type`** attribute is **not** `"submit"` or `"text"` -- that you consider to be particularly useful or your favorite, noting that you will be **listing** them in a **`table`** element for *this* problem as described further below, and **demonstrating** them in a **`form`** element for the *next* problem!

Starting from the **`html-template.html`** posted on the course public site and along with this homework handout, (and also available on nrs-projects from `~st10/html-template.html`), create a strict-style HTML document that meets the class style standards as well as the following

requirements:

- Include **`prob2`** somewhere in its file name, and give its file name the suffix **`.html`** .
  - For example:

    `cp ~st10/html-template.html` *your*-**`prob2`**-*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!)
- Give the **`title`** element within the **`head`** element appropriate descriptive content.

Within its **`body`** element:

- Include an appropriate **`h1`** element indicating that what follows are your choices for 5 other particularly-useful or 5 other favorite form widgets implemented using an **`input`** element (*besides* the classic submit button and textfield).
  - (This can be as simple as "5 more input elements" or "5 other useful `input` widgets" or "My five favorite other `input` types", etc.)
- Include a **`table`** element with six rows and two columns that meets the CS 328 class style standards, that also meets the following requirements:
  - It should include an appropriate **`caption`** element.
  - The first row should contain column headers **Type** and **Use for**, implemented using the appropriate element with the appropriate attribute included for better accessibility.
  - The remaining five rows should each contain the value of a **`type`** attribute described and/or demonstrated in zyBooks Chapter 2,

    and then a short description of what that type of input should be used for.
    - (For example, if you were allowed to include a row for a submit button, its type would be submit, and its use could be for "submitting a form's data".)
- Within its **`footer`** element, add a **`p`** element whose content includes **your name**.

Make sure an **`.xhtml`** copy of your document validates as strict-style HTML using https://validator.w3.org/nu or https://html5.validator.nu/, and submit your resulting files *your*-**`prob2`**-*file-name*.**`html`** and *your*-**`prob2`**-*file-name*.**`xhtml`**.

Problem 3

Consider the five other types of the `input` element that you included in **Problem 2's `table`** element.

Starting from the **`html-template.html`** posted on the course public site and along with this homework handout, (and also available on nrs-projects from `~st10/html-template.html`), 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!)
- Give the **`title`** element within the **`head`** element appropriate descriptive content.

Within its **`body`** element:

- Include an appropriate **`h1`** element noting that this is a form demonstrating several of the available types of `input` elements.
- 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"/working URL of your choice (because we haven't gotten to writing an actual application program to handle this form yet).
    - If you select a shorter one, you'll find it easier to peruse the name=value pairs at its end while trying out your form in a browser.
  - It should have a **`method`** attribute whose value is **`"get"`**.
  - It should contain at least one **`fieldset`** element that contains an appropriate **`legend`** element of your choice, and within this/these should be:
    - at least one instance of each of the five of the **`input`** elements you included in **Problem 1's `table`** element
      - (Depending on your choices, you might find it makes sense to include more than one instance of some of these five. That's fine!)
    - appropriate logically-related **`label`** elements for each of those **`input`** elements
    - and also one **`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 submitting 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 using https://validator.w3.org/nu or https://html5.validator.nu/, and submit your resulting files *your*-**`prob3`**-*file-name*.**`html`** and *your*-**`prob3`**-*file-name*.**`xhtml`**.

Problem 4

You are going to create a SQL script named **`new-titles.sql`** for this problem. Give this file permissions of **`600`** by typing this at the nrs-projects prompt:

`chmod 600 new-titles.sql`

As you hopefully noticed in making the relation-structure version of its tables in your Homework 1 file **`design-bks.txt`**, the **`title`** relation represents the bookstore's inventory, essentially -- each of its elements represents how many copies are available to be sold of a particular book title. This **`title`** relation is related to the **`publisher`** relation, representing a publisher of books.

This **`title`** relation also happens to include the **cost** *to the bookstore* of that particular book title, the **price** the bookstore *sells* it for, at what quantity they'd like to consider re-ordering more copies, and the default number they re-order at that point.

So - to review writing SQL **`insert`** statements, since they may be included within PL/SQL subroutines later this semester, consider the theme for your bookstore, as you described in your Homework 2 file **`about-bks.html`**.

Then, write a small SQL script **`new-titles.sql`** that meets these requirements:

- Start it with comment(s) containing **`CS 328 - HW 3 - Problem 4`**, your name, and the last-modified date.
- Start spooling to a file **`new-titles-out.txt`** (and make sure you **`spool off`** at the script's end!)
- Write a **`prompt`** command including your name.
- Include SQL **`insert`** statements to insert at **least two new rows** into the **`title`** table for two book titles, ***real or imaginary, your choice,*** that **fit in** with the **theme** of your bookstore.
  - You get to make up reasonable-to-you values for the attributes of these new **`title`** rows.
  - However, give all of their attributes values (that is, for full credit, they must **all** be **non-null**).
- You decide on the publisher for each of these new titles, and make sure you use that publisher's **`pub_id`** for that title's **`pub_id`** foreign key.
  - It is fine to insert more publisher rows if you would like, but also fine to use one of the existing publishers for each of your new titles.
  - (If you decide to add any publisher rows, be sure to put their **`insert`** statements **BEFORE** the **`insert`** statements for titles published by those publishers!)
- After your **`insert`** statements, write a SQL `commit;` statement to commit your changes.
- Remember to **`spool off`** at the end of your script.

IMPORTANT NOTE: if you need to debug your script -- or even just re-run it -- be sure to **re-run** **`pop-bks.sql`** ***before*** doing so, otherwise you might get errors due to trying to insert your new rows more than once! Remember that you can make a copy of this script in the same directory as your **`new-titles.sql`** script by using the command:

```
cp ~st10/pop-bks.sql  .    # note that space and period at the end!
```

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

Problem 5

You are going to create a SQL script named **`prob5.sql`** for this problem. Give this file permissions of **`600`** by typing this at the nrs-projects prompt:

```
chmod 600 prob5.sql
```

To continue getting more familiar with the bookstore tables, as a little more SQL warm-up and review before we start out coverage of PL/SQL, and for possible use in future PL/SQL subroutines, write a script **`prob5.sql`** that meets the following specifications:

- Start it with comment(s) containing **`CS 328 - HW 3 - Problem 5`**, your name, and the last-modified date.
- Start spooling to a file **`prob5-out.txt`** (and make sure you **`spool off`** at the script's end!)
- Write a **`prompt`** command including your name.
- For each of the following parts, write a **`prompt`** command giving the problem part being answered, then your answer for that part.

Since this problem is intended to help you continue to both get familiar with the bookstore database and review writing SQL queries, provided along with this homework handout is an example **`prob5-out.txt`** so you can tell if your queries are on the right track. (Note that the new titles added in Problem 4 will be different between your version and mine!)

Problem 5 - part a

For future-cheezy-PL/SQL-function use...

How could you find out the *current* **largest** value for the primary key of table `order_needed`?

Write a SQL query that projects the current largest value of table `order_needed`'s primary key attribute, giving the resulting column the name **`"Largest PK"`**.

Problem 5 - part b

How might you write a SQL query whose result would let you know if there is a book title in the **`title`** table with a particular ISBN, in such a way that it WON'T cause an error if there is *no* such title?

Several ways are possible, but here is just one of those: you could write a query that projects the **number of rows** that have that ISBN, giving the resulting column the name **`"Quantity"`**. (If no such row exists, this projects 0, and does not result in an error message.)

So, write TWO example versions of this query:

- Write a SQL query that projects the number of rows in the **`title`** table with ISBN of

`'9780131103627'`.

- Write a SQL query that projects the number of rows in the **`title`** table with ISBN of `'5555555555555'` (which is a non-existent ISBN!).

Problem 5 - part c

Consider: when selling a copy of one of the titles sold by your bookstore, you might want to know that title's current quantity on hand (so you can decrease it after this sale), its order point (so you can determine if you want to order more after this sale), and its auto order quantity (so you will know how many you usually order if it is time to reorder it).

Write a SQL query that projects these three attributes for a title whose ISBN is **`'9780131103627'`**.

Problem 5 - part d

ISBNs are lovely for unique identifiers, but not great for human readability!

Write a SQL query that projects a single column for each title in the **`title`** database: its ISBN concatenated to a space, a dash, and a space, concatenated to its title, giving the resulting column the name **`"Available Titles"`**.

Problem 5 - part e

Write a SQL query that projects, for each title in the **`title`** database, the title, and then the **name** of its publisher, displaying the rows in **order** of publisher name, and for those with the same publisher, in **secondary order** by title name.

Make sure you turned spooling off at the end of **`prob5.sql`**, and submit your resulting **`prob5.sql`** and **`prob5-out.txt`**.