CS 328 - Homework 11

Deadline

11:59 pm on **FRIDAY, May 3, 2024**

Purpose

- To think a bit more about accessibility and SQL injection,
- to practice a bit more with basic unobtrusive-style client-side JavaScript,
- to write a PL/SQL stored procedure or stored function for your selected "second" database,
- and to call it as part of a postback PHP that:
  - uses your selected "second" database,
  - is styled with an external CSS,
  - and includes some use of unobtrusive-style client-side JavaScript.

How to submit

IF you happen to update any of the files for your selected "second" database, submit updated versions of them using the special homework number of **33**.

Otherwise, each time you wish to submit any of the files for these problems, submit them using `~st10/328submit` on nrs-projects, with a homework number of **11**.

Important notes

- Be sure to follow **unobtrusive-style client-side JavaScript coding guidelines** as discussed in class.
- **DO NOT USE ANY JAVASCRIPT LIBRARIES (jQuery, Prototype, script.aculo.us, etc.) FOR THESE PROBLEMS** -- you are to use "plain" JavaScript in this course, to get experience with what underlies all of those libraries.
- **NOTE**: you are welcome to use `require_once/include_once/require/include` in your PHP documents!
  - BUT when you use them in homework problems' documents, be sure to also SUBMIT copies of all files that you are requiring/including!
- **DO NOT USE ANY PHP FRAMEWORKS FOR THESE PROBLEMS** -- one of this course's purposes is to provide you with practice with "plain" PHP.
- Remember: You are required to use **`normalize.css`** for all of your web pages for CS 328, and to add **`link`** elements for additional CSS external stylesheets *after* this (but still **within** the `head` element).

  EXCEPT for `normalize.css`, **DO NOT USE ANY CSS FRAMEWORKS or PREDEFINED LIBRARIES for this course** (unless you get prior, explicit permission). One of this course's

purposes is to provide you with some practice with the basics of "plain" CSS, so you can better make use of frameworks and predefined libraries later.

- You are expected to follow all course coding standards posted on the public course web site; course documents are also expected to validate as "strict"-style HTML, and valid CSS.

Problem 1 - more on accessibility - 10 points

We have talked in class a bit about accessibility-- to add a bit of depth to your knowledge in accessibility, consider:

- the short article **"Designing for accessibility is not that hard"** and
- the Teach Access Initiative's short tutorial on accessibility

...posted on the now-added link to **"A selection of Accessibility and Web Design references"** on the public course web site (and also posted along with this homework handout).

Then, in a **plain-text file** whose name includes **`hw11-access`** and ends with the suffix **`.txt`**:

- include your name
- give at least **three** rules-of-thumb or guidelines from the article "Designing for accessibility is not that hard" that you want to make sure to remember as you design web sites/web applications **OR** that you wish others would remember as they do so
- give at least **three** rules-of-thumb or guidelines (**different** from those you just selected from the article above) from the posted Teach Access Initiative's short tutorial on accessibility that you want to make sure to remember as you design web sites/web applications **OR** that you wish others would remember as they do so

Submit your file **`*hw11-access*.txt`**.

Problem 2 - more on SQL injection - 10 points

We have talked in class a bit about SQL injection -- to add more depth to your knowledge of this, consider the two articles:

- **"SQL Injection Attacks by Example"** and
- **"How security flaws work: SQL injection"**

...that are posted along with this homework handout on the public course web site, under "Homeworks and Handouts".

Read these articles; pay special attention to the "**Mitigations**" section near the end of article (1), and note that an example of article (2)'s prepared statements are statements using OCI's bind variables.

Then, in a **plain-text file** whose name includes **`hw11-sql-inj`** and ends with the suffix **`.txt`**:

- include your name
- list **at least THREE** important "take-aways" from these articles, and **for each**, explain **WHY** you chose it. (So, there are SIX PARTS to this -- your three selections, AND WHY you chose each.)

Submit your file **`*hw11-sql-inj*.txt`**.

Problem 3 - a little client-side JavaScript - 15 points

The purpose of this problem is to get more practice unobtrusive-style client-side JavaScript.

FUN FACTS (that might be of use in this homework):

- JavaScript function **`parseFloat`** expects a string that is reasonable to parse into a floating-point number and returns such a floating-point number if it can -- but may return the special value **`NaN`** (not a number) if given an argument such as `"Jimmy"`
  - As noted in zyBooks Section 7.2, there is also a JavaScript function **`isNaN`** -- it expects one argument and returns **`true`** if the argument is not a number, and returns **`false`** otherwise.
- There is a JavaScript function **`parseInt`** as well, that expects a string that is reasonable to parse into an integer number and returns such an integer number if it can -- but may return the special value **`NaN`** (not a number) if given an argument such as `"Jimmy"`.
  - What happens if you give **`parseInt`** a non-integer number? Looks like it returns an integer version -- and looks like it just truncates any fractional part!

3 part a

Determine at least one type of numeric computation you would like to perform. (It can be as simple as addition, or as involved as you would like.)

Then, create a **PHP or HTML** document (your choice) in a file whose name includes **`number-fun`** that meets the following requirements:

- Include your name and last modified date in its opening comment, **AND** the URL this can be run from
  - (You *will* lose some credit if this URL does not work when I or the grader paste it into a browser!)
- Include an appropriate **`title`** element.
- Include an appropriate **`h1`** element
- Include **at least two** number fields (**`input`** elements with **`type="number"`**) (so the user can enter the needed numbers for the computation you chose).
  - Fun fact: if you give a number field a **`step`** attribute whose value is not an integer, then that number field can accept non-integer numbers!
  - Reference: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/number
- at least one **`button`** element (so the user can indicate that they would like for a computation to now be done)
- (you may also include additional elements as you would like)
- how will you show the result? You get to determine this. (You could display it in a textfield, for example, or within a paragraph, or within a textarea, etc.)

3 part b

Using unobtrusive-style client-side JavaScript, write an **external JavaScript** in a file whose name

also includes **`number-fun`** to now perform the numeric computation(s) you decided upon, using **`*number-fun*.`**{**`php`** or **`html`**}'s number fields' contents when its **`button`** element is clicked, making sure to somehow show the computation's results to the user. Do what is needed for **`*number-fun*.`**{**`php`** or **`html`**} to use this external JavaScript **`*number-fun*.js`**.

Submit your resulting versions of **`*number-fun*.`**{**`php`** or **`html`**} and **`*number-fun*.js`**.

Problem 4 - one more PL/SQL stored function or procedure - 15 points

Consider your selected "second" database, the one you specified on Homework 10 - Problem 1.

As foretold in Homework 10, for this problem you are to decide on and implement either a **PL/SQL stored function** or a **PL/SQL stored procedure** using your selected "second" database.

What might be a useful PL/SQL stored function or stored procedure for this database? It does not have to be complex, but it has to be potentially useful, and you will be calling it from the application tier as part of Problem 5.

For it to be feasible to call your PL/SQL stored function or stored procedure from a postback PHP document, your PL/SQL stored function or stored procedure needs to meet these requirements:

- It needs to involve table(s) from your selected "second" database.
- It needs to expect **at least one parameter** (for more OCI bind variable practice).
- It needs to **NOT** depend on `dbms_output.put_line` calls (since their output cannot reach the application tier).
- It should be potentially useful to the audience for and/or to an application programmer developing applications atop your selected "second" database.

It does not have to be complex, as long as it might be potentially useful. For example:

- A PL/SQL stored function might conveniently compute and return a count, sum, average, maximum, or minimum computation based on a choice selected by an end-user.
- A PL/SQL stored procedure might insert, update, or delete into a table based on information entered into a form by an end-user.

Create a **SQL script** in a file whose name includes **`second-db`** that meets the following requirements:

- Include comment(s) containing at least your name, **`CS 328 - Homework 11 - Problem 4`**, and the last-modified date.
- Include a SQL*Plus **`spool`** command to spool the results of running this SQL script to a file whose name includes **`second-db-out`** and whose suffix is **`.txt`**,

  followed by a **`prompt`** command including **your name**.
- (Be sure to **`spool off`** at the end of this script.)
- Following the CS 328 style standards, design and implement a potentially-useful PL/SQL stored function or stored procedure for your selected "second" database.

- Follow that definition with **at least two testing calls** that demonstrate that it works.
  - (For a stored procedure, you also will likely need to include one or more **`select`** statements showing that the desired side-effect(s) occurred.)

Submit your files **`*second-db*.sql`** and **`*second-db-out*.txt`**.

(IMPORTANT NOTE:

- I or the grader should be able to run your Homework 10 - Problem 1's `*design.sql` and `*populate.sql`, and then your `*second-db*.sql`, successfully.
  - SO: **IF** you make any changes to your `*design.sql` or `*populate.sql`,

    then make sure you ALSO:
    - submit updated copies of your `*design.sql` or `*populate.sql` (using a homework number of **33**)
    - along with your `*second-db*.sql` and `*second-db-out*.txt` (submitted using a homework number of **11**)!

Problem 5 - one more application atop your selected "second" database - 50 points

Create a PHP **postback** document using your selected "second" database whose "main driver" PHP is in a file whose name *also* includes **`second-db`** and meets the following requirements:

- Include your name and last modified date in its opening comment, **AND** the URL this can be run from
  - (You *will* lose some credit if this URL does not work when I or the grader paste it into a browser!)
- Include an appropriate **`title`** element.
- It needs to include **at least two states**.
- At least **one** of its states must create a form that includes a form widget or set of form widgets **dynamically-built** from the results of **using OCI to query** your selected "second" database.

  For example, it could include:
  - ...a `select`/drop-down widget whose `option` elements are built based on a query's results,
  - ...a logically-connected set of radio buttons whose values are built based on a query's results,
  - ...a set of checkboxes whose names or values are build based on a query's results.
- At least **one** of its states must **respond** to a previous state's form using OCI using at least one **bind variable** to query something from your selected "second" database requested by the user.
- At least **one** of its states must used OCI to **call** your **PL/SQL stored procedure** or **PL/SQL stored function** from Problem 4.
- Its results need to be styled using an external CSS in a file whose name *also* includes **`second-db`** (set up so the results are styled first by `normalize.css` and then by this **`*second-db*.css`**).

Requirements for this **`second-db.css`**:

- It is fine if this is simply a copy of your `second-db.css` from Homework 10, as long as it also works for styling this postback PHP application.
- Include a comment including at least your name and the last-modified date.
- Include at least **five** distinct rules in this **`second-db.css`** that have a visible effect (but you can certainly add more if you are inspired!).

- At least one of its states needs to include some small-but-demonstrable use of **unobtrusive-style client-side JavaScript**, at least part of which is from an **external JavaScript** in a file whose name *also* includes **`second-db`**.
  - This could be as simple as a JavaScript pop-up based on some user action, or some simple form validation (as we will be discussing during Week 14), or could be something more involved; you get to choose.
  - But **simple-and-working** is **better** than complex-and-not-working!
  - (And remember, it needs to be **unobtrusive-style** client-side JavaScript.)

Submit your files **`*second-db*.php`**, **`*second-db*.css`**, **`*second-db*.js`** (and any files besides `hum_conn_no_login.php` or `hum_conn_sess.php` that it uses, if any).