Databases and the Web – Exercise 7

This week the lab class time will be given over to in-class test 1 but you should devote some of your self-study time to working on this exercise too. Here we'll start developing pages that pass information and act upon that information. For starters we'll simply add a new city to the world database.

Outcomes:

When you've finished this exercise you should:

- Understand how data are passed to-and-fro between the client and server.
- Be able to write PHP to create a form, process the form and update the user.
- Have written some server-side validation code and understand the use of client-side validation.

Debugging: Because PHP is *server-side* you **must** check your code in the browser through the web server "http://..." *not* by loading the page as "file://"! It is a good idea to:

i. Open the PHP page in the browser (http://localhost/~linux/… or http://studentnet/~k*xxxxxxx*/...)
ii. Make small changes to code.
iii. Save the file (upload to StudentNet if necessary).
iv. Refresh the page in the browser.
v. Fix any reported syntax errors.
vi. Test the changes.

Repeat *ad nauseum*. (Get used to using Alt-Tab to cycle through windows ☺)

Task 1: Create a page to enter information about a new city

Create something that has elements like this example page that will (by Task 3!) allow you to add a new city to the **`world`** database **`city`** table.

1. Create a PHP page that has a **`title`** and all the usual stuff.
2. Add a **`form`** that uses the POST method & whose action attribute refers back to the same page.
3. Inside the form add **`input`** tags for the necessary fields from the **`city`** table.
   - Make sure they have **`<label>`**s, **`name`** and appropriate **`maxlength`** attributes.
   - **Question**: Do you need to allow the city ID to be entered?
4. Add an appropriate tag to enable the user to select the *country* in which the city belongs.
   - **Hint**: **`while`** loop and **`select`**/**`option`** elements?
   - This is *most important bit* of the whole page – you must maintain the relationship between the **`city`** and **`country`** tables.
   - Since this is PHP and it's preparing the HTML page on the server it is fairly easy to query the **`country`** table and use it to list all possible country names and their

codes. Remember: it's the foreign key that this page needs for any INSERT operation into **`city`** but the user does not care about the *code*, just the *name*.

5. Add a submit button.
6. Now is a convenient time to test the page.
   - Make sure you view the source in your browser and then use *e.g.* "\t" and "\n" to make the fields more legible … it consumes some bandwidth but makes debugging the output from your PHP *so* much easier…
   - Write out the contents of the $_POST array, *e.g.* something like the following:
     - ```

         <pre><?php print_r($_POST); ?></pre>

       ```
   - After you submit data you should see it echoed in the **`$_POST`** array.

Task 2: Validate the data entered

Before you get around to accidentally corrupting your database with invalid data, validate the fields in PHP (*server-side*).

NB: This is the complex programming part so make sure you understood the examples from the week 5-7 slides.

It will be helpful to (eventually) use *regular expressions* for validation but for now it's sufficient to simply check the length of the entered data fields using the PHP function strlen (the logic is the same, it's just the validation test that can develop.)

You could simplify the code by storing validation results in, *e.g.* an array like **`$validField['fieldName']=TRUE;`** which you can then use to first decide if the whole form is valid and then at each **`input`** element decide whether or not an error needs to be signalled.

1. City names should be no more than 35 letters.
2. District names should be no more than 20 letters.
3. Population should be non-negative numeric values.
4. Country codes are exactly 3 capital letters.
   - **Question**: Why should you validate the *submitted* country code?
5. Data entered should be put back into the input fields if the form was incomplete.
   - Truncate strings appropriately.
   - Make any chosen country code the default (country).
6. Invalid data should be signalled in some way by writing an error message to the page and highlighting the invalid fields.
   - An ideal use for CSS classes – you could add an "error" class to the invalid input elements and use CSS to specify some special style for the dodgy data.
     E.g using **`echo`**.
     `<input type="input" name="name"`
       ```

```
<?php
    if (!$validField['name']) echo ' class="error"';
    if ($_POST['name']) echo    ' value="' .
                                $_POST['name'] . '"';
?>
/>
```

E.g. using "printf"

```
<input type="input" name="name"
<?php
    if (!$validField['name']) echo ' class="error"';
    if ($_POST['name'])
          printf(" value=\"%s\"", $_POST['name']);
?>
/>
```

Task 3: Insert into your database

This is easy *but* should **only** happen once all the fields contain valid data.

1. Build an SQL query string like
   ```
   $sql = 'INSERT INTO City '
        . '(Name,CountryCode,District,Population) VALUES ('
        . mysql_real_escape_string($_POST['city']) … etc.
   ```
2. Execute the query with `mysql_query` … *etc*.
3. Check the **INSERT** by verifying how many rows were affected with `mysql_affected_rows` … *etc*.
4. Give an error or success message, as appropriate.
5. Test it! Add something like the city of "Ankh Morpork", population 1 million souls that is situated in the "Sto Plains" district in a country of your choice ☺

Extra Task: (For next week's too) Client-side validation

Implement a simple client-side validation scheme using JavaScript.