

Assignment: Profiles, Positions and jQuery

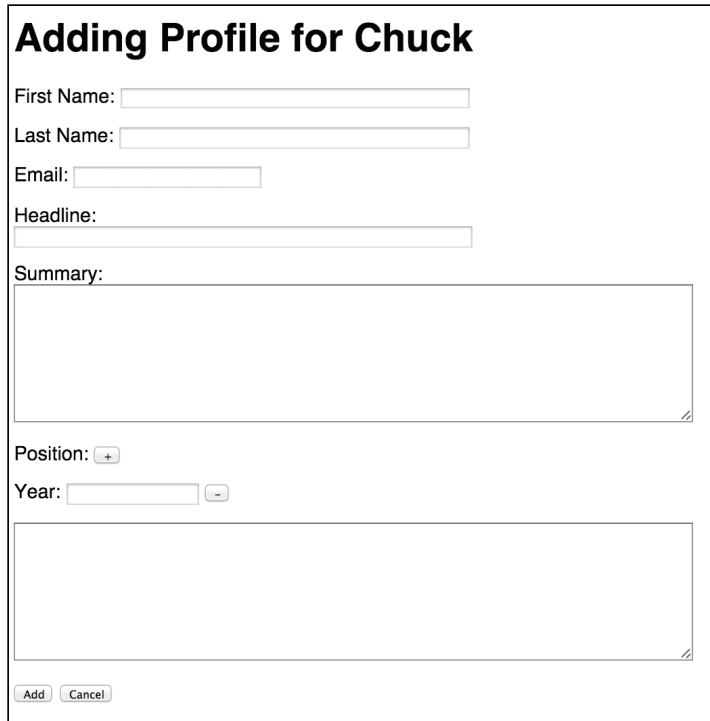
In this assignment you will extend our simple resume database to support Create, Read, Update, and Delete operations (CRUD) into a **Position** table that has a many-to-one relationship to our **Profile** table.

This assignment will use JQuery to dynamically add and delete positions in the add and edit user interface.

Sample solution

You can explore a sample solution for this problem at

<http://www.wa4e.com/solutions/res-position/>



Adding Profile for Chuck

First Name:

Last Name:

Email:

Headline:

Summary:

Position:

Year:

Resources

There are several resources you might find useful:

- You might want to refer back to the resources for the [previous assignment](#).
- An article from Stack Overflow on [Add/Remove HTML Inside a div Using JavaScript](#) (you can scroll past the JavaScript-only answers and see the jQuery answer at the bottom)
- The documentation for [PDO lastInsertId\(\)](#) where you can retrieve the most recently assigned primary key as a result of an INSERT statement.
- Recorded lectures and materials from www.wa4e.com:
 - jQuery ([Sample Code](#))

Additional Table Required for the Assignment

This assignment will add one more table to the database from the previous assignment. We will create a **Position** table and connect it to the **Profile** table with a many-to-one relationship.

```
CREATE TABLE Position (  
  position_id INTEGER NOT NULL AUTO_INCREMENT,  
  profile_id INTEGER,  
  rank INTEGER,  
  year INTEGER,  
  description TEXT,
```

```
PRIMARY KEY(position_id),

CONSTRAINT position_ibfk_1
    FOREIGN KEY (profile_id)
    REFERENCES Profile (profile_id)
    ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

There is no logical key for this table.

The **rank** column should be used to record the order in which the positions are to be displayed. Do not use the **year** as the sort key when viewing the data.

Including JQuery

You should include the JQuery JavaScript along with the bootstrap CSS in your code similar to the following:

```
<head>
...
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
      integrity="sha384-1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
      crossorigin="anonymous">

<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css"
      integrity="sha384-fLW2N01lMqjakBkx3l/M9EahuwP5feNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r"
      crossorigin="anonymous">

<script
      src="https://code.jquery.com/jquery-3.2.1.js"
      integrity="sha256-DZAnKJ/6XZ9Si04Hgrsxu/8s717jcIzLy3oi35EouyE="
      crossorigin="anonymous"></script>
...
</head>
```

The Screens for This Assignment

We will be extending the user interface of the previous assignment to implement this assignment. All of the requirements from the previous assignment still hold. In this section we will talk about the additional UI requirements.

- **add.php** You will need to have a section where the user can press a "+" button to add up to nine empty position entries. Each position entry includes a year (integer) and a description.
- **view.php** Will show all of the positions in an un-numbered list.
- **edit.php** Will support the addition of new position entries, the deletion of any or all of the existing entries, and the modification of any of the existing entries. After the "Save" is done, the data in the database should match whatever positions were on the screen and in the same order as the positions on the screen.
- **index.php** No change needed.
- **login.php** No change needed.
- **logout.php** No change needed.
- **delete.php** No change needed.

If the user goes to an add, edit, or delete script without being logged in, die with a message of "ACCESS DENIED".

You might notice that there are several common operations across these files. You might want to build a set of utility functions to avoid copying and pasting the same code over and over across several files.

Data validation

In addition to all of the validation requirements from the previous assignment, you must make sure that for all the positions both the year and description are non-blank and that the year is numeric.

All fields are required

or

Year must be numeric

Handling the Input From Multiple Positions

If you look at the sample implementation, it only allows a maximum of nine positions in the form. This is checked and enforced in the JavaScript for both the add.php and edit.php code.

The logic is somewhat simple and gets confusing when there is a combination of adds and deletes. It will never add more than nine new or total positions, but if you delete some of the positions, you do not get a position "back" to re-add unless you press "Save". So if you add eight positions and then delete five positions without pressing "Save", you can only add one more entry rather than four more entries.

This makes the JavaScript more simple and you are welcome to take the same approach.

The result is that if you add two positions and delete one position, you will end up with a form that looks like the following in the generated document object model:

```
<div id="position1">
<p>Year: <input type="text" name="year1" value="">
<input type="button" value="-" onclick="$('#position1').remove();return false;"></p>
<textarea name="desc1" rows="8" cols="80"></textarea>
</div>
<div id="position3">
<p>Year: <input type="text" name="year3" value="">
<input type="button" value="-" onclick="$('#position3').remove();return false;"></p>
<textarea name="desc3" rows="8" cols="80"></textarea>
</div>
```

In a sense we are simulating an array with the naming convention of the fields with the number at the end of the field. A way to handle multiple inputs with a naming convention like this is to use code like the following:

```
function validatePos() {
    for($i=1; $i<=9; $i++) {
        if ( ! isset($_POST['year'.$i]) ) continue;
        if ( ! isset($_POST['desc'.$i]) ) continue;
        $year = $_POST['year'.$i];
        $desc = $_POST['desc'.$i];
        if ( strlen($year) == 0 || strlen($desc) == 0 ) {
            return "All fields are required";
        }
    }
}
```

```

    }

    if ( ! is_numeric($year) ) {
        return "Position year must be numeric";
    }
}
return true;
}

```

Note that we handle gaps by simply checking the data that is present and skipping any data that is missing.

Setting the Foreign Key for Positions

When you are building the **add.php** code to add a new profile and some number of positions, you need to insert the **profile_id** as a foreign key for each of the position rows. But since you have not yet added the profile you do now know the **profile_id** which will be selected by the database.

Fortunately there is a way to ask PDO for the most recently inserted primary key after the insert has been done using the **lastInsertId()** method provided by PDO. Here is some sample code:

```

// Data is valid - time to insert
$stmt = $pdo->prepare('INSERT INTO Profile
    (user_id, first_name, last_name, email, headline, summary)
VALUES ( :uid, :fn, :ln, :em, :he, :su)');
$stmt->execute(array(
    ':uid' => $_SESSION['user_id'],
    ':fn' => $_POST['first_name'],
    ':ln' => $_POST['last_name'],
    ':em' => $_POST['email'],
    ':he' => $_POST['headline'],
    ':su' => $_POST['summary']
));
$profile_id = $pdo->lastInsertId();

...

$stmt = $pdo->prepare('INSERT INTO Position
    (profile_id, rank, year, description)
VALUES ( :pid, :rank, :year, :desc)');
$stmt->execute(array(
    ':pid' => $profile_id,
    ':rank' => $rank,
    ':year' => $year,
    ':desc' => $desc
));
$rank++;

```

The variable **\$profile_id** contains the primary key of the newly created profile so you can include it in the INSERT into the Position table.

Dealing with Changes to Positions When Editing

When you implement **edit.php** the user can do any combination of adds, removals, or edits of the position data. So when you are processing the incoming POST data, you need to somehow get the data in the database to match the incoming POST data.

One (difficult) approach is to retrieve the "old" positions from the database, and loop through all old positions and figure out which need to be deleted, updated, or inserted. If you want to try to do that for this assignment - feel free - but consider it an "extra challenge".

For your first implementation of handling the POST data in **edit.php** just delete all the old Position entries and re-insert them:

```
// Clear out the old position entries
$stmt = $pdo->prepare('DELETE FROM Position
    WHERE profile_id=:pid');
$stmt->execute(array( ':pid' => $_REQUEST['profile_id']));

// Insert the position entries
$rank = 1;
for($i=1; $i<=9; $i++) {
    if ( ! isset($_POST['year'.$i]) ) continue;
    if ( ! isset($_POST['desc'.$i]) ) continue;
    $year = $_POST['year'.$i];
    $desc = $_POST['desc'.$i];

    $stmt = $pdo->prepare('INSERT INTO Position
        (profile_id, rank, year, description)
        VALUES ( :pid, :rank, :year, :desc)');
    $stmt->execute(array(
        ':pid' => $_REQUEST['profile_id'],
        ':rank' => $rank,
        ':year' => $year,
        ':desc' => $desc
    ));
    $rank++;
}
```

This approach has the nice advantage that you are reusing code between **edit.php** and **add.php**. The only difference is that in **edit.php** you just remove the existing entries first.

What To Hand In

As a reminder, your code must meet all the specifications (including the general specifications) above. Just having good screen shots is not enough - we will look at your code to see if you made coding errors. For this assignment you will hand in:

1. A screen shot (including the URL) of your add.php showing two positions
2. A screen shot (including the URL) of your edit.php showing one position modified, one position deleted and one new position
3. A screen shot (including the URL) of your view.php showing the correct new positions after the edit is complete
4. A screen shot (including the URL) of your add.php showing the error message for a bad year
5. A screen shot of your Position database table showing at least three rows
6. Source code of add.php

7. Source code of view.php
8. Source code of edit.php

Optional Challenges

- Try to so the more inticate approach to updating positions in **edit.php** without using the "delete all the previous positions" trick. It will help if you add the **position_id** form markup that you generate for the positions that came from the database when **edit.php** starts.

General Specifications

Here are some general specifications for this assignment:

- You **must** use the PHP PDO database layer for this assignment. If you use the "mysql_" library routines or "mysqli" routines to access the database, you will **receive a zero on this assignment**.
- Your name must be in the title tag of the HTML for all of the pages for this assignment.
- All data that comes from the users must be properly escaped using the **htmlentities()** function in PHP. You do not need to escape text that is generated by your program.
- You must follow the POST-Redirect-GET pattern for all POST requests. This means when your program receives and processes a POST request, it must not generate any HTML as the HTTP response to that request. It must use the "header('Location: ...');" function and either "return" or "exit();" to send the location header and redirect the browser to the same or a different page.
- All error messages must be "flash-style" messages where the message is passed from a POST to a GET using the SESSION.
- Please do not use HTML5 in-browser data validation (i.e. type="number") for the fields in this assignment as we want to make sure you can properly do server side data validation. And in general, even when you do client-side data validation, you should still validate data on the server in case the user is using a non-HTML5 browser.

Database Setup Detail

[More ...](#)

Provided by: www.wa4e.com

Copyright Creative Commons Attribution 3.0 - Charles R. Severance