HW 6 – Teams CRUD App

Contents

[1 Overview 1](#_Toc196127522)

[2 General Tips & Recommendations 1](#_Toc196127523)

[3 *addTeams.php* 2](#_Toc196127524)

[4 *editTeams.php* – Select Mode 3](#_Toc196127525)

[5 *editTeams.php* – Edit Mode 5](#_Toc196127526)

[6 *editTeams.php* – Update & Cancel Modes 6](#_Toc196127527)

[7 *editTeams.php* – Delete Mode 6](#_Toc196127528)

[8 Video Demo 6](#_Toc196127529)

[9 Submission Requirements 7](#_Toc196127530)

[Appendix 1 n/a 8](#_Toc196127531)

# Overview

You will create an app that allows a user to add, edit, and delete teams from the *league* database. There is a video demoing the completed app posted on the Schedule showing the completed app. I recommend watching that before reading further.

The app consists of 2 pages: *addTeams.php* and *editTeams.php. addTeams* displays all teams and allows the user to add a new team. *editTeams* display all teams, allows a user to select a team and then edit values associated with the team, and to delete a team. In the next few sections, we’ll step through the requirements for each page and provide tips/hints.

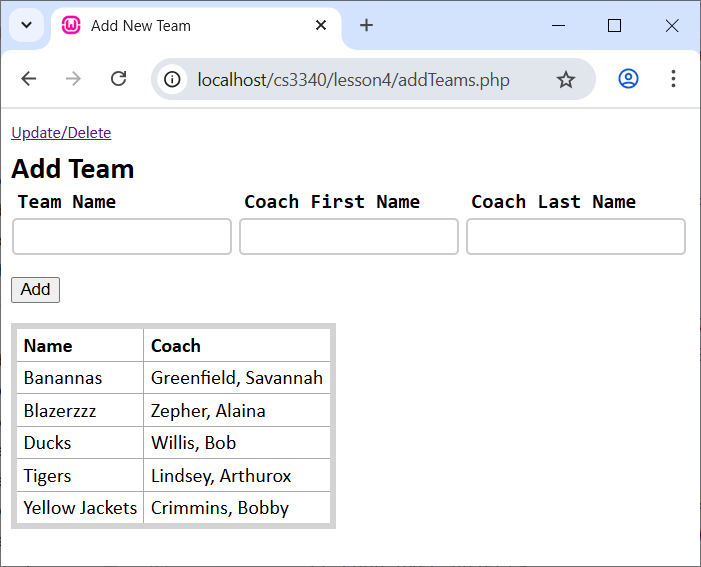
# General Tips & Recommendations

After coding this myself, I note the following:

* I recommend that you use methods for much of the work as possible. In my solution, I strived to write the shortest <body> for each of the pages, trying to show the logic that was taking place about which parts/actions needed to be displayed, and then using methods to do display.
* I only wrote “page methods”. Just functions I added at the end of the page. In other words, I did not add any methods to the Team nor Player class. Those are domain classes and should be kept separate from any display or database code.
* I added all functions after the end of the HTML. I wanted to get them out of the page, into a separate file(s), but I couldn’t decide how to organize them. I wanted to be able to separate them based on “display” and “SQL”, but often the code was heavily coupled. And, I couldn’t decide what to name those helper/utility files holding the functions that would be useful and predictive of what they contained. So, ultimately, I found it simpler to just put them all in the page itself including the *build\_connection* function, which is duplicated(!), appearing in both pages. I do not know what best practice is.
* Use baby-steps! When you need to write a function, start with something completely hard-coded. It might be: echo ‘hello world’; or perhaps something slightly more meaningful. The point is you don’t want to write code that has a lot more moving parts until you know that it will be called/displayed under the right circumstances. Similarly, when you want to write code to do an insert based on user-supplied data, hard-code the data straight into the insert and make sure it works first.
* You are welcome to follow the approach I outline below, or come up with your own approach. **The end result must be code that you wrote.**
* *Var\_dump(object)* is your best friend. It shows everything about an object: class, instance variables, data types, values, *etc.*

# *addTeams.php*

The *addTeams.php* page, when first launched (*i.e.* not a postback) appears as shown below when. Note that all teams are displayed in alphabetical order and that the coach’s name is displayed: *lastName, firstName*. A team can be added by supplying the data in the text fields and pressing *Add*. The page posts back to itself and displays the team that was added in the table. No data validation is needed. The *Update/Delete* link displays the *editTeams.php* page.



**Hints/Suggestions:**

1. My page body, is shown below. The actual methods I used are highlighted. Some methods utilize additional helper methods.

<body>  
 <a href="editTeams.php" class="smaller">Update/Delete</a>  
 <h2>Add Team</h2>  
 <form action="<?php echo $\_SERVER['PHP\_SELF']; ?>" method="POST">  
 <?php displayAddTeamInput(); ?>  
 <p><input type="submit" value="Add"></p>  
 </form>  
 <?php  
 if ($\_POST) {  
 insertTeam();  
 }  
 displayTeams();  
 ?>  
</body>

1. The *displayTeams* method is similar to what you did in HW 5, except you’ll be working teams instead of players. Also, the display of teams is static in a sense, it is just for the user’s reference, they don’t “select” anything from the display. Thus, the *TeamID* is not needed.
2. Sometimes, I found the [Heredoc](https://www.php.net/manual/en/language.types.string.php) syntax (scroll down to *Heredoc*) to be useful if you need to *echo* a large string of HTML. To do so, you use “<<<” followed by any identifier (I’ve used “END”). The closing “END;” must be in the first column(!). Such a statement can also contain variables, see: *$team[1]* below.

function displayStuff() {  
 echo <<< END  
 <table class="new\_team">  
 <tr>  
 <td><input type="text" name="name" value="$team[1]"></td>  
 </tr>  
 </table>  
END;

1. To add a team, you can use code like this:

$conn = build\_connection(); *// Create connection*$sql = "INSERT INTO teams (Name, CoachLastName, CoachFirstName) values (?,?,?)";  
$stmt = $conn->prepare($sql); *// Set query*

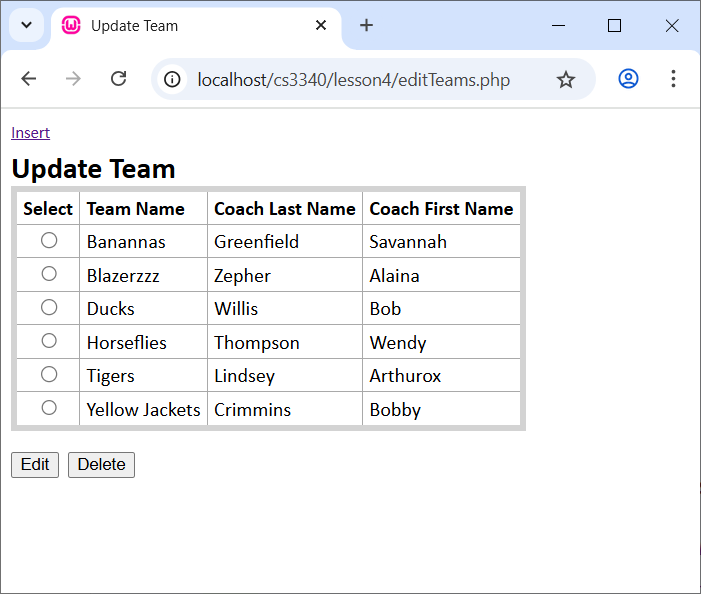
.*..*$stmt->execute([$name, $coachLastName, $coachFirstName]);  
$conn = null;  
$stmt = null;

Note:

* The *INSERT* statement uses parameters, *i.e.* the “?”’s in place of the values.
* The actual values are supplied as an argument to the *execute* method in the form of an array, [$name, $coachLastName, $coachFirstName]
* The actual values for the variables (*e.g $name, $coachLastName, $coachFirstName*) are obtained from the postback and would have been assigned in the code above (not shown!).
* We are not using the *Team* class in any way.

# *editTeams.php* – Select Mode

The *editTeams.php* page, when first launched (*i.e.* not a postback) appears as shown below. The *Insert* link displays the *addTeams.php* page. This section only considers the initial display, before any buttons have been pressed. The following sections discuss the various behaviors (states) of this page.



**Hints/Suggestions:**

1. My page body, is shown below. The actual methods I used are highlighted. Some methods utilize additional helper methods.

<body>  
 <a href="addTeams.php" class="smaller">Insert</a>  
 <h2>Update Team</h2>  
 <form action="<?php echo $\_SERVER['PHP\_SELF']; ?>" method="POST">  
 <?php  
 if (isset($\_POST['btn\_update'])) {  
 updateTeam();  
 echo "<p class='edit\_message'>Team updated</p>";  
 }  
 elseif (isset($\_POST['btn\_cancel'])) {  
 echo "<p class='edit\_message'>Update canceled</p>";  
 }  
 elseif (isset($\_POST['btn\_edit'])) {  
 displayTeamToEdit();  
 }  
 elseif (isset($\_POST['btn\_delete'])) {  
 if(deleteTeam()) {  
 echo "<p class='edit\_message'>Team deleted</p>";  
 }  
 else {  
 echo "<p class='edit\_message'>Can't delete team, players on team</p>";  
 }  
 }  
 displayTeamsForUpdate();  
 echo "<p><input type='submit' name='btn\_edit' value='Edit'>&nbsp;&nbsp;";  
 echo "<input type='submit' name='btn\_delete' value='Delete'></p>";  
 ?>  
 </form>  
</body>

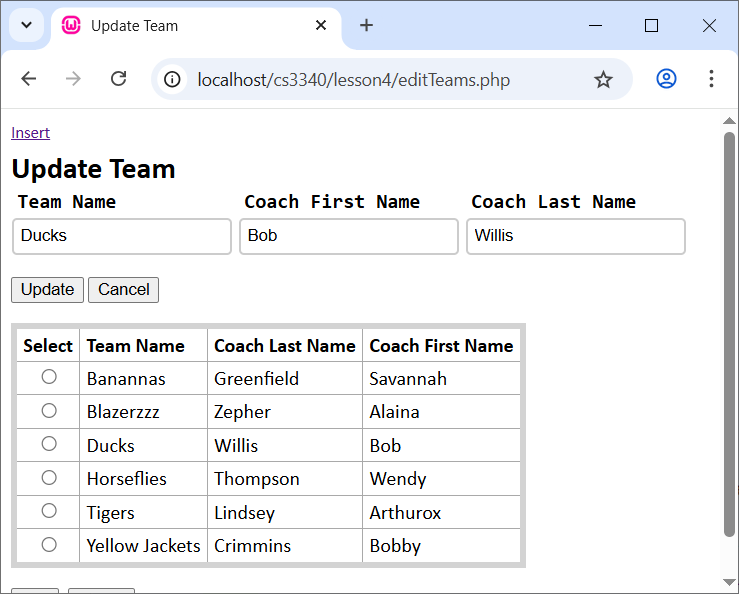
However, remember that *baby-steps* is the key! Don’t start with all that. Start with something like what is shown below. This just displays that table and buttons. The page posts back, but no update occurs.

<body>  
 <a href="addTeams.php" class="smaller">Insert</a>  
 <h2>Update Team</h2>  
 <form action="<?php echo $\_SERVER['PHP\_SELF']; ?>" method="POST">  
 <?php  
 displayTeamsForUpdate();  
 echo "<p><input type='submit' name='btn\_edit' value='Edit'>&nbsp;&nbsp;";  
 echo "<input type='submit' name='btn\_delete' value='Delete'></p>";  
 ?>  
 </form>  
</body>

1. The radio buttons in the table need to have the *value* attribute set to the *TeamID* so that when *Edit* is pressed and the page posts back, you’ll be able to know which team was selected for edit.

# *editTeams.php* – Edit Mode

When the *Edit* button is pressed, the display shows text fields and the user can enter data for a team. The display also has *Update* and *Cancel* buttons, which are discussed in following sections.



**Hints/Suggestions:**

1. When *Edit* is pressed, it posts back the radio button that is selected. From that, you can obtain the *Value* which should contain the *TeamID*. Then, that used in the SQL statement to *SELECT* that team from the database using a *WHERE* clause.
2. To select the team, I used code as shown below. Note: (a) I am using different methods than used in HW 5 (*prepare, execute, fetch*) and in the *addTeams.php* page, (b) The *teamID* must be converted to an *int* because that is what it is in the database, (c) I am not using the *Team* class, (d) The results of the *SELECT* statement in the *$team* variable which is an array (not a *Team* object). (e) To access the values, use: *$team[0]* to access the *TeamID*, *$team[1]* to access the (Team) *Name*, *etc.*

$conn = build\_connection();  
$stmt = $conn->prepare("SELECT \* FROM teams WHERE TeamID=?");  
$teamID = (int)$\_POST["teamID"];  
$stmt->execute([$teamID]);  
$team = $stmt->fetch();  
$conn = null;  
$stmt = null;

1. I used text fields to display the team to be edited and placed these inside a table with borders removed to display.
2. Remember when Update is pressed (considered in the next section), and posts back, you’ll need the *value* of the text fields so that you can update the database. Thus, you’ll need to give them a *name* attribute. You should choose something meaningful and consistent to avoid confusion. I got thoroughly confused using things like: team\_id, TeamID, teamID, team, *etc.* Finally, I went back to my code and insured that every variable across both pages, referencing say, *TeamID* database field, had a consistent identifier, and the same with the *name* attribute.
3. **Very Important**. When you press *Update* (considered in the next section), you need to know what team you are editing, *i.e.* you need to know the *TeamID*. Thus, you’ll need to stuff the *TeamID* into a *hidden field* in the page. You don’t want it to display it, but you do want it to post back. I stuck a line like this in a table row, but outside a *<td>* tag.

<input type="hidden" name="teamID" value="$team[0]">

# *editTeams.php* – Update & Cancel Modes

When *Update* is pressed, and postback occurs, you want to: (a) Issue the SQL *UPDATE* statement, (b) Redisplay the page without the edit area, (c) Display a message confirming the update. When *Cancel* is pressed, a postback occurs and a message is displayed stating that the update was aborted.

**Hints/Suggestions:**

To update a team, I used code as shown below

$conn = build\_connection();  
$sql = "UPDATE teams SET Name=?, CoachLastName=?, CoachFirstName=? WHERE TeamID=?";  
$stmt = $conn->prepare($sql);

...  
$stmt->execute([$name, $coachLastName, $coachFirstName, $teamID]);  
$conn = null;  
$stmt = null;

# *editTeams.php* – Delete Mode

When *Delete* is pressed, and postback occurs, you want to issue the SQL *DELETE* statement. If the team has no players, the team will be deleted and a confirmation message displayed. If the team does have players, then a message is displayed stating that the team cannot be deleted.

**Hints/Suggestions:**

1. To delete a team, I used code as shown below

$conn = build\_connection();$sql = "DELETE FROM teams WHERE TeamID=?";  
$stmt = $conn->prepare($sql);  
...  
$stmt->execute([$teamID]);  
$conn=null;  
$stmt = null;

1. If you use a *try/catch* around the delete code above, the catch block will activate if the user tries to delete a team with no players. Using that, you can figure out how to display an appropriate message.

# Video Demo

Make a video demoing your app using Kaltura Capture ([Directions](https://knowledge.kaltura.com/help/d2l-add-media-kaltura-capture-overview)) in Blazeview. Do the following in video:

1. Open myPhpAdmin and display your database.
   1. Open your *players* table and display the data
   2. Open your *teams* table and display the data
   3. Select the database and open the Designer
2. Display *addTeams.php*
   1. Add a team
   2. Indicate that it was added to table
   3. Follow the “Update/Delete” link
3. On *editTeams.php:*
   1. Indicate the new team that was added in the select team table.
   2. Select a team to edit, make some changes, update, and then indicate that it was updated in the table.
   3. Select a team to edit, then Cancel.
   4. Select a team with no players and Delete. Indicate that that team is no longer present in table.
   5. Return to *addTeams.php* and indicate that team is no longer present in table. Return to *editTeams.php*.
   6. Select a team with players and Delete. Indicate message displayed and that team was not deleted.

# Submission Requirements

1. Zip the following files into a file named: *hw6\_yourLastName.zip*:
2. *addTeams.php*
3. *editTeams.php,*
4. *Team.php*
5. *Player.php*
6. any other files you created to support your app, including a *css* file.
7. Login into Blazeview, choose HW 6 assignment.
   1. Add the *hw6\_yourLastName.zip* file
   2. In the Comments area, choose: “Insert Stuff”, and then choose: Embed Kaltura Video and choose your video.

Appendix

1. n/a