PHP PDO MySQL demo

Introduction

The general purpose of this demo is to educate developers about PHP's PDO (PHP Data Objects) interface and how to get started using it. In addition, to help developers understand SQL injection and how it can be avoided, this demo looks at using the PHP PDO class to access a database in an interactive web application. By following the included example scenario of SQL injection and how it was addressed with the PDO class, developers should be able to transfer their existing knowledge of using the mysqli_* functions that they may be accustomed to using and begin translating that into new knowledge that will lead them to creating more secure web sites and web applications of their own. Finally, we take a look at using my custom PDO_MySQL class that aims to help simplify and streamline the process of switching to using prepared statements with the PHP PDO interface.
These files are available on GitHub in this repo: PDO_MySQL

Getting started with this demo

To get started with your own live version of this demo:

- Download the project and startup a localhost server such as WAMP, MAMP, LAMP, or XAMPP.
    - I created and tested this using XAMPP version 5.6.3,

with Apache version 2.4.10, PHP version 5.6.14, and MySQL version 5.6.21.

- Place the project in the htdocs folder in which the localhost webserver will host the files. Navigate to the directory in your browser.
- In phpMyAdmin use the import utility to import the `pokemon.sql` file. This will create a new database with the populated species table. This is the database the demo will be referencing.
- Finally, replace all the ‘username’ and ‘password’ database login credentials at the top of each page to use the appropriate ones for a user in your database.

The mysqli_* old method

References demo_old_way.php

The following demonstrates the syntax of `mysqli_*` functions and their usage to query a database. In this example, we’ll be searching for pokemon that are similar to what the user types into the box.

```
<?php
    //define our connection settings
    $host = "localhost";
    $dbuser = "username";
    $dbpass = "password";
    $dbname = "pokemon";
    //create a variable to hold the reference to our data
base connection
```

```
    $db = mysqli_connect($host,$dbuser,$dbpass,$dbname);
?>
<table>
<?php
    //check to see if the database is connected before co
ntinuing and attempting to query it
    if($db){
        if(isset($_GET['pkmn_name'])){
            //get the user's input from the GET global ar
ray
            $name = $_GET['pkmn_name'];
            //define the query
            $query = "SELECT SPECIES_NAME, TYPE1 FROM SPE
CIES WHERE SPECIES_NAME LIKE '%".$name."%'";
            //query the database using the mysqli_query()
 function
            $result = mysqli_query($db,$query);
            //check to see if the query was successful
            if($result){
                //use mysqli_num_rows() function to deter
mine if the query returned any rows
                if(mysqli_num_rows($result) > 0){
                    //while there are still rows left in
the result set, continue iterating and writing out the ta
ble rows with the data from the record
                    while($row = mysqli_fetch_assoc($resu
lt)){
                        echo("<tr><td>".$row['SPECIES_NAM
```

```
E']."</td><td>".$row['TYPE1']."</td></tr>");
                    }//end while
                }//end mysqli_num_rows
            }//end check for successful query
        }//end isset
    }//end check to see if connected
?>
</table>
```

You will notice that simple pokemon names, such as *Pikachu* for example, will work fine. However, say you want to find Farfetch'd. Can you search for his entire name?

This breaks because the single quote creates an incomplete SQL query string.
Also, if you would try entering: `' OR 1=1;--` (include a space at the end)
You will notice this returns everything in the table of pokemon, because 1=1 always is true. Likewise, if you would put any other valid SQL there, it would also run. **Be careful!**
Entering something like `' OR 1=1; DROP * FROM species;--` would query just fine and **would delete all your data**.
To fix this we will use the PHP PDO class and use prepared statements.

The PHP PDO new method

References demo_new_way.php

This demonstrates the syntax of the PDO interface for MySQL and its usage to query the database for pokemon that are similar to what the user types into the box.

```
<?php
    //define our connection settings
    $host = "localhost";
    $dbuser = "username";
    $dbpass = "password";
    $dbname = "pokemon";
    //create a new instance of the PDO class by stating w
e want the mysql: driver
    //also it needs our connection settings
    $db = new PDO('mysql:host='.$host.';dbname='.$dbname.
';charset=utf8', $dbuser, $dbpass);
    //set the error mode to exception so they can be used
 with try/catch
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXC
EPTION);
    //set the emulate prepares to false so it actually us
es prepared statements
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
?>
<table>
<?php
    //check to see if the database is connected before co
ntinuing and attempting to query it
```

```
    if($db){
        if(isset($_GET['pkmn_name'])){
            //get the user's input from the GET global ar
ray
            $name = $_GET['pkmn_name'];
            $name = "%".$name."%";//need to include the w
ildcard characters as input to the query
            //define the query
            $query = "SELECT SPECIES_NAME, TYPE1 FROM SPE
CIES WHERE SPECIES_NAME LIKE ?";
            //create a $stmt variable to hold our working
 statemnt and use the prepare() function to prepare the S
QL
            $stmt = $db->prepare($query);
            //bind the $name variable to the first ? in t
he SQL statement. Tell it to treat it like a stirng
            $stmt->bindValue(1, $name, PDO::PARAM_STR);
            //execute the statement
            $stmt->execute();
            //check to see if the statement was successfu
l
            if($stmt){
                //use rowCount() function to determine if
 the query successfully returned any results
                if($stmt->rowCount() > 0){
                    //while there are still rows left in
the result set, continue iterating and writing out the ta
ble rows with the data from the record
```

```
                    while($row = $stmt->fetch(PDO::FETCH_
ASSOC)){
                        echo("<tr><td>".$row['SPECIES_NAM
E']."</td><td>".$row['TYPE1']."</td></tr>");
                    }//end while
                }//end rowCount
            }//end check for successful query
        }//end isset
    }//end check to see if connected
?>
</table>
```

You will notice that for pokemon names, such as *Pikachu* for example, will work just as well as a name like *Farfetch'd*. Now type jsut a single quote and see what happens.

This works because we have prepared the SQL statement by telling the database to only treat this input as strictly input and not as additional SQL.
Again, try entering: `' OR 1=1;--` (include a space at the end)
You will notice this **does not work**, which is good!. This is because it looks for pokemon names that look like exactly like that and there of course are none!

To see a better, *err rather more intuitive*, way to use the PDO interface, see my custom PDO_MySQL class which accesses the PDO class using similar methods, structure, and logic flow as in the older `mysqli_*`

methods, which many people are accustom to.

It also has some additional little useful utilities for more advanced web app setups.

The PDO_MySQL custom PHP class

References demo_better_way.php

This demonstrates the syntax of my custom PDO_MySQL class and its streamlined PDO usage to prepare statements and query the database. You will notice that this still works for any pokemon names.

```
<?php
    require('pdo_mysql_class.php');//neededed to import t
he custom PDO_MySQL class into the program
    //define our connection settings
    $host = "localhost";
    $dbuser = "username";
    $dbpass = "password";
    $dbname = "pokemon";
    $db = new PDO_MySQL();//create a new instance of the
class
    $db->connect($host,$dbuser,$dbpass,$dbname);//use the
 connect function
?>
<table>
<?php
```

```
    //check to see if the database is connected before co
ntinuing and attempting to query it
    if($db->is_connected()){
        if(isset($_GET['pkmn_name'])){
            //get the user's input from the GET global ar
ray
            $name = $_GET['pkmn_name'];
            $name = "%".$name."%";//need to include the w
ildcard characters as input to the query
            //define the query
            $query = "SELECT SPECIES_NAME, TYPE1 FROM SPE
CIES WHERE SPECIES_NAME LIKE ?";
            //query the database using the query_prepared
() function to have it automatically prepare the SQL stat
ement
            //and pass it an array of the variables that
match the ? in the statement in the same order as they ap
pear
            $result = $db->query_prepared($query,array($n
ame));
            //use the has_rows() function to determine if
 the query successfully returned any results
            if($db->has_rows($result)){
                 //while there are still rows left in the
 result set, continue iterating and writing out the table
 rows with the data from the record
                while($row = $db->fetch_row($result)){
                    echo("<tr><td>".$row['SPECIES_NAME'].
```

```
"</td><td>".$row['TYPE1']."</td></tr>");
                }//end while
            }//end has_rows
        }//end isset
    }//end is_connected
?>
</table>
```

This works because we have used the PDO interface in the backend which is set up to automatically prepare all SQL statements and perform all the variable binding behind the scenes, allowing the developer to focus more on creating their application rather than having to consciously perform all the steps themselves.
Again, try entering: `' OR 1=1;--` (include a space at the end)
You will notice this **still does not work**, which is good!.
When you are satisfied with this example and understand this simple functionality of the class, take a look at the `pdo_mysql_class.php` file and read the commented usage for a more complete understanding of what it is set up for and capable of.

Completed PHP PDO for MySQL Demo

References demo_final.php

This is the completed form of this demo using my custom PDO_MySQL class for connecting to a MySQL database and querying using prepared statements.

Also introduced are just a few of its handy utilities and functionality I have provided within the class.
Also note, there is some JavaScript used in this example to get the sounds to play when the pokemon sprites are clicked.
Turn your sound up!

```
<?php
    require('pdo_mysql_class.php');//neededed to import t
he custom PDO_MySQL class into the program
    $db = new PDO_MySQL();//create a new instance of the
class.
    $db->connect("localhost","username","password","pokem
on");//use the connect function
?>
<?php
    //check to make sure the database is connected before
 continuing and attempting to query it
    if($db->is_connected()){
        if(isset($_GET['pkmn_name'])){
            //get the user's input from the GET global ar
ray
            //need to include the wildcard characters as
the input to the query
            $name = "%".$_GET['pkmn_name']."%";
            //define the query
            $query = "SELECT SPECIES_CRY_BLOB, SPECIES_IM
G_BLOB, SPECIES_ID, SPECIES_NAME, TYPE1, TYPE2 FROM SPECI
ES WHERE SPECIES_NAME LIKE ?";
```

```
            //query the database using the query_prepared
() function to have it automatically prepare the SQL stat
ement
            //and pass it an array of the variables that
match the ? in the statement in the same order as they ap
pear
            $result = $db->query_prepared($query,[$name])
;
            //print out to the user the number of matched
 results
            echo("<h4>".$db->num_rows($result)." resutls
found</h4>");
            //use the has_rows() function to determine if
 the query successfully returned any results
            if($db->has_rows($result)){
                //use the field_names() function to get a
n array of the column names in the database that were ret
urned by the associative array when queried
                $fields = $db->field_names($result);
                echo("<thead>");
                //write out the table header using the fi
eld names
                for($i=1;$i<sizeof($fields);$i++){
                    $fieldname = $fields[$i];
                    echo("<th>$fieldname</th>");
                }
                echo("</thead>");
                echo("<tbody>");
```

```
                //while there are still rows left in the
result set, continue iterating and writing out the table
rows with the data from the record
                while($row = $db->fetch_row($result)){
                    echo("<tr>");
                    echo( "<td><img class='species_img' s
rc='".base64_encode($row['SPECIES_I
MG_BLOB'])."' data-wav='".base64_encode($row['SPECIES_CRY
_BLOB'])."' alt='{$row['SPECIES_NAME']}' draggable='false
'/></td>");
                    $j = 0;
                    foreach($row as $item){
                        if($j > 1){
                            $field = $item;
                            if($field=="") $field = "(non
e)";
                            echo("<td>$field</td>");
                        }
                        $j++;
                    }//end foreach
                    echo("</tr>");
                }//end while
                echo("</tbody>");
            }//end has_rows
        }//end isset
    }//end is_connected
?>
</table>
```

```
<script type="text/javascript">
var sprites = document.getElementsByClassName('species_im
g');
var audioSrc = document.getElementById('#cry');
for(var i=0;i<sprites.length;i++){
    sprites[i].addEventListener('click',function(e){
        var data = e.target.getAttribute('data-wav');
        cry.setAttribute('src','data:audio/wav;base64,'+d
ata);
        cry.play();
    });
}
</script>
```

Conclusion

In conclusion you can see that by not validating and sanitizing user input it could lead to serious concequences. The best way to currently combat this problem from both a security and a developer’s standpoint is to be using prepared statements whenever possible. Hopefully by the end of this demo you can get an understanding of the concepts of SQL injection and how you can prevent it in your own projects. Also, I hope that this demo is helpful to both new and seasoned PHP coders, web devs, and anyone trying to be more secure with their database interactions on the web.

If you enjoyed this please leave feedback, suggestions, contribute, or simply let me know if this has helped you.
As mentioned in the introduction, my custom PDO_MySQL project is available in a separate repository: PDO_MySQL on GitHub.

Contact:

Email: rwc4@pct.edu

Website: ryanc16.com

Github: ryanc16