PHP MySQL vs. Unity

Introduction

When using the Unity game engine, there are methods to connect your game to a MySQL database over the internet. The easiest way is to use JavaScript and PHP, especially for people with a more artistic approach because it is better to have a more basic approach to coding, instead of wasting a lot of time learning some strange programming language. But, - yes there is a but- for this method you also need to know your PHP. Luckily, the PHP part isn't that hard and the MySQL part isn't also.

The PHP side

Why learn PHP instead of PERL? The answer is simple: PHP is easy to understand and widely used. Every time you build a website, PHP always comes in very handy. It is also very save, because you can restrict it easily and if you post your Unity code online, people won't see passwords for access to your precious database and they can't alter the code so they can ruin it. So what does the PHP side do? It simply accepts HTML FORM input which a browser can send in and after that it can send data back. The FORM element in HTML is a webform with which you can post data to a website. This way you can post a variable with the name *'action'*, which can hold as value an action that needs to be performed. For example with the HTML GET function:

"http://myHost.com/PHPScript.php?action=display&text=hello"

You can see clearly that there are two variables posted to the PHPScript.php, one with the action *"display"* and the text that needs to be displayed *"hello"*. The PHP script will probably look like this:

```
<?php
        $action = $_REQUEST['action'];
        if($action == "display") {
                echo $_REQUEST['text'];
        }
?>
```

These few lines print the text given in the URL to the screen. And of course you can have different values for *'action'*, so you can describe different actions that need to be performed.

The Unity side

So how do you have Unity get the URL to the website and the information back to the game? The answer is simple. If you know HTML, you know of the FORM element, which is very basic. There are two methods of posting the information of that FORM element to the PHP script that processes that information. One is the GET method, which will add the variables and the values of those variables to the end of the URL just like in the example above where PHP can read out the URL. The other one is the POST method which will post that information invisible to you to the PHP script. Unity can load text from a website with the *'WWW("http://myURL.com")'* function. It just reads the whole text that the server returns and since it isn't a browser it's just plain text: HTML-tags will also be just plain text if you do not process them further in Unity. You can also download an image with it, if you put in a

URL directly to an image. With this function you can also use the *‘WWW()’* function with a GET method. In this way, you get something like:

```
var w = WWW(“http://myHost.com/PHPScript.php?action=display&text=hello”);
yield w;
```

‘w’ contains the plain text, which in this case will be the output of the PHP script: *“hello”*. *‘yield w;’* means that the program must wait until all the data is loaded from the website before it can continue. But there is a more nice way of doing this by using the POST method instead of the GET method. Unity has also a *‘WWWForm()’* function. This function creates a FORM, which you can post. It will look like this:

```
var form = new WWWForm();
form.addField(“action”, “display”);
form.addField(“text”,”hello”);
var w = WWW(“http://myHost.com/PHPScript.php”, form);
yield w;
```

So you first declare a variable *‘form’* which is a FORM element. Then you just add fields to the form, which are variables and the values of the variables. After that you post the form to the URL which you give in the *‘WWW()’* function. *‘w.data’* will contain the string *“hello”*. It is important to know that it is a string, so if you have a numerical variable you will first need to transform it to a float or an integer or something like that before you can do math with it.

The MySQL side

Now we come to the actual MySQL side of it all. So if you take your PHP script, you can combine it with MySQL. For example, we are going to use a high score table. MySQL is nothing more than a bunch of tables collected into a database. For each table you can define the number of columns, the names of each column and the number of rows and you can add, delete, rename all of them. So we have a database called *‘unity’*, which contains the table *´highscores’*. The highscore table has three columns: *“id”,”name”, “score”.* So the table would look like this:

Id	**Name**	**Score**
1	Joost	500
2	Piet	720
3	Klaas	350

The PHP and MySQL could be like this:

```
<?php
	mysql_connect("localhost","root","");
	mysql_select_db("unity");
	if($_REQUEST['action']=="show_highscore") {
		$query = "SELECT * FROM `highscores` ORDER BY `score` DESC";
		$result = mysql_query($query);
		while($array = mysql_fetch_array($result)) {
			echo $array['name']."</next>";
			echo $array['score']."</next>";
		}
	}
	if($_REQUEST['action']=="submit_highscore") {
		$name = $_REQUEST['name'];
		$score = $_REQUEST['score'];
		$query = "INSERT INTO `highscores` (`name`,`score`) VALUES ('$name','$score')";
		mysql_query($query);
	}
?>
```

If you have never used MySQL, I'm going to explain it a little bit and I suggest that you go to google and search for a tutorial in how to create a table, but for the moment we have this script. I'll go through it line by line:

- *Connects to the correct database host*
- *Selects the appropriate database*
- *Checks if the action that is submitted is to show high scores*
- *A MySQL command, which will fetch scores from the table 'highscores' and sorts it by score*
- *Runs the MySQL command of the previous line*
- *This is to read out every row in the table*
- *Prints every "name" to the screen, closed by an "</next>"*
- *Prints ever "score" to the screen, closed by an "</next>"*

- *Checks if the action that is submitted is to submit a high score*
- *Collects the posted ´name'*
- *Collects the posted ´score'*
- *A MySQL command, which will insert a new row into the table 'highscores' with the ´name' and ´score', the ´id' will be added automatically so that every row has an unique 'id'.*
- *This runs the MySQL command of the previous line*

- *Closes the PHP part*

I hope you understand it now a little bit better and we will move on to the Unity part, which will use this PHP script. Remember: this is all very basic, so you can add stuff in the PHP script like checking the probability of a score, so you can shield you database from cheaters and such. The *"</next>"* is there so you can split the long string that gets posted, which will be handled by Unity.

The last part: the unity and MySQL part

So the MySQL part will run, but still Unity can't post a high score. To solve that problem, here is the code to do that, with explanations at the end for each line.

```
function submit_highscore(player_name,player_score) {
        var form = new WWWForm();
        form.AddField("action","submit_highscore");
        form.AddField("name",player_name);
        form.AddField("score",player_score);
        var url = "http://localhost/unity/highScores.php";
        var w = WWW(url,form);
        yield w;
}

function show_highscore() {
        var form = new WWWForm();
        form.AddField("action","show_highscore");
        var url = "http://localhost/unity/highScores.php";
        var w = WWW(url,form);
        yield w;
        received_data = Regex.Split(w.data,"</next>");
        scores = (received_data.Length-1)/2;
        for(var i = 0; i<scores;i++) {
                print("Name: "+received_data[2*i]+"    Score: "+received_data[2*i+1]);
        }
}
```

- *The function that will submit a high score, which needs to have a name and a score as input*
- *The form that will be posted*
- *The 'action' is added to the form, which is to submit a high score*
- *The 'name' is added to the form, which contains the given player name*
- *The 'score is added to the form, which contains the given players' score*
- *The url of the website that contains the database and the PHP script*
- *The function that will post the form to the website*
- *The function that waits until it all is loaded*

- *The function that reads the high scores out of the database*
- *The form that will be posted*
- *The 'action' is added to the form, which is to show the high scores*
- *The form is posted to the URL*
- *The information from the website is being downloaded*
- *'Regex.Split()' splits, in this case, the long string in parts after every "</next>" and stores those parts as the array 'received_data'. The array looks like this:*

  { name1,score1,name2,score2,name3,score3, (and so forth) }

- *Scores is the number of scores that are submitted. You can see why it is the length of the array divided by two, because of that the names and scores are in the same array, and the -1 is because after the last score there is also a "</next>" which will be split and outputs an "" at the end.*
- *Have a loop for every score*
- *Prints the name of every player and the score of every player.*

Conclusion

The method I described here is easy to use and it is save. This is because the processing of the data is not client-sided, but it is server-sided. Also the PHP knowledge you need to have is fairly minimal if you want to do easy stuff and on the internet there are tons of tutorials to learn it yourself. The JavaScript side is also fairly simple. If you use this way to connect Unity to a MySQL database, you can easily customize it to your own needs. You can for example also post screenshots of your game directly to the database in an easy way, but you must search for that for yourself. So if you want a easy, save and highly adjustable way of Unity with a database, use the WWWForm()!

Links

http://unity3d.com
http://unity3d.com/support/documentation/ScriptReference/index.html
http://unity3d.com/support/documentation/ScriptReference/WWWForm.html
http://unity3d.com/support/documentation/ScriptReference/WWW.html