MySQLi Easy Prepared Statements

Thank you for purchasing this script from. The purpose of this script is to allow you to easily and securely query a MySQL database in PHP using prepared statements.

Please feel free to contact us on CodeCanyon if you need help or clarification with anything.

First steps

We need to start off by including the class in our code, so you need to put the follow line of code in before you you call anything in the class:

```
require(“easypreparedstatements.class.php”);
```

Now you’ve got the script included, the next step is instantiate the helper, so next we’ll call the following:

```
$easypreparedstatements = new easyPreparedStatements("host", "username",
"password", "databasename");
```

In order to get this to work:
- replace “host” with your MySQL host, this is usually “localhost”
- replace “username” with your MySQL username, this, by default is “root”
- replace “password” with your MySQL password, this can be left blank
- replace “databasename” with the name of your database, you chose this name when you created the database.

Note you can instantiate the MySQLi helper class as many times as you want, just be sure to change the variable name.

Now save your PHP script and run it, it should return a blank page; if an error message appears make sure that the credentials you entered are correct, that the credentials have permission to access the database and that the mysqlihelper.class.php script exists where you’ve included it.

After we’ve got the helper class included and instantiated we’re ready to move on and start using the script!

Prepared Statements

The query function of the script has the following syntax:
`$easypreparedstatements->query($query, $params=array())`

In order the query goes into the $query variable, and the data supplied goes in the `$params` array. In your SQL query all the variables you want to pass are represented with question marks (“`?`”). This may be a bit confusing here, but let’s go through a few examples to clear things up.

Select Statement
Let’s suppose we have a blog and want to select the id, title and content from the posts table, we can do so with the following SQL statement:

```
SELECT id, title, post FROM posts WHERE `category`= tech
```

The first step is in your SQL statement is to replace all variables you want to pass into the statement with question marks, in this case I want to pass the category name into the query:

```
SELECT id, title, post FROM posts WHERE `category`= ?
```

You can then put this all together and use the query function in Easy Prepared Statements class to run this in PHP:

```
$blogposts = $easypreparedstatements->query("SELECT id, title, post FROM
posts WHERE `category`= ?", array("tech"));
```

Now the $blogposts variable will contain an array of everything the query has returned. You can print out all this data using:

```
print_r($blogposts);
```

You can loop through the results using the foreach function in PHP.

Insert Statement

Like the select statement above we can do a insert as follows:

```
$easypreparedstatements->query("INSERT INTO `posts` (`id`, `title`,
`post`, `time`, `category`) VALUES ('?', '?', '?', '?', '?')",
array(uniqid("",TRUE), "Title", "Post", time(), "tech"));
```

On success this function will return the number of affected rows.

Update, delete, create table, replace, etc

Like above, the class supports UPDATE, DELETE, CREATE TABLE, REPLACE amongst other MySQL query types. Just, using the syntax above, replace the query with the MySQL query (with variables as question marks) and plug in all your variables and you're all set.

Data returned by query function

When you run a query using this class, using either the preparedQuery or query function; the data that the function will return depends on what query type you run. So here's a table that sums them up:

Query type	Data returned
Error (invalid query) or a query that failed	FALSE
SELECT	Data from SELECT query (as an array).

INSERT
UPDATE
DELETE
	Number of affected rows from the query.

CREATE TABLE REPLACE Other MySQL queries (not mentioned above)	TRUE (on success)

Defining MySQL variable types yourself

So, in this script when you use query you don't need to bind variables yourself. Though, if this this is something you want to do we have made this possible.

So basically instead of using the query function in the class you use the preparedQuery function which has a similar syntax as the query function:

```
$easypreparedstatements->preparedQuery($query, $params=array())
```

The difference is however, that the first key in the $params variable must be the character keys for the variable keys, for example:

```
$blogposts = $easypreparedstatements->preparedQuery("SELECT id, title, 
post FROM posts WHERE `category`= ?", array(“s”,“tech”));
```

See the “s” as the first key in the parameters array? If you are passing more than 1 variable, just add them to the string. For example, if my first variable was a string, the next an integer, then a blob (contents of a file) and finally a double variable (a number with decimal places) the first parameter in the $params array would be “sibd”.

Here’s a table that sums up the variable characters to the corresponding variable type:

Letters	Variable type
s	string
i	integer
d	double
b	blob

That’s it; is it?

So, that’s all there is in this class. If you need any advice or want to ask me any questions; feel free to drop me a question on CodeCanyon. Thanks again for your purchase!