Prepared Statements in PHP and MySQLi

Apr
09

*By **Matt Bango***

This article is intended for readers who have experience using PHP and MySQL. You should also have a general understanding of databases and programming (both procedural and object-oriented) as well as how to use PHP to execute a simple query to MySQL. I will not cover how to install PHP or MySQL, however at the end of the article are some links to help you get started with the installation process and for some further reading on the subject. I will be covering the basics of **prepared statements** in PHP and MySQLi and why you should consider using them in your own code as well as some technical explanation as to why you should use them.

Introduction

If you are like me and most other people, you probably have not taken the time to learn about web security when you first started writing server-side code. This is very dangerous as most people never even go back and try to make their code secure (or they simply forget). Writing their code in the same way that they originally learned how to can cause some serious vulnerabilities in the code, allowing hacking techniques such as SQL injections to be fairly easy. If you have no idea what MySQL injections or cross side scripting is, then you should do some research, for example just go to Google and type in "SQL injections" and there will be plenty of reading for you. I also would recommend a book called, **"**How to Break Web Software**"**, it is a fantastic book that one of my professors told one of my classes about. It can teach you **a lot** about security, it is highly recommended. I will have an article written shortly on SQL Injections, so check back soon! If you do know what some of these nasty hacking techniques are then you are probably wondering why you should want to use prepared statements. There are basically three reasons why you should seriously consider writing prepared statements to execute your queries.

I. Prepared statements are **more secure**.
II. Prepared statements have **better performance**.
III. Prepared statements are **more convenient to write**.

Now that we know why prepared statements are better, let's build an example so you can see for yourself. We'll build a simple login example using prepared statements. First, I'll show you the way most people would write it, then I'll show you the way you could do it with a prepared statement which will be **more secure, have better performance** and **be more convenient to write**. Let's get started!

The Well-Known Way

If you are reading this article, chances are you already know how to execute a simple MySQL query in PHP. For those of you who do not know how to do this, it would look similar to this:

```
<?php
   /* Connect to the Database */
   $dbLink = mysql_connect("localhost", "username", "password");

   if (!dbLink) {
      echo 'db link fail';
   }

   /* Select the database */
   mysql_select_db("databaseName");

   /* Query and get the results */
   $query = “SELECT * FROM testUsers WHERE username='$user' AND password='$pass'";
   $result = mysql_query($query);

   /* Loop through the results */
   while($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
      echo “Username: “ . $row[‘username’;
   }
?>
```

What is the problem with this code? Simple, someone could use a simple SQL injection to get around the password authentication. Why is this code dangerous? If you know what an SQL injection does, it basically bypasses the password condition by commenting it out and uses an always true statement which allows access. Building strings on the fly like this should make you **very** nervous, but how do we make it more secure? Say hello to prepared statements.

Prepared Statements

What is so great about prepared statements and why are they more secure? The simple answer is because prepared statements can help increase security by separating the SQL logic from the data being supplied. In the previous example we saw how the data is basically built into the SQL logic by building the query as a string on the fly. Let’s take a look at what a prepared statement can look like.

```
<?php
   /* Create a new mysqli object with database connection parameters */
   $mysqli = new mysql('localhost', 'username', 'password', 'db');

   if(mysqli_connect_errno()) {
      echo "Connection Failed: " . mysqli_connect_errno();
      exit();
   }

   /* Create a prepared statement */
   if($stmt = $mysqli -> prepare("SELECT priv FROM testUsers WHERE username=? AND
password=?")) {

      /* Bind parameters
         s - string, b - boolean, i - int, etc */
      $stmt -> bind_param("ss", $user, $pass);

      /* Execute it */
      $stmt -> execute();
```

```
      /* Bind results */
      $stmt -> bind_results($result);

      /* Fetch the value */
      $stmt -> fetch();

      echo $user . "'s level of priviledges is " . $result;

      /* Close statement */
      $stmt -> close();
   }

   /* Close connection */
   $mysqli -> close();
?>
```

Doesn't look too bad, right? In short, the above code basically creates a new mysqli object and connects to the database. We then create a prepared statement and bind the incoming parameters to that statement, execute it and get the result. We then close the statement and connect and we're done! Pretty easy!

Let's take a look at where the security happens in these few lines:

```
   if($stmt = $mysqli -> prepare("SELECT priv FROM testUsers WHERE username=? AND
password=?")) {

      $stmt -> bind_param("ss", $user, $pass);
```

Instead of grabbing and building the query string using things like $_GET['username'], we have ?'s instead. These ?'s separate the SQL logic from the data. The ?'s are place holders until the next line where we bind our parameters to be the username and password. The rest of the code is pretty much just calling methods which you can read about by following some of the links at the end of the article.

Summary

In this article we have covered why and how you should use prepared statements. You should now have a solid understanding of the benefits associated with using prepared statements as well as how to use prepared statements. If you did not know before, after reading this article you should have a basic understanding of the object-oriented interface to MySQLi. I hope this was helpful to you and if you have any questions feel free to post some comments below!

Further Reading

I. MySQLi Manual – The MySQLi manual that is on PHP.net.
II. PHP.net – Excellent PHP resource.
III. Prepared Statements – Pretty awesome article over at Database Journal on prepared statements.
IV. Installing PHP and MySQLi – Good installation tutorial.