

Forum:[Hack Forums](#)

Thread: [\[The Empire\] An introduction to PDO](#)

1n9i9c7om - March 28, 2014, 7:45 am

[img]http://imgazer.imageshack.us/a/img90/6030/bannermo.gif[/img]

What is PDO?

The [information about PDO on php.net](#) are very good, so I decided to just quote them here:

PHP.net Wrote: The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions. Note that you cannot perform any database functions using the PDO extension by itself; you must use a database-specific PDO driver to access a database server.

PDO provides a data-access abstraction layer, which means that, regardless of which database you're using, you use the same functions to issue queries and fetch data. PDO does not provide a database abstraction; it doesn't rewrite SQL or emulate missing features. You should use a full-blown abstraction layer if you need that facility.

TL;DR: You can use it to connect to a database and run queries.

This tutorial will be based on using MySQL with PDO, but it isn't very different with other database engines as well.

Connect to a database

The code:

PHP Code

```
<?php
```

```
$dbc = new PDO('mysql:dbname=Database;host=127.0.0.1;charset=utf8', 'username', 'password');
```

The explanation:

This line of code creates a new PDO object and saves it into the variable "\$dbc". The constructor takes 3 arguments: The [DSN connection string](#), username and password.

In your development environment, you should also use this line:

PHP Code

```
<?php
```

```
$dbc->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

This will throw an exception when an error occurs, you should use it for debugging, make sure to remove this line or change it to this:

PHP Code

```
<?php
```

```
$dbc->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
```

Prepared statements

PDO has support for [prepared statements](#).

Prepared statements will help you to keep your queries secure, as you don't have to use something

like mysql's "mysql_real_escape_string" on every user-input anymore.

That's the main reason why you should use PDO or mysqli over mysql.

I'll explain prepared statements together with explaining how to run queries now.

Select from database

The code:

Without prepared statements:

PHP Code

```
<?php
```

```
$selectAdmins = $dbc->query("SELECT uid FROM users WHERE role = 1");
```

```
$selectAdminsArr = $selectAdmins->fetchAll();
```

```
foreach($selectAdminsArr as $row)
```

```
{
```

```
    echo "The user with the UID " . $row['uid'] . " is an admin.";
```

```
}
```

With prepared statements:

PHP Code

```
<?php
```

```
$selectUser = $dbc->prepare("SELECT uid FROM users WHERE name = :name LIMIT 1");
```

```
$selectUser->bindParam(":name", $_POST['name']);
```

```
$selectUser->execute();
```

```
$selectedUid = $selectUser->fetchColumn();
```

The explanation:

Without prepared statements:

When using the query() method, you automatically run the query. The results are saved in \$selectAdmins, but to make it accessible for us, we have to specify how we want the results to be returned.

[fetchAll\(\)](#) returns an array containing all results.

With prepared statements:

We prepare the query before we run it. The ":name" is like a placeholder for the name, we'll tell PHP to 'replace' it with a value with the [bindParam](#) method. That's the simplest way to explain that. We run the query with the [execute](#) method.

[fetchColumn](#) is another way to return a result. However, it will only return one value per row. I personally only use it when I only need a single value returned from the database, like the UID in this example.

Insert into database

The code:

PHP Code

```
<?php
```

```
$insertUserQuery = $dbc->prepare(
```

```
"INSERT INTO users (name, email, password) VALUES (:name, :email, :password)");
```

```
$insertUserQuery->bindParam(":name", $_POST['usrName']);
```

```
$insertUserQuery->bindParam(":email", $_POST['usrEmail']);
```

```
$insertUserQuery->bindParam(":password", $_POST['usrPassword']);
```

```

if($insertUserQuery->execute())
{
    echo "User successfully registered";
}
else
{
    echo "Something went wrong";
}

```

The explanation:

We create a query again and bind the parameters, quite easy as soon as you understood prepared statements, right?

As you can see, the [execute](#) method returns a bool, it returns true when the query executes successfully and false if it fails somewhere.

Update a row

The code:

PHP Code

```
<?php
```

```

$updateUserQuery = $dbc->prepare("UPDATE user SET email = :email WHERE uid = :uid");
$updateUserQuery->bindParam(":email", $_POST['newMail']);
$updateUserQuery->bindParam(":uid", $_SESSION['uid'], PDO::PARAM_INT);
$updateUserQuery->execute();

```

The explanation:

As every query is basically the same (you type the query, use placeholders for your user input and bind it), I won't go too much into detail in this query.

You probably wondered what the "PDO::PARAM_INT" argument does? It basically tells PDO to bind this parameter as an integer. That's useful to use because it won't add quotation marks like it'd do with strings. PDO::PARAM_STRING is the default data type.

[Here](#)'s a list of all PDO constants.

Delete a row

The code:

PHP Code

```
<?php
```

```

$deleteUserQuery = $dbc->prepare("DELETE FROM users WHERE uid = :uid");
$deleteUserQuery->bindParam(":uid", $_SESSION['uid'], PDO::PARAM_INT);
$deleteUserQuery->execute();

```

The explanation:

There isn't much to explain here anymore. You typed the query, binded the parameter and execute it. The user is now deleted.

The end

I hope you enjoyed this tutorial. Switching from mysql_* to PDO is easy, as you already know the MySQL syntax. If used correctly, it should save you lots of time as you don't have to use mysql_real_escape_string or something similar anymore.

If you have any questions, feel free to post them below.

Thanks for reading!
1n9i9c7om

Immortal - March 28, 2014, 8:12 am

very nice thread, Pdo is a really helpful function php has provided for us developers.

I personally use an orm, RedBean <http://redbeanphp.com/>

themastersinner - March 28, 2014, 8:13 am

Very nice tutorial, great for anyone who is thinking of switching to PDO.

w?rloctt - March 28, 2014, 1:14 pm

Very nice tutorial as always! Great job 1n9i9c7om!

VensOnline - April 7, 2014, 5:07 pm

Great tutorial, like always with the empire :)

CarrotMilk - April 7, 2014, 5:57 pm

Just adding: <http://prash.me/php-pdo-and-prepared-statements/>

DerekM - April 7, 2014, 8:09 pm

Thanks for the tutorial post, very useful. 1n9i9c7om ?

(Apr 7, 2014 05:57 PM)CarrotMilk Wrote: [\[CITE\]](#) Just adding:
<http://prash.me/php-pdo-and-prepared-statements/>

Must use prepared statements or else!

Wow haven't seen you around in a while CarrotMilk.

I remember you from the SST section, you stood out for being one of few intelligent members around there.

bababenny - April 19, 2014, 8:16 pm

Great info about the need to use prepared staments thats what I was missing.