PHP – Base de datos

Ciclo: DAW
Módulo: DWES
Curso: 2020-2021
Autor: César Guijarro Rosaleny

Introducción

Una de las partes fundamentales para la mayoría de las aplicaciones de un tamaño considerable, es el uso de las bases de datos (bbdd).

Al añadir bbdd a una aplicación web se amplía el esquema que interviene en la generación de una página web con una nueva capa:

A esta disposición se le denomina **arquitectura web de 3 capas**:

- **Capa primera**: cliente web (navegador)
- **Capa segunda**: servidor web (Apache, Nginx...) con interprete del lenguaje utilizado (PHP, ASP…)
- **Capa tercera**: Sistema gestor de BBDD (MySQL, MariaDB, Oracle…)

El SGBD puede estar instalado en la misma máquina que el servidor web o en otra máquina diferente.

PHP ofrece tres API diferentes para conectarse al SGBD: **mysql**, **mysqli** y **PDO**.

La extensión **mysql** es la más antigua de las 3, y ha sido declarada obsoleta en PHP 5.0, y eliminada en PHP 7.

Está pensada para ser utilizada sólo con versiones de MySQL anteriores a la 4.1.3.

Si se utiliza una versión MySQL 4.1.3 o posterior, **se recomienda encarecidamente usar la extensión *mysqli* en su lugar**.

La extensión **mysqli**, o como a veces se le conoce, la extensión de MySQL mejorada, se desarrolló para aprovechar las nuevas funcionalidades encontradas en los sistemas MySQL con versión 4.1.3 o posterior. La extensión *mysqli* viene incluida en las versiones PHP 5 y posteriores.

Los Objetos de Datos de PHP, o **PDO**, son una capa de abstracción de bases de datos específicas para apliaciones PHP. *PDO* ofrece una API homogénea para las aplicaciones PHP, **independientemente del tipo de servidor de bases de datos con el que vaya a conectar la aplicación**.

A pesar de que *PDO* tiene sus ventajas, tales como una API limpia, sencilla y portable, su mayor inconveniente es que no permite utilizar todas las funcionalidades avanzadas en la última versión del servidor MySQL. Por ejemplo, PDO no permite hacer uso de las declaraciones múltiples de MySQL.

En este tema vamos a ver como conectar PHP con una base de datos y gestionar sus tablas con la extensión *PDO*.

Conexión con la bbdd

Las conexiones se establecen creando instancias de la clase base PDO.

```
<?php
$pdo = new PDO('mysql:host=localhost;dbname=libreria', 'profesor',
'profesor');
```

El constructor de la clase PDO acepta tres parámetros. El primero es el **DSN** (**Data Source Name**) que especifica el tipo de la bbdd (mysql), el host (localhost) y el nombre de la base de datos (libros). Los otros dos parámetros son opcionales y permiten especificar el nombre de usuario y el password.

PDO maneja los errores en forma de excepciones, por lo que la conexión debería ir encerrada siempre en un bloque *try/catch*.

Podemos establecer el modo del manejo de errores con la opción PDO::setAttribute():

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

Existen 3 modos de manejo de errores:

- **PDO::ERRMODE_SILENT**. Es el modo predeterminado. Para comprobar el error debemos emplear PDO::errorCode() y PDO::errorInfo().
- **PDO::ERRMODE_WARNING**. Además de establecer el código de error, PDO emitirá un mensaje E_WARNING. Modo empleado para

depurar o hacer pruebas para ver errores sin interrumpir el flujo de la aplicación.

- **PDO::ERRMODE_EXCEPTION**. Además de establecer el código de error, PDO lanzará una excepción PDOException y establecerá sus propiedades para luego poder reflejar el error y su información. Este modo se emplea en la mayoría de situaciones, ya que permite manejar los errores y a la vez esconder datos que podrían ayudar a alguien a atacar tu aplicación.

También podemos establecer el juego de caracteres de la bbdd mediante la opción PDO::exec();

```
$pdo->exec("set names utf8");
```

Con todo lo anterior, un script típico de conexión a la bbdd quedaría:

```
<?php

try {
	$pdo = new PDO('mysql:host=localhost;dbname=libreria', 'profesor',
'profesor');
	$pdo->exec("set names utf8");
	$pdo->setAttribute(\PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
	echo "Error al conectar con la bbdd";
}
```

Una vez realizada con éxito una conexión a la bbdd, será devuelta una instancia de la clase *PDO* al script. La conexión permanecerá activa durante el tiempo de vida del objeto *PDO*.

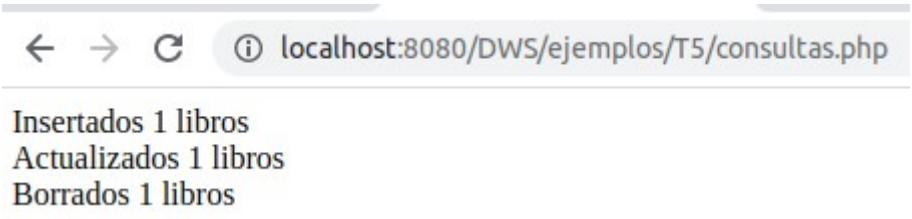
Ejecución de consultas

PDO::exec()

Para ejecutar consultas, podemos usar la función **PDO::exec()**, la cual ejecuta la consulta y **devuelve el número de filas modificadas, insertadas o borradas**, **no devuelve resultados de una sentencia SELECT**.

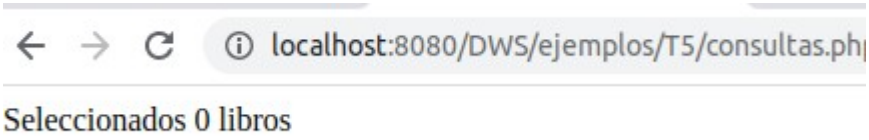
Por ejemplo, crea en *apache_htdocs/DWS/ejemplos/T5* el archivo *conexion.php* con la conexión a la bbdd *libreria* como hemos visto antes. Ahora crea el archivo *consultas.php* con el siguiente contenido:

```php
<?php

require "conexion.php";

$sql = "INSERT INTO libros (titulo, autor, precio) VALUES ('El nombre de la rosa', 'Umberto Eco', 18.95)";
$sql2 = "UPDATE libros SET precio=16.95 WHERE titulo='El nombre de la rosa'";
$sql3 = "DELETE FROM libros WHERE titulo = 'El nombre de la rosa'";

$count = $pdo->exec($sql);
echo "Insertados $count libros<br>";

$count = $pdo->exec($sql2);
echo "Actualizados $count libros<br>";

$count = $pdo->exec($sql3);
echo "Borrados $count libros<br>";
```



Añade ahora otra sentencia SELECT y ejecutala con *PDO::exec()* (puedes comentar el resto de sentencias):

```
$sql4 = "SELECT * FROM libros";

$count = $pdo->exec($sql4);
echo "Seleccionados $count libros<br>";
```



Como ves, aunque la ejecución de la sentencia no da ningún error, no nos devuelve las filas del SELECT.

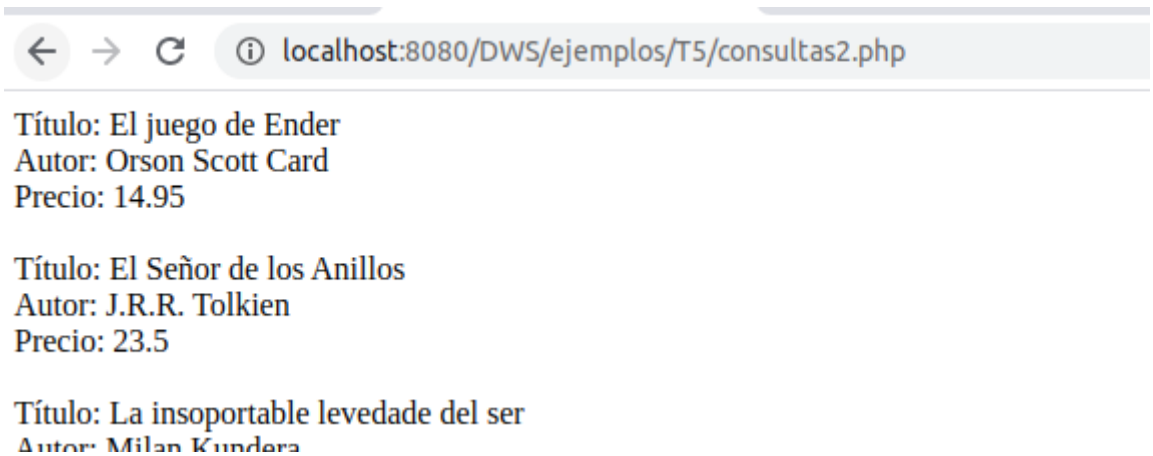
PDO::query()

PDO::query() ejecuta una sentencia SQL, devolviendo un conjunto de resultados como un objeto *PDOStatement*.

Una buena característica de *PDO::query()* es que permite iterar sobre el conjunto de filas devuelto por una ejecución de una sentencia SELECT con éxito.

Por ejemplo, crea el archivo *consultas2.php* en la carpeta anterior:

```php
<?php

require "conexion.php";

$sql = "SELECT * FROM libros";

foreach ($pdo->query($sql) as $row) {
        echo "Título: " . $row['titulo'] . "<br>";
        echo "Autor: " . $row['autor'] . "<br>";
        echo "Precio: " . $row['precio'] . "<br><br>";
}
```



Seguridad en las consultas

SQL Injection

SQL Injection es uno de los ataques más habituales y sencillos de hacer siempre que no se tomen medidas a la hora de tratar los datos. Se trata de aprovechar la falta de seguridad para ejecutar consultas SQL no deseadas.

Vamos a ver un ejemplo de como funcionan este tipo de ataques y como prevenirlos (por lo menos los ataques más básicos, ya que nunca podemos asegurar que nuestra web es segura al 100%).

Lo que haremos será crear un formulario para buscar libros por su título. Crea el archivo *buscar.html* en la carpeta anterior:

```
<!DOCTYPE html>
<html lang="es">
<head>
	<meta charset="UTF-8">
	<title>Buscador libros</title>
</head>
<body>
	<form action="consultas3.php" method="GET">
		<label for="titulo">Título:</label>
		<input type="text" name="titulo">
		<input type="submit" value="OK">
	</form>
</body>
</html>
```

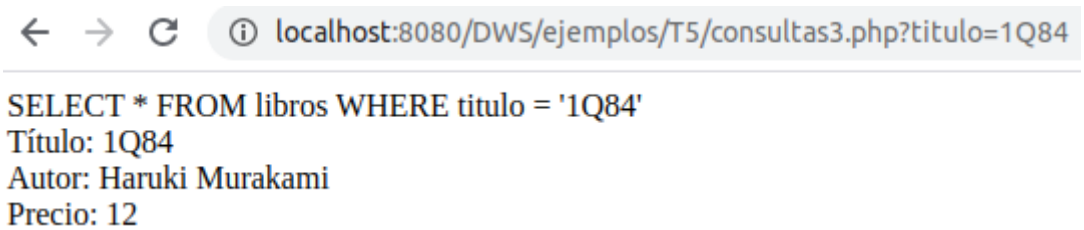
Como ves, es un formulario muy sencillo donde envía el título a buscar a *consultas3.php* con el método GET.

Vamos a crear ahora el archivo que ejecutará la consulta. Crea el archivo *consultas3.php*:

```php
<?php

require "conexion.php";

$titulo = $_GET['titulo'];

$sql = "SELECT * FROM libros WHERE titulo = '$titulo'";

echo $sql . "<br>";

foreach ($pdo->query($sql) as $row) {
        echo "Título: " . $row['titulo'] . "<br>";
        echo "Autor: " . $row['autor'] . "<br>";
        echo "Precio: " . $row['precio'] . "<br><br>";
}
```

El código es muy sencillo. Sólo obtiene el título pasado, ejecuta el SELECT y muestra los datos del libro. Además, mostramos la sentencia SQL para entender mejor el ataque.

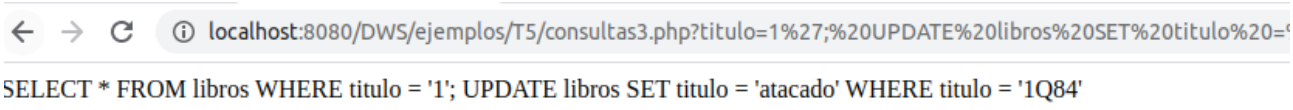
Si lo ejecutamos y buscamos el libro 1Q84 vemos que todo funciona como toca:



Fíjate en la última parte de la URL: consultas3.php?titulo=1Q84. Como hemos pasado los datos a través del método GET, el título está en la URL.

Cambia la URL por consultas3.php?titulo=1'; UPDATE libros SET titulo = 'atacado' WHERE titulo = '1Q84 (igual tienes que cambiar las comillas simples al copiar y pegar, ya que hay veces que cambia el carácter).

Si abres la URL en un navegador, verás que el resultado es:



Lo que hacemos es cerrar la sentencia SELECT con una comilla simple y un punto y coma y a continuación ejecutar una sentencia UPDATE. Si compruebas ahora tu bbdd verás que el título del libro 1Q84 ha cambiado por el de 'atacado' (si no ha cambiado, lo más seguro es que el problema está en las comillas simples a la hora de copiarlas. Lo mejor es ponerlas tú a mano).

Como ves, podemos ejecutar cualquier sentencia SQL que queramos en la bbdd, con lo que podríamos borrar todos los datos de cualquier tabla, crear usuarios con privilegios de administrador…

Para que este tipo de ataque tenga éxito, el atacante tiene que conocer el nombre de las tablas y los campos, lo cual puede parecer difícil, pero existe una variante de este ataque llamado **blind SQL Injection** donde pueden sacar de forma fácil los nombres de todas las tablas y sus campos.

Hay multitud de páginas explicando este tipo de ataques, por ejemplo https://www.elladodelmal.com/2007/06/blind-sql-injection-i-de-en-mysql.html

Vamos a intentar solucionar la vulnerabilidad de nuestra web. Nuestra primera idea puede ser enviar los datos por POST en vez de por GET. En realidad ésto no solucionaría nada, ya que la cadena de ataque se podría poner en el campo del formulario y estaríamos igual.

¿Cuál es entonces la solución?. Como siempre, no confiar en los datos que nos pasa el usuario. Podemos sanitizar y comprobar los datos como vimos en el tema de los ficheros o, como veremos a continuación, utilizar sentencias preparadas, que ya se ocupan de prevenir este tipo de ataques.

Sentencias preparadas

Muchas de las BBDD más maduras admiten el concepto de **sentencias preparadas**. Éstas pueden definirse como un tipo de plantillas compiladas para SQL que las aplicaciones quieren ejecutar, pudiendo ser personalizadas utilizando parámetros variables.

Las sentencias preparadas ofrecen dos grandes beneficios:

- La consulta sólo necesita ser analizada (o preparada) una vez, pero puede ser ejecutada muchas veces con los mismos o diferentes parámetros.
- Los parámetros para las sentencias preparadas no necesitan estar entrecomillados; el controlador se encarga de ésto automáticamente.

La ejecución de sentencias preparadas consiste en tres etapas: la preparación, la vinculación y la ejecución.

Vamos a aprovechar nuestro ejemplo de SQL Injection para explicar como funcionan este tipo de ejecución de sentencias.

Preparación

Usaremos **PDO::prepare()**, la cual prepara una sentencia para su ejecución y devuelve un objeto sentencia. La sentencia SQL puede contener cero o más marcadores de parámetros con nombre (:name) o signos de interrogación (?).

Copia el archivo *consultas3.php* en otro llamado *consultas4.php* y cambia el *action* del formulario al nuevo archivo (*consultas4.php*).

```
<!DOCTYPE html>
<html lang="es">
<head>
	<meta charset="UTF-8">
	<title>Buscador libros</title>
</head>
<body>
	<form action="consultas4.php" method="GET">
		<label for="titulo">Título:</label>
		<input type="text" name="titulo">
		<input type="submit" value="OK">
	</form>
</body>
</html>
```

Ahora vamos a preparar la sentencia SQL en *consultas4.php* usando el método *PDO::prepare()* con parámetros por nombre:

```
$sql = "SELECT * FROM libros WHERE titulo = :titulo";
$sth = $pdo->prepare($sql);
```

Vinculación

Para ligar los parámetros con sus marcadores, usaremos **PDOStatement::bindParam()**. A este método le pasaremos el nombre del parámetro, su valor y el tipo (entero, cadena...).

```
$sth->bindParam(':titulo', $titulo, PDO::PARAM_STR);
```

Por último, para ejecutar la sentencia preparada, usamos **PDOStatement::execute()**.

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

Los parámetros también pueden vincularse en forma de *array* en *execute()*, sin necesidad de utilizar *bindParam()*.

```
$params = [
        ':titulo' => $_POST['titulo']
];
$sth->execute($params);
```

Ejecución

Para recuperar los resultados de una sentencia SELECT, podemos usar los métodos **PDOStatement::fetch()** que obtiene la siguiente fila de un conjunto de resultados, o **PDOStatement::fetchAll()** que devuelve un array que contiene todas las filas del conjunto de resultados.

En ambos métodos podemos pasar el estilo de los datos que nos devolverá la sentencia. Los más comunes son **PDO::FETCH_ASSOC**, que devuelve un array indexado por los nombres de las columnas del conjunto de resultados y **PDO::FETCH_NUM**, que devuelve un array indexado por el número de columna.

```php
$result = $sth->fetchAll(PDO::FETCH_ASSOC);

var_dump($result);
```

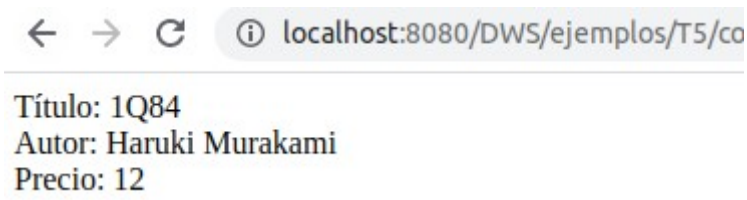
```
/var/www/html/DWS/ejemplos/T5/consultas4.php:20:
array (size=1)
  0 =>
    array (size=4)
      'id' => string '5' (length=1)
      'titulo' => string '1Q84' (length=4)
      'autor' => string 'Haruki Murakami' (length=15)
      'precio' => string '12' (length=2)
```

```php
while ($row = $sth->fetch(PDO::FETCH_NUM)) {
        var_dump($row);
}
```

```
/var/www/html/DWS/ejemplos/T5/consultas4.php:23:
array (size=4)
  0 => string '5' (length=1)
  1 => string '1Q84' (length=4)
  2 => string 'Haruki Murakami' (length=15)
  3 => string '12' (length=2)
```

Nuestro archivo completo podría quedar más o menos así:

```php
<?php

require "conexion.php";

$titulo = $_POST['titulo'];


$sql = "SELECT * FROM libros WHERE titulo = :titulo";
$sth = $pdo->prepare($sql);

$sth->bindParam(':titulo', $titulo, PDO::PARAM_STR);
$sth->execute();

$result = $sth->fetchAll(PDO::FETCH_ASSOC);

foreach ($result as $row) {
        echo "Título: " . $row['titulo'] . "<br>";
        echo "Autor: " . $row['autor'] . "<br>";
        echo "Precio: " . $row['precio'] . "<br><br>";
}
```



Bibliografía

https://www.php.net/manual/es/index.php