

Materia: Producción web

Carrera: Analista de sistemas

Profesor: [Fernando Gonzalo Gaitán](mailto:fernando.g.gaitan@gmail.com)

**Clase PDO**

línea horizontal

# 

# Introducción

Es una clase de PHP que nos permite crear objetos de conexión a la base de datos. Funciona con MySQL, pero también con otros motores de bases de datos.

# 

# Conexión

Podemos conectarnos a una base de datos, por ejemplo MySQL, de la siguiente forma:

$conexion = new PDO('mysql:host=host;dbname=tu\_base\_de\_datos;charset=utf8', 'usuario', 'contrasena');

Tener en cuenta que la conexión puede fallar, y en caso de que esto suceda provocará una excepción. Por tanto, siempre es recomendable hacer la llamada dentro de los bloques try catch:

try{

$conexion = new PDO('mysql:host=host;dbname=tu\_base\_de\_datos;charset=utf8', 'usuario', 'contrasena');

}catch(PDOException $e){

echo "Falló la conexión: " . $e->getMessage();

}

# Consultas

Supongamos que queremos recuperar la lista de usuarios de una tabla:

$consulta = $conexion->prepare('

SELECT id, nombre, email, is\_admin

FROM usuarios

');

$consulta->execute();

Podemos recuperar el resultado de la consulta como un array asociativo:

$usuarios = $consulta->fetchAll();

foreach($usuarios as $usu)

{

echo $usu['nombre'] . ' - ' . $usu['email'];

}

# fetchAll() o fetch()

Para recuperar varios registros podemos usar:

fetchAll()

Y para un registro específico:

fetch()

El primer método, **fetchAll()** devuelve un array (vacío en caso de que no encuentre registros)

Mientras fetch() devuelve el primer registro que encuentre. Un array asociativo o un objeto (PDO::FETCH\_OBJ) y en caso de no encontrar ningún registro false.

# 

# Consultas con filtros

Podemos incluir información externa dentro de un filtro de la siguiente forma:

# $email = 'fernando.gaitan@davinci.edu.ar';

# $consulta = $conexion->prepare("

# SELECT id, nombre, email, is\_admin

# FROM usuarios

# WHERE email = {$email}

# ");

# $consulta->execute();

Pero muchas veces las consultas se construyen en base a información que viene desde un formulario, y tal vez esa información puede llegar a tener código malicioso.

Para eso debemos usar algún método que elimine cadenas de texto que generen SQL injection. Por ejemplo:

$email = 'fernando.gaitan@davinci.edu.ar';

$consulta = $conexion->prepare('

SELECT id, nombre, email, is\_admin

FROM usuarios

WHERE email = :email

');

$consulta->bindValue(':email', $email);

$consulta->execute();

$usuario = $consulta->fetch(PDO::FETCH\_ASSOC);

if($usuario){

echo "El usuario se llama: {$usuario->nombre}";

}else{

echo 'Usuario no encontrado';

}

# 

# bindValue() vs bindParam()

Una alternativa a bindValue() es el método bindParam() La diferencia entre ambos es que el primero envía información por un valor, mientras que el segundo lo hace por referencia:

Supongamos que queremos modificar un registro:

$nombre = 'Juan';

$id = 1;

$consulta = $conexion->prepare('

UPDATE usuarios

SET

nombre = :nombre

WHERE id = :id

');

$consulta->bindParam(':nombre', $nombre);

$consulta->bindParam(':id', $id);

$consulta->execute();

Podríamos modificar bindParam() por bindValue() y el resultado sería el mismo.

Sin embargo bindParam() al pasar datos por referencia, se fijará en el valor actual de la variable cuando se llame al método execute():

$nombre = 'Juan';

$consulta->bindParam(':nombre', $nombre);

$nombre = 'Raúl';

$consulta->execute();

En este caso el valor que se va a enviar es ‘Raúl’, porque es el valor actual de la variable $nombre, es decir el valor por referencia.

Si en cambio hiciéramos esto con bindValue():

$nombre = 'Juan';

$consulta->bindValue(':nombre', $nombre);

$nombre = 'Raúl';

$consulta->execute();

El valor será: ‘Juan’. Porque aunque la variable $nombre cambió su valor, bindValue() pasa información por valor y no por referencia.

# PDO con Singleton

Un singleton es una clase que puede tener una sola instancia.

Dentro de un proyecto podríamos tener varias consultas a una base de datos, y al crear una instancia de PDO es recomendable reutilizar esa instancia para cada consulta:

class Cnx {

private static $db = null;

private function \_\_construct()

{

//

}

public static function getInstance()

{

if(is\_null (self::$db) ) {

self::$db = new PDO('mysql:host=host;dbname=tu\_base\_de\_datos;charset=utf8', 'usuario', 'contrasena');

}

return self::$db;

}

public static function close()

{

self::$db = null;

}

}

Dentro de una propiedad estática vamos a guardar una sóla instancia:

private static $db = null;

Al intentar recuperar la única instancia verificamos si esa propiedad es nula, es decir, que todavía no se creó una instancia:

if(is\_null (self::$db) )

De ser así guardamos dentro de esa propiedad una instancia de PDO:

self::$db = new PDO('mysql:host=host;dbname=tu\_base\_de\_datos;charset=utf8', 'usuario', 'contrasena');

Finalmente recuperamos el valor único de esa instancia:

return self::$db;

Además de eso tendremos un método que cierre la conexión, cuando el script no requiera de más conexiones, igualando nuevamente esa instancia a null:

public static function close()

{

self::$db = null;

}

Pero lo más importante, el constructor será privado para evitar nuevas instancias:

private function \_\_construct()

{

//

}