Tutorial de PDO

PDO es una extensión de PHP que permite acceder a diferentes sistemas de bases de datos utilizando las mismas funciones

Contenido modificable

Si ves errores o quieres modificar/añadir contenidos, puedes crear un pull request. Gracias

Indice de contenido

1. Introducción

PDO significa PHP Data Objects, Objetos de Datos de PHP, una extensión para acceder a bases de datos. PDO permite acceder a diferentes sistemas de bases de datos con un controlador específico (MySQL, SQLite, Oracle...) mediante el cual se conecta. Independientemente del sistema utilizado, se emplearán siempre los mismos métodos, lo que hace que cambiar de uno a otro resulte más sencillo.

Para ver los controladores (*drivers*) disponibles en tu servidor, puedes emplear el método *getAvailableDrivers()*:

```
print_r(PDO::getAvailableDrivers());
```

El sistema PDO se fundamenta en 3 clases: PDO, PDOStatement y PDOException. La clase PDO se encarga de mantener la conexión a la base de datos y otro tipo de conexiones específicas como transacciones, además de crear instancias de la clase PDOStatement. Es ésta clase, PDOStatement, la que maneja las sentencias SQL y devuelve los resultados. La clase PDOException se utiliza para manejar los errores.

2. Conectar a una base de datos con PDO

El primer argumento de la clase PDO es el DSN, Data Source Name, en el cual se han de especificar el tipo de base de datos (*mysql*), el host (*localhost*) y el nombre de la base de datos (se puede especificar también el puerto). Diferentes sistemas de bases de datos tienen distintos métodos para conectarse. La mayoría se conectan de forma parecida a como se conecta a MySQL:

```
try {
    $dsn = "mysql:host=localhost;dbname=$dbname";
    $dbh = new PDO($dsn, $user, $password);
} catch (PDOException $e){
    echo $e->getMessage();
}
```

DBH significa Database Handle, y es el nombre de variable que se suele utilizar para el objeto PDO.

Para cerrar una conexión:

```
$dbh = null;
```

Puedes ver más información acerca de como conectarse a determinados sistemas de bases de datos en php.net.

3. Excepciones y opciones con PDO

PDO maneja los errores en forma de excepciones, por lo que la conexión siempre ha de ir encerrada en un bloque try/catch. Se puede (y se debe) especificar el modo de error estableciendo el atributo *error mode*:

```
$dbh->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_SILENT);
$dbh->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_WARNING);
$dbh->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

No importa el modo de error, si existe un fallo en la conexión siempre producirá una excepción, por eso siempre se conecta con try/catch.

- PDO::ERRMODE_SILENT. Es el modo de error por defecto. Si se deja así habrá que comprobar los errores de forma parecida a como se hace con mysqli. Se tendrían que emplear PDO::errorCode() y PDO::errorInfo() o su versión en PDOStatement PDOStatement::errorCode() y PDOStatement::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.

El modo de error se puede aplicar con el método PDO::setAttribute o mediante un array de opciones al instanciar PDO:

```
// Con un array de opciones
try {
    $dsn = "mysql:host=localhost;dbname=$dbname";
    $options = array(
      PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    );
    $dbh = new PDO($dsn, $user, $password);
} catch (PDOException $e){
    echo $e->getMessage();
}
// Con un el método PDO::setAttribute
try {
    $dsn = "mysql:host=localhost;dbname=$dbname";
    $dbh = new PDO($dsn, $user, $password);
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e){
    echo $e->getMessage();
}
```

Existen más opciones aparte de el modo de error ATTR_ERRMODE, algunas de ellas son:

- ATTR_CASE. Fuerza a los nombres de las columnas a mayúsculas o minúsculas (CASE_LOWER, CASE_UPPER).
- ATTR_TIMEOUT. Especifica el tiempo de espera en segundos.
- ATTR_STRINGIFY_FETCHES. Convierte los valores numéricos en cadenas.

4. Registrar datos con PDO

La clase PDOStatement es la que trata las sentencias SQL. Una instancia de PDOStatement se crea cuando se llama a *PDO->prepare()*, y con ese objeto creado se llama a métodos como *bindParam()* para pasar valores o *execute()* para ejecutar sentencias. PDO facilita el uso de sentencias preparadas en PHP, que mejoran el rendimiento y la seguridad de la aplicación. Cuando se obtienen, insertan o actualizan datos, el esquema es: PREPARE -> [BIND] -> EXECUTE. Se pueden indicar los parámetros en la sentencia con un interrogante "?" o mediante un nombre específico.

- Utilizando interrogantes para los valores

```
// Prepare
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES (?, ?)");
// Bind
$nombre = "Peter";
$ciudad = "Madrid";
$stmt->bindParam(1, $nombre);
$stmt->bindParam(2, $ciudad);
// Excecute
$stmt->execute();
// Bind
$nombre = "Martha";
$ciudad = "Cáceres";
$stmt->bindParam(1, $nombre);
$stmt->bindParam(2, $ciudad);
// Execute
$stmt->execute();
```

- Utilizando variables para los valores

```
// Prepare
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES (:nombre, :ciudad)");
// Bind
$nombre = "Charles";
$ciudad = "Valladolid";
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':ciudad', $ciudad);
// Excecute
$stmt->execute();
// Bind
$nombre = "Anne";
$ciudad = "Lugo";
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':ciudad', $ciudad);
// Execute
$stmt->execute();
```

También existe un método lazy, que es pasando los valores mediante un array (siempre array, aunque sólo haya un valor) al método *execute()*:

```
// Prepare:
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES (:nombre, :ciudad)");
$nombre = "Luis";
$ciudad = "Barcelona";
// Bind y execute:
if($stmt->execute(array(':nombre'=>$nombre, ':ciudad'=>$ciudad))) {
    echo "Se ha creado el nuevo registro!";
}
```

Es el método *execute()* el que realmente envía los datos a la base de datos. Si no se llama a execute no se obtendrán los resultados sino un error.

Una característica importante cuando se utilizan variables para pasar los valores es que se pueden insertar objetos directamente en la base de datos, suponiendo que las propiedades coinciden con los nombres de las variables:

```
class Clientes
{
    public $nombre;
    public $ciudad;
    public function __construct($nombre, $ciudad){
        $this->nombre = $nombre;
        $this->ciudad = $ciudad;
    }
    // ....Código de la clase....
}
$cliente = new Clientes("Jennifer", "Málaga");
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES (:nombre, :ciudad)");
if($stmt->execute((array) $cliente)){
    echo "Se ha creado un nuevo registro!";
};
```

5. Diferencia entre bindParam() y bindValue()

Existen dos métodos para enlazar valores: *bindParam()* y *bindValue()*:

- Con *bindParam()* la variable es enlazada como una referencia y sólo será evaluada cuando se llame a *execute()*:

```
// Prepare:
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
$nombre = "Morgan";
// Bind
$stmt->bindParam(':nombre', $nombre); // Se enlaza a la variable $nombre
// Si ahora cambiamos el valor de $nombre:
$nombre = "John";
$stmt->execute(); // Se insertará el cliente con el nombre John
```

- Con *bindValue()* se enlaza el valor de la variable y permanece hasta execute():

```
// Prepare:
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
$nombre = "Morgan";
// Bind
$stmt->bindValue(':nombre', $nombre); // Se enlaza al valor Morgan
// Si ahora cambiamos el valor de $nombre:
$nombre = "John";
$stmt->execute(); // Se insertará el cliente con el nombre Morgan
```

En la práctica *bindValue()* se suele usar cuando se tienen que insertar datos sólo una vez, y *bindParam()* cuando se tienen que pasar datos múltiples (desde un array por ejemplo).

Ambas funciones aceptan un tercer parámetro, que define el tipo de dato que se espera. Los *data types* más utilizados son: PDO::PARAM_BOOL (*booleano*), PDO::PARAM_NULL (*null*), PDO::PARAM_INT (*integer*) y PDO::PARAM_STR (*string*).

6. Consultar datos con PDO

La consulta de datos se realiza mediante PDOStatement::fetch, que obtiene la siguiente fila de un conjunto de resultados. Antes de llamar a fetch (o durante) hay que especificar como se quieren devolver los datos:

- PDO::FETCH_ASSOC: devuelve un array indexado cuyos keys son el nombre de las columnas.
- PDO::FETCH_NUM: devuelve un array indexado cuyos keys son números.
- PDO::FETCH_BOTH: valor por defecto. Devuelve un array indexado cuyos keys son tanto el nombre de las columnas como números.
- PDO::FETCH_BOUND: asigna los valores de las columnas a las variables establecidas con el método PDOStatement::bindColumn.
- PDO::FETCH_CLASS: asigna los valores de las columnas a propiedades de una clase. Creará las propiedades si éstas no existen.
- PDO::FETCH_INTO: actualiza una instancia existente de una clase.
- PDO::FETCH_OBJ: devuelve un objeto anónimo con nombres de propiedades que corresponden a las columnas.
- PDO::FETCH_LAZY: combina PDO::FETCH_BOTH y PDO::FETCH_OBJ, creando los nombres de las propiedades del objeto tal como se accedieron.

Los más utilizados son FETCH_ASSOC, FETCH_OBJ, FETCH_BOUND y FETCH_CLASS. Vamos a poner un ejemplo de los dos primeros:

```
// FETCH_ASSOC
$stmt = $dbh->prepare("SELECT * FROM Clientes");
// Especificamos el fetch mode antes de llamar a fetch()
$stmt->setFetchMode(PDO::FETCH_ASSOC);
// Ejecutamos
$stmt->execute();
// Mostramos los resultados
while ($row = $stmt->fetch()){
    echo "Nombre: {$row["nombre"]} <br>";
    echo "Ciudad: {$row["ciudad"]} <br><br>";
}
// FETCH_OBJ
$stmt = $dbh->prepare("SELECT * FROM Clientes");
// Ejecutamos
$stmt->execute();
// Ahora vamos a indicar el fetch mode cuando llamamos a fetch:
while($row = $stmt->fetch(PDO::FETCH_OBJ)){
    echo "Nombre: " . $row->nombre . "<br>";
    echo "Ciudad: " . $row->ciudad . "<br>";
}
```

Con FETCH_BOUND debemos emplear el método *bindColumn()*:

```
// Preparamos
$stmt = $dbh->prepare("SELECT nombre, ciudad FROM Clientes");
// Ejecutamos
$stmt->execute();
// bindColumn
$stmt->bindColumn(1, $nombre);
$stmt->bindColumn('ciudad', $ciudad);
while ($row = $stmt->fetch(PDO::FETCH_BOUND)) {
    $cliente = $nombre . ": " . $ciudad;
    echo $cliente . "<br>";
}
```

El estilo de devolver los datos FETCH_CLASS es algo más complejo: devuelve los datos directamente a una clase. Las propiedades del objeto se establecen ANTES de llamar al constructor. Si hay nombres de columnas que no tienen una propiedad creada para cada una, se crean como *public*. Si los datos necesitan una transformación antes de que salgan de la base de datos, se puede hacer automáticamente cada vez que se crea un objeto:

```
class Clientes
{
    public $nombre;
    public $ciudad;
    public $otros;
    public function __construct($otros = ''){
        $this->nombre = strtoupper($this->nombre);
        $this->ciudad = mb_substr($this->ciudad, 0, 3);
        $this->otros = $otros;
    }
    // ....Código de la clase....
}
$stmt = $dbh->prepare("SELECT * FROM Clientes");
$stmt->setFetchMode(PDO::FETCH_CLASS, 'Clientes');
$stmt->execute();
while ($objeto = $stmt->fetch()){
    echo $objeto->nombre . " -> ";
    echo $objeto->ciudad . "<br>";
}
```

Con lo anterior hemos podido modificar cómo queríamos mostrar nombre y ciudad de cada registro. A nombre lo hemos puesto en mayúsculas y de ciudad sólo hemos mostrado las tres primeras letras.

Si lo que quieres es llamar al constructor ANTES de que se asignen los datos, se hace lo siguiente:

```
$stmt->setFetchMode(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE, 'Clientes';
```

Si en el ejemplo anterior añadimos PDO::FETCH_PROPS_LATE, el nombre y la ciudad se mostrarán como aparecen en la base de datos.

También se pueden pasar argumentos al constructor cuando se quieren devolver datos en objetos con PDO:

```
$stmt->setFetchMode(PDO::FETCH_CLASS, 'Clientes', array('masdatos');
```

O incluso datos diferentes para cada objeto:

```
$i = 0;
while ($row = $stmt->fetch(PDO::FETCH_CLASS, 'Clientes', array($i))){
    // Código para hacer algo
    $i++;
}
```

Finalmente, para la consulta de datos también se puede emplear directamente *PDOStatement::fetchAll()*, que devuelve un array con todas las filas devueltas por la base de datos con las que poder iterar. También acepta estilos de devolución:

```
// fetchAll() con PDO::FETCH_ASSOC
$stmt = $dbh->prepare("SELECT * FROM Clientes");
$stmt->execute();
$clientes = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach($clientes as $cliente){
    echo $cliente['nombre'] . "<br>";
}
// fetchAll() con PDO::FETCH_OBJ
$stmt = $dbh->prepare("SELECT * FROM Clientes");
$stmt->execute();
$clientes = $stmt->fetchAll(PDO::FETCH_OBJ);
foreach ($clientes as $cliente){
    echo $cliente->nombre . "<br>";
}
```

7. Diferencia entre query() y prepare()/execute()

En los ejemplos anteriores para las sentencias en PDO, no se ha introducido el método *query()*. Este método ejecuta la sentencia directamente y necesita que se escapen los datos adecuadamente para evitar ataques SQL Injection y otros problemas.

execute() ejecuta una sentencia preparada lo que permite enlazar parámetros y evitar tener que escapar los parámetros. *execute()* también tiene mejor rendimiento si se repite una sentencia múltiples veces, ya que se compila en el servidor de bases de datos sólo una vez.

Ya hemos visto como funcionan las sentencias preparadas con *prepare()* y *execute()*, vamos a ver un ejemplo con query():

```
$stmt = $dbh->query("SELECT * FROM Clientes");
$clientes = $stmt->fetchAll(PDO::FETCH_OBJ);
foreach ($clientes as $cliente){
    echo $cliente->nombre . "<br>";
}
```

Se cambia *prepare* por *query* y se quita el *execute*.

8. Otras funciones de utilidad

Existen otras funciones en PDO que pueden ser de utilidad:

- PDO::exec(). Ejecuta una sentencia SQL y devuelve el número de filas afectadas. Devuelve el número de filas modificadas o borradas, no devuelve resultados de una secuencia SELECT:

```
    // Si lo siguiente devuelve 1, es que se ha eliminado correctamente:
    echo $dbh->exec("DELETE FROM Clientes WHERE nombre='Luis'");
    // No devuelve el número de filas con SELECT, devuelve 0
    echo $dbh->exec("SELECT * FROM Clientes");
```

- PDO::lastInsertId(). Este método devuelve el id autoincrementado del último registro en esa conexión:

```
    $stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
    $nombre = "Angelina";
    $stmt->bindValue(':nombre', $nombre);
    $stmt->execute();
    echo $dbh->lastInsertId();
```

- PDOStatement::fetchColumn(). Devuelve una única columna de la siguiente fila de un conjunto de resultados. La columna se indica con un integer, empezando desde cero. Si no se proporciona valor, obtiene la primera columna.

```
    $stmt = $dbh->prepare("SELECT * FROM Clientes");
    $stmt->execute();
    while ($row = $stmt->fetchColumn(1)){
        echo "Ciudad: $row <br>";
    }
```

- PDOStatement::rowCount(). Devuelve el número de filas afectadas por la última sentencia SQL:

```
    $stmt = $dbh->prepare("SELECT * FROM Clientes");
    $stmt->execute();
    echo $stmt->rowCount();
```

9. Transacciones con PDO

Cuando tenemos que ejecutar varias sentencias de vez, como INSERT, es preferible utilizar transacciones ya que agrupa todas las acciones y permite revertirlas todas en caso de que haya algún error.

Una transacción en PDO comienza con el método *PDO::beginTransaction()*. Este método desactiva cualquier otro commit o sentencia SQL o consultas que aún no son commited hasta que la transacción es committed con *PDO::commit()*. Cuando este método es llamado, todas las acciones que estuvieran pendientes se activan y la conexión a la base de datos vuelve de nuevo a su estado por defecto que es auto-commit. Con *PDO::rollback()* se revierten los cambios realizados durante la transacción.

```
try {
    $dbh->beginTransaction();
    $dbh->query("INSERT INTO Clientes (nombre, ciudad) VALUES ('Leila Birdsall', 'Madrid')");
    $dbh->query("INSERT INTO Clientes (nombre, ciudad) VALUES ('Brice Osterberg', 'Teruel')");
    $dbh->query("INSERT INTO Clientes (nombre, ciudad) VALUES ('Latrisha Wagar', 'Valencia')");
    $dbh->query("INSERT INTO Clientes (nombre, ciudad) VALUES ('Hui Riojas', 'Madrid')");
    $dbh->query("INSERT INTO Clientes (nombre, ciudad) VALUES ('Frank Scarpa', 'Barcelona')");
    $dbh->commit();
    echo "Se han introducido los nuevos clientes";
} catch (Exception $e){
    echo "Ha habido algún error";
    $dbh->rollback();
}
```