PHP Data Object - PDO

PDO es una interface de acceso a datos que nos permite, mediante varios drivers, conectarnos a diferentes bases de datos. Esta librería escrita en C viene activada por defecto desde PHP 5.1 por lo cual la podrás utilizar en la mayoría de los servidores que actualmente soportan PHP5.

PDO proporciona una capa de abstracción *acceso a datos*, que significa que**, independientemente de la base de datos que está utilizando, se utiliza las mismas funciones para realizar consultas y obtener datos**.

PDO *no* proporciona una abstracción *base de datos*.

El trabajar con PDO tiene algunas ventajas, entre otras:

- Tienes drivers de **12 bases de datos diferentes** [enumerarlas utilizando `print_r (PDO :: getAvailableDrivers ());`]
- Enfoque de programación orientada a objetos
- **Evitar SQL Inyección**
- Mapeo objetos

La conexión

Principalmente lo que debemos hacer es realizar una simple conexión a cualquier base de datos, una de las ventajas y características que tiene PDO es qué **tiene el mismo string de conexión**, -por así decirlo-, **para cualquier motor de base de datos que está soporta**. Aquí veremos algunas de estás conexiones en MySQL y PostgreSQL, para ver la diferencia y la simplicidad.

Con **MySQL** y **PostgreSQL** nos conectábamos de la siguiente forma:

```
# MySQL
mysql_connect("$host", "$usuario", "$contraseña");
mysql_select_db("$baseDatos");

# PostgreSQL
pg_connect("host = $host dbname = $baseDatos user = $usuario password =
$contraseña");
```

Ahora con PDO lo hacemos general de la siguiente forma, (sólo cambiaría el driver de conexión)

Notamos que utiliza la misma estructura para la conexión en cualquiera de los 2 motores de base de datos.

Para todos los ejemplos utilizaré MySQL, pero también podría utilizar cualquier otra de las bases de datos soportadas adaptando un poco el código que sigue:

```
1 $conexion = new PDO('driver:host=servidor;dbname=bd', user, pass);
```

Y el ejemplo práctico:

```
1 $conexion = new PDO('mysql:host=localhost;dbname=pruebas', 'root' , '' );
```

 # Por ejemplo en PostgreSQL

```
1 $conexion = new PDO(pgsql:host=localhost;dbname=pruebas', 'root', '');
```

Ahora en $conexion tenemos una instancia de PDO_MySQL.

Este objeto **representa una conexión con una base de datos**, y nos permite ejecutar consultas y obtener sus resultados.

Por defecto PDO viene configurado para NO mostrar ningún error. Es decir que para saber si se ha producido un error, tendríamos que estar comprobando los **métodos errorCode()** y **errorInfo()**. Para facilitarnos la tarea vamos a habilitar las excepciones. De esta forma cada vez que ocurra un error saltará una excepción que capturaremos y podremos tratar correctamente para mostrarle un mensaje al usuario. Para realizar esta tarea utilizaremos la **función setAttribute()** de la siguiente forma:

```
$con = new PDO('mysql:host=localhost;dbname=nombreBaseDatos', 'user','pass',
                $opcionesPDO));
$con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Los posibles valores que se le podría asignar a *ATTR_ERRMODE* son:

- **PDO::ERRMODE_SILENT** es el valor por defecto y como he mencionado antes no lanza ningún tipo de error ni excepción, es tarea del programador comprobar si ha ocurrido algún error después de cada operación con la base de datos.
- **PDO::ERRMODE_WARNING** genera un error E_WARNING de PHP si ocurre algún error. Este error es el mismo que se muestra usando la API de mysql mostrando por pantalla una descripción del error que ha ocurrido.
- **PDO::ERRMODE_EXCEPTION** es el que acabamos de explicar que genera y lanza una excepción si ocurre algún tipo de error.

Como acabamos de hacer que se lancen excepciones cuando se produzca algún error, el paso que tenemos que dar a continuación es capturarlas por si se producen, para ello realizamos lo siguiente:

```
try
{
  $con = new PDO('mysql:host=localhost;dbname=nombreBaseDatos', 'user', 'pass',
$opcionesPDO));
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch(PDOException $e)
{
  echo 'Error conectando con la base de datos: ' . $e->getMessage();
}
```

Consultas PDO

Ahora que sabemos como conectarnos a la base de datos, vamos a crear una sentencia para poder recuperar datos

Hay dos formas de realizar consultas mediante PDO, aunque en ambos casos se trabajará con código SQL.

Existen **2 tipos de consultas en PDO**

- De ejecución directa. Utilizando la llamada a **query()**
- De declaraciones preparadas (**Prepared Statements**) utilizando el **prepare()**

Aunque tenemos disponibles las dos llamadas es mucho más seguro utilizar la llamada prepare() ya que esta se encarga de escapar por nosotros los parámetros y nos asegura que no sufriremos problemas de SQL Injection.

La función **query()** se *suele utilizar cuando la sentencia que vamos a ejecutar no contiene parámetros que ha enviado el usuario*.

Veamos un ejemplo utilizando la función **query():**

```
try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $datos = $con->query('SELECT nombre FROM personal');
  foreach($datos as $row)
    echo $row[0] . '<br/>';
}
catch(PDOException $e)
{
  echo 'Error conectando con la base de datos: ' . $e->getMessage();
}
```

Si a pesar de las advertencias aun quieres ejecutar sentencias con **query()** pasándole parámetros de usuarios, la forma correcta de hacerlo sería escapando esos parámetros con la función **quote()** como se muestra a continuación:

PDO::query() → ejecuta una sentencia SQL en una única llamada a función, devolviendo el conjunto de resultados (si los hay) que devuelve la sentencia **como un objeto PDOStatement**.

Valores devueltos→ devuelve un objeto PDOStatement, o **FALSE** en caso de error.

```
$ape = 'Hernandez';

try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $datos = $con->query('SELECT nombre FROM personal WHERE apellidos like ' .
                       $con->quote($ape));
  foreach($datos as $row)
    echo $row[0] . '<br/>';
}
catch(PDOException $e)
{

  echo 'Error conectando con la base de datos: ' . $e->getMessage();

}
```

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

PDO::prepare() →

Prepara una sentencia para su ejecución.

Si el servidor de la base de datos prepara con éxito la sentencia , **PDO::prepare()** devuelve un objeto PDOStatement. Si no es posible, **PDO::prepare()** devuelve **FALSE** o emite una excepción PDOException (dependiendo del manejo de errores)

La forma de utilizar la función **prepare()** **que es la más recomendada es la siguiente**:

```
try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $con->prepare('SELECT nombre FROM personal');
  $stmt->execute();

  while( $datos = $stmt->fetch() )
    echo $datos[0] . '<br/>';
}
catch(PDOException $e)
{
  echo 'Error: ' . $e->getMessage();
}
```

Como se ve, es realmente simple ejecutar consultas. Simplemente tenemos que indicarle a la función **prepare()** la sentencia sql que queremos ejecutar. Esta función nos devolverá un **PDPStatement** sobre el cual ejecutaremos la función **fetch()** para mostrar su valor.

ANEXO:

La clase PDOStatement

Representa una sentencia preparada y, después de la ejecución de la instrucción, unos conjuntos de resultados asociados.

Funciones más importantes:

- **fetch()** → obtiene la siguiente fila del conjunto de resultados.

Lo que hace el método **fetch()** es devolvernos **un array asociativo** con los datos que le pedimos, en caso de encontrar un resultado ,pero en caso de no encontrar nada nos devolverá **false**.

- **fetchAll()** → devuelve un array que contiene las filas del conjunto de resultados.

Este método a diferencia de **fetch()** **NO** nos devolverá un array asociativo sino que nos devolverá un array de arrays asociativos(Array asociativo bidimensional)

- **fetchColumn_** ([int **$ column_number**]) → Devuelve una única columna de la siguiente fila de un conjunto de resultados o FALSO si no hay más filas.

$column_number → Número de la columna que desea recuperar de la fila indexada, 0 si no se proporciona ningún valor

Para sentencias del tipo **Select Count(*)**

- **rowCount ()** → devuelve el nº de filas afectadas por la última sentencia INSERT, UPDATE, DELETE. Para sentencias del tipo SELECT, en algunas BD NO está garantizado y por tanto no debería confiarse en el. Se puede sustituir por fetchColumn

Si necesitamos pasarle valores a la sentencia sql, utilizaríamos los parámetros. Los parámetros los indicamos en la misma sentencia sql y los podemos escribir de **dos formas distintas**:

1. Mediante el signo **?** **o** mediante un nombre de una variable precedido por el símbolo **:nombreParam**.
2. Nos permite una identificación más fácil de los parámetros, y es utilizando la función **bindParam**

Pero cualquiera de las 2 formas es correcta.

1ª Forma:

```
$ape = 'Hernandez';

try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $con->prepare('SELECT nombre FROM personal WHERE apellidos like :apellidos');
  $stmt->execute(array(':apellidos' => $ape ));

  while( $datos = $stmt->fetch() )
    echo $datos[0] . '<br />';
//ten en cuenta que al salir del while el puntero esta situado al final del la consulta
 //por tanto si quieres leer dos veces el resultado de la consulta puedes hacer los siguiente
//que es mucho más facil utilizando el foreach
$rows = $stmt->fetchAll();
foreach ($rows as $r) {
    echo $r[0];
}

foreach ($rows as $r) {
    echo $r[0];
}

}
catch(PDOException $e)
{
  echo 'Error: ' . $e->getMessage();
}
```

Otra manera de hacer lo mismo, pero utilizando **?** :

```
//Preparamos la consulta marcando donde irán los parametros con ?
$consulta = $db->prepare('SELECT * FROM items WHERE id_item = ? OR id_item =
?');

//Ejecutamos la consulta incluyendo los parámetros en el mismo orden en el que
deben incluirse
$consulta->execute(array(2, 4));
```

El ejemplo anterior generará una consulta de la siguiente manera:

```
SELECT * FROM items WHERE id_item = '2' OR id_item = '4'
```

2ª Forma. Otra manera de hacer lo mismo:

```
$id = 6;
 //Esta vez utilizamos un nombre-clave para cada parámetro
$consulta = $db->prepare('SELECT * FROM items WHERE id_item = :id');
 //Con dicho nombre-clave, agregamos el valor del parámetro
$consulta->bindParam(':id', $id, PDO::PARAM_INT);
//Y ejecutamos la consulta
$consulta->execute();
```

El ejemplo anterior generará una consulta de la siguiente manera:

```
SELECT * FROM items WHERE id_item = 6
```

A la función **bindParam()** le pasamos el nombre del parámetro, su valor y finalmente el tipo que es. Los tipos de parámetros que le podemos pasar los podemos ver en las constantes predefinidas de PDO y son:

- PDO::PARAM_BOOL
- PDO::PARAM_NULL
- PDO::PARAM_INT
- PDO::PARAM_STR
- PDO::PARAM_LOB

Al igual que las sentencias SELECT, podemos utilizar las funciones **query()** y **prepare()** para ejecutar INSERT, UPDATE, DELETE. La forma de hacerlo es igual que lo hemos estado viendo hasta ahora.

Altas, Bajas y Modificaciones

El mecanismo sigue siendo el mismo que en las consultas anteriores, preparar la consulta, agregar los parámetros y ejecutar.

Alta

```
$nom = 'Jose';
$ape = 'Hernandez';

try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $con->prepare('INSERT INTO personal (nombre, apellidos) VALUES
(:nombre, :apellidos)');
  $stmt->execute( array( ':nombre'    => $nom,
                         ':apellidos' => $ape));
```

```
$rows= $stmt->rowCount();
if( $rows == 1 )
    echo 'Inserción correcta';
}
catch(PDOException $e)
{
  echo 'Error: ' . $e->getMessage();
}
```

Modificación

```
$nom = 'Jose';
$ape = 'Hernandez';

try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $con->prepare('UPDATE personal SET apellidos = :apellidos WHERE
nombre = :nombre');
 $stmt->execute( array( ':nombre'    => $nom,
                        ':apellidos' => $ape));
 $rows= $stmt->rowCount();

  if( $rows > 0 )
    echo 'Actualización correcta';
  else
    if( $rows==0 )
      echo 'No se encuentra ese registro en la BD';

}
catch(PDOException $e)
{
  echo 'Error: ' . $e->getMessage();
}
```

Bajas

```
$ape = 'Hernandez';

try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $con->prepare('DELETE FROM personal WHERE apellidos = :apellidos');


  $stmt->execute( array( ':apellidos' => $ape));
  $rows= $stmt->rowCount();

  if( $rows > 0 )
    echo 'Borrado correcto';
}
catch(PDOException $e)
{
  echo 'Error: ' . $e->getMessage();
}
```

MAPEO DE OBJETOS EN PHP

Para acabar con esta entrada sobre PDO vamos a ver otra de las funcionalidades que nos aporta y que puede ser muy útil. **PDO nos permite realizar consultas y mapear los resultados en objetos de nuestro modelo**. Para ello primero tenemos que crearnos una clase con nuestro modelo de datos.

```
class Usuario
{
  private $nombre;
  private $apellidos;

  public function nombreApellidos()
  {
    return $this->nombre . ' ' . $this->apellidos;
  }
}
```

Existen 2 Formas de Mapear Objetos:
Hay que tener en cuenta que para que funcione correctamente, **el nombre de los atributos en nuestra clase tiene que ser iguales que los que tienen las columnas en nuestra tabla de la BD**. **Y además el constructor de la clase debe de estar VACIO (sin parámetros)** Con esto claro vamos a realizar la consulta.

setFetchMode() (Mapeo Real, es el más recomendable)

La novedad que podemos ver en este script es la llamada al método **setFetchMode()** pasándole como primer argumento la constante **PDO::FETCH_CLASS** que le indica que haga un mapeado en la clase que le indicamos como segundo argumento, en este caso la clase Usuario que hemos creado anteriormente. Después al recorrer los elementos con **fetch()** los resultados en vez de un vector los obtendremos en el objeto indicado.

```
include 'Usuario.php';
try
{
  $con = new PDO('mysql:host=localhost;dbname=personal', 'user', 'pass');
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 $stmt= $con->prepare('SELECT nombre, apellidos FROM personal');
  $stmt->execute();

  $stmt->setFetchMode(PDO::FETCH_CLASS, 'Usuario');

  foreach($stmt as $usuario)
    echo $usuario->getNombreApellidos() . '<br>';

}
catch(PDOException $e)
{
  echo 'Error: ' . $e->getMessage();}
```

fetchObject (mapeo Ficticio)

Devuelve una fila de resultados de una consulta como si fuese un objeto.

Ejemplo:

```
$consulta = $con->prepare('SELECT nombre, apellidos FROM personal');
 $consulta->execute();
  /* obtener el array de objetos */
    while ($obj = $consulta->fetchObject())
                {
        printf ("%s %s\n", $obj->nombre,$obj->apellidos);
    }
```

Pero **OJO** se accede a ellos por el nombre del campo de la tabla, no por el nombre del atributo del objeto, por eso digo que es un mapeo ficticio.

El patrón Singleton con PHP

Algunas veces nos interesa tener controlada la creación de objetos por motivos como por ejemplo el ahorro de memoria. Una forma de hacer esto es utilizando el patrón de diseño "Singleton". Vamos a estudiar un ejemplo en PHP para ver su utilidad, funcionamiento e implementación.

Un claro ejemplo de la utilidad de este patrón sería una clase para trabajar con bases de datos. Si cada vez que necesitemos conectarnos tenemos que crear una instancia del objeto, estaríamos aumentando el consumo de memoria de nuestra aplicación de una manera innecesaria.

Para conseguir nuestro objetivo lo primero que tenemos que hacer es **no permitir crear instancias del objeto,** para esto basta con "ocultar" el constructor declarándolo como "**private**". Lo siguiente que necesitamos es crear un **método estático que nos devuelva una instancia de ese objeto**, si existe una ya creada la devolvemos y si no creamos una nueva.

Veamos un ejemplo de implementación:

```
class Database {
        static private $instance = null;

        private function __contruct() {}

        public static function getInstance() {
                if (self::$instance == null) {
                        self::$instance = new Database();
                }
                return self::$instance; }
        public function connect($dsn) {
                ...
        }
        public function query($sql) {
```

```
            ...
        }
        public function executeQuery() {
            ...
        }

        public function getResult() {
            ...
        }

        public function disconnect() {
            ...
        }
}
```

Éste sería un prototipo de una clase para trabajar con una base de datos. Como podemos ver, ya no podremos hacer "$db = new Database();", por lo que nos aseguramos el ahorro de memoria. En su lugar para obtener el objeto que deseamos tendremos que hacer:

```
...
$db = Database::getInstance();
$db->connect($dsn);
...
```

Siempre que llamemos al método estático "getInstance()" se comprobará que no exista ya una instancia de este objeto, si no existe crearemos una, la guardaremos como una propiedad estática de la clase y la devolveremos, muy simple.

Otra forma de ejecutar el código anterior sin tener que inicializar ninguna variable:

```
...
Database::getInstance()->connect($dsn);
...
```

Como se puede apreciar, utilizando este patrón adecuadamente, a parte de ahorrar memoria, tendríamos un código más limpio y libre de variables globales, que pueden ser modificadas por accidente en otros procesos

Ejemplo utilizando PDO:

```
<?php

//normalmente estas constantes van en un fichero aparte
//ejemplo webconfgi.php
define('DNS','mysql:dbname=juan;host=localhost');
define('USERNAME','root');
define('PASSWORD','juan');
```

```
class Database
{
  private static $dns       = DNS;
  private static $username  = USERNAME;
  private static $password  = PASSWORD;
  private static $instance;

  private function __construct() { }

  public static function getInstance()
  {
    if (!isset(self::$instance))
    {
    // self se utiliza para referenciar métodos staticos o atributos estaticos
    //this no puede referenciar a metodos estaticos
    //para acceder a metodos staticos fuera de la clase ::
      self::$instance = new PDO(self::$dns, self::$username,
self::$password);
      self::$instance->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    }
    return self::$instance;
  }


 /**
  * Impide que la clase sea clonada
  *
  */
  public function __clone()
  {
    trigger_error('La clonación de este objeto no está permitida',
E_USER_ERROR);
  }
}

?>
```

```
//Obtiene el conector de base de datos llamando al método estático de
la clase de base de datos [patrón singleton]
$con = database::getInstance();


//Operacion de Insercion
$sql = "insert into `user` (`username`, `password`, `fullname`,
`email`) values (:username, :password, :fullname, :email)";//nombre de
los parametros
$consulta = $con -> prepare($sql);

$result = $consulta -> execute(array(":username" => $username,
":password" => $password, ":fullname" => $fullname, ":email" =>
$email));    //valor de los parametros

if($result)
    echo "Insercion Correcta";
else
    echo "Error en la Insercion";

//Operacion de Modificacion ahora con ?
```

```php
$sql = "update `user` set `username` = ?,  `password` = ?, `fullname`
= ?, `email` = ? where `user_id` = ?";    //preparo el statements

$sentencia = $con -> prepare($sql);

$result = $sentencia -> execute(array($username, $password, $fullname,
$email, $user_id));   //asigno los valores al statement

if($result)
    echo "Se ha efectuado la Modificación Correctamente";
else
    echo "Modificación Fallida";

//operación de borrado
$sql = "delete from `user` where `user_id` = ?";   //preparo el
statements

$sentencia = $con -> prepare($sql);

$result = $sentencia -> execute(array($user_id));   //asigno los
valores

if($result)
    echo "Eliminado Correctamente";
else
    echo "Operacion de Borrado Fallida";

//consulta select
$sql = "select * from `user` where `user_id` = ?";   //preparo el
statements

$consulta = $con -> prepare($sql);

$consulta -> execute(array($user_id));   //prepared statements value

$consulta -> setFetchMode(PDO::FETCH_ASSOC); // [PDO::FETCH_NUM for
integer key value]

$result = $consulta -> fetchAll();
print_r($result)
```