PHP Data Objects (PDO)

La extensión [PDO (PHP Data Objects)](http://www.php.net/manual/es/book.pdo.php) permite acceder a distintas bases de datos utilizando las mismas funciones, lo que facilita la portabilidad. En PHP 5 existen drivers para acceder a las bases de datos más populares (MySQL, Oracle, MS SQL Server, PostgreSQL, SQLite, Firebird, DB2, Informix, etc).

La clase PDO no necesita ser incluida, la misma está disponible desde la versión 5 de PHP y existe en memoria.

Utilizando PDO podemos solventar muchas dificultades que surgen al utilizar la API de mysql. Un par de ventajas que se obtienen con PDO es que el proceso de escapado de los parámetros es sumamente sencillo y sin la necesidad de estar atentos a utilizar en todos los casos funciones para este cometido como mysql\_real\_escape\_string(). Otra ventaja es que PDO es una API flexible y nos permite trabajar con cualquier tipo de base de datos y no estar restringidos a utilizar MySql.

**Usuarios de Windows**

1. PDO y la mayoría de controladores vienen en PHP como extensiones compartidas; para activarlas, simplemente se debe editar el fichero php.ini
2. extension=php\_pdo.dll

**Nota**:

Este paso no es necesario para PHP 5.3 o superior, pues ya no se requiere la DLL para utilizar PDO.

1. Después, elija los demás ficheros DLL de bases de datos específicas, y utilice [dl()](http://php.net/manual/es/function.dl.php) para cargarlos en tiempo de ejecución, o actívelos en php.ini por debajo de la línea php\_pdo.dll. Por ejemplo:
2. extension=php\_pdo.dll
3. extension=php\_pdo\_firebird.dll
4. extension=php\_pdo\_informix.dll
5. extension=php\_pdo\_mssql.dll
6. extension=php\_pdo\_mysql.dll
7. extension=php\_pdo\_oci.dll
8. extension=php\_pdo\_oci8.dll
9. extension=php\_pdo\_odbc.dll
10. extension=php\_pdo\_pgsql.dll
11. extension=php\_pdo\_sqlite.dll

Estas DLL deberían existir en el directorio [extension\_dir](http://php.net/manual/es/ini.core.php#ini.extension-dir) del sistema.

**Nota**:

Recuerde que después de realizar cambios en el fichero php.ini, será necesario reiniciar PHP para que las nuevas directivas de configuración surtan efecto.

Lo primero que tenemos que hacer es realizar la conexión a la base de datos. La conexión la realizaremos con el constructor de la clase de la siguiente forma:

$con = new PDO('mysql:host=localhost;dbname=nombreBaseDatos', 'user', 'pass');

PDO puede utilizar las excepciones para gestionar los errores, lo que significa que cualquier cosa que hagamos con PDO podríamos encapsularla en un bloque [try/catch](http://php.net/manual/es/language.exceptions.php) para gestionar si produce algún error.

Para ejecutar una consulta SQL utilizando PDO, debes diferenciar aquellas sentencias SQL que no devuelven como resultado un conjunto de datos, de aquellas otras que sí lo devuelven.

En el caso de las consultas de acción, como **INSERT**, **DELETE** o **UPDATE**, el método exec devuelve el número de registros afectados.

$registros = $con->exec('DELETE FROM stock WHERE unidades=0');

print "<p>Se han borrado $registros registros.</p>";

<?php

$con= **new** PDO("mysql:host=localhost;dbname=dwes", "root", "");

$uni=3;

$registros = $con->exec("DELETE FROM stock WHERE unidades=**$uni**");

**print** "<p>Se han borrado **$registros** registros.</p>";

?>

Si la consulta genera un conjunto de datos, como es el caso de SELECT, debes utilizar el método **query.**

$resultado = $con->query("SELECT producto, unidades FROM stock");

Ejercicio1:

<?php

**try**

{

$con = **new** PDO('mysql:host=localhost;dbname=dwes', 'root', '');

$resultado = $con->query("SELECT producto, unidades FROM stock");

**while** ($registro = $resultado->fetch()) {

**echo** "Producto ".$registro['producto'].": ".$registro['unidades']."<br />";

}

//cambia la repetitiva por foreach

}

**catch**(PDOException $e){

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

}

?>

Por defecto, el método fetch genera y devuelve a partir de cada registro un array con claves numéricas y asociativas. Para cambiar su comportamiento, admite un parámetro opcional que puede tomar uno de los siguientes valores:

* **PDO::FETCH\_ASSOC**. Devuelve solo un array asociativo.
* **PDO::FETCH\_NUM**. Devuelve solo un array con claves numéricas.
* **PDO::FETCH\_BOTH**. Devuelve un array con claves numéricas y asociativas. Es el comportamiento por defecto.
* **PDO::FETCH\_OBJ**. Devuelve un objeto cuyas propiedades se corresponden con los campos del registro.

$resultado = $con->query("SELECT producto, unidades FROM stock");

while ($registro = $resultado->fetch(PDO::FETCH\_OBJ)) {

echo "Producto ".$registro->producto.": ".$registro->unidades."<br />";

}

Por defecto PDO trabaja en modo "autocommit", esto es, confirma de forma automática cada sentencia que ejecuta el servidor. Para trabajar con transacciones, PDO incorpora tres métodos:

* **beginTransaction**. Deshabilita el modo "autocommit" y comienza una nueva transacción, que finalizará cuando ejecutes uno de los dos métodos siguientes.
* **commit**. Confirma la transacción actual.
* **rollback**. Revierte los cambios llevados a cabo en la transacción actual.

Una vez ejecutado un commit o un rollback, se volverá al modo de confirmación automática.

$ok = true;

$con->beginTransaction();

if($con->exec('DELETE …') == 0) $ok = false;

if($con->exec('UPDATE …') == 0) $ok = false;

…

if ($ok) $con->commit(); // Si todo fue bien confirma los cambios

else $con->rollback(); // y si no, los revierte

Ten en cuenta que no todos los motores soportan transacciones. Tal es el caso, del motor MyISAM de MySQL. En este caso concreto, PDO ejecutará el método beginTransaction sin errores, pero naturalmente no será capaz de revertir los cambios si fuera necesario ejecutar un rollback.

Consultas preparadas.

Al igual que con MySQLi, también utilizando PDO podemos preparar consultas parametrizadas en el servidor para ejecutarlas de forma repetida. El procedimiento es similar e incluso los métodos a ejecutar tienen prácticamente los mismos nombres.

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

 $datos = $con->query('SELECT nombre FROM personal');

Para preparar la consulta en el servidor MySQL, deberás utilizar el método **prepare** de la clase PDO. Este método devuelve un objeto de la clase **PDOStatement**. Los parámetros se pueden marcar utilizando signos de interrogación como en el caso anterior.

$consulta = $con->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');

Los parámetros se pueden marcar utilizando signos de interrogación como en el caso anterior.O también utilizando parámetros con nombre, precediéndolos por el símbolo de dos puntos.

$consulta = $con->prepare('INSERT INTO familia (cod, nombre) VALUES (:cod, :nombre)');

Antes de ejecutar la consulta hay que asignar un valor a los parámetros utilizando el método bindParam de la clase PDOStatement.. Si utilizas signos de interrogación para marcar los parámetros:

$cod\_producto = "TABLET";

$nombre\_producto = "Tablet PC";

$consulta->bindParam(1, $cod\_producto);

$consulta->bindParam(2, $nombre\_producto);

Si utilizas parámetros con nombre, debes indicar ese nombre en la llamada a bindParam.

$consulta->bindParam(":cod", $cod\_producto);

$consulta->bindParam(":nombre", $nombre\_producto);

Tal y como sucedía con la extensión MySQLi, cuando uses bindParam para asignar los parámetros de una consulta preparada, deberás usar siempre variables como en el ejemplo anterior.

Una vez preparada la consulta y enlazados los parámetros con sus valores, se ejecuta la consulta utilizando el método execute.

$consulta->execute();

Alternativamente, es posible asignar los valores de los parámetros en el momento de ejecutar la consulta, utilizando un array (asociativo o con claves numéricas dependiendo de la forma en que hayas indicado los parámetros) en la llamada a execute.

$parametros = array(":cod" => "TABLET", ":nombre" => "Tablet PC");

$consulta->execute($parametros);

Ejemplo

<?php

$con= **new** PDO("mysql:host=localhost;dbname=dwes", "root", "");

$alta1 = $con->prepare('INSERT INTO tienda VALUES (?,?,?)');

$alta2 = $con->prepare('INSERT INTO tienda VALUES (:c, :nom, :t)');

$alta3 = $con->prepare('INSERT INTO tienda VALUES (?,?,?)');

$alta4 = $con->prepare('INSERT INTO tienda VALUES (:co,:nom,:t)');

//alta1

$codigo = 4;

$nombre = "alta 1";

$telefono = "99";

$alta1->bindParam(1, $codigo);

$alta1->bindParam(2, $nombre);

$alta1->bindParam(3, $telefono);

$alta1->execute();

//alta2

$codigo = 5;

$nombre = "alta 2";

$telefono = "88";

$alta2->bindParam(":c", $codigo);

$alta2->bindParam(":nom", $nombre);

$alta2->bindParam(":t", $telefono);

$alta2->execute();

//alta3

$insertar = **array**(6,"alta 3","77");

$alta3->execute($insertar);

//alta 4

$insertar = **array**(":co"=>7,":nom"=>"alta 4",":t"=>"66");

$alta4->execute($insertar);

/\* $rows = $alta4->execute( array( ":co"=>7,

":nom"=>"alta 4",

":t"=>"66"));

if( $rows == 1 )

echo 'Inserción correcta';\*/

?>

Ejemplo:

Obtener el nombre del estudiante cuyo teléfono es 456, utilizando la función prepare()

<?php

$telefono = '456';

**try**

{

$con = **new** PDO('mysql:host=localhost;dbname=estudiantes', 'root', '');

$stmt = $con->prepare('SELECT nombre FROM estudiantes WHERE telefono like :telefono');

$stmt->execute(**array**(':telefono' => $telefono ));

**while**( $datos = $stmt->fetch() )

**echo** $datos[0] . '<br/>';

}

**catch**(PDOException $e)

{

**echo** 'Error: ' . $e->getMessage();

}

?>

Ejercicio:

 La tabla stock de la base de datos dwes, la tienda 1 (CENTRAL) tiene 2 unidades del producto de código 3DSNG y la tienda 3 (SUCURSAL2) ninguno. Suponiendo que los datos son esos (no hace falta que los compruebes en el código), utiliza una transacción para mover una unidad de ese producto de la tienda 1 a la tienda 3.

Realizar el ejercicio con PDO