Índice

Sumario

Base de Datos

PHP y bases de datos

Cuando una aplicación web necesita conservar información de forma permanente para recuperarla posteriormente, suele ser conveniente recurrir a un sistema gestor de bases de datos (SGBD).

Normalmente los SGBD son aplicaciones externas que se instalan y administran de forma separada. PHP permite utilizar la mayoría de los SGBD más conocidos, libres (MySQL, PostgreSQL, MariaDB, Firebird, etc) o comerciales (Oracle, MS SQL Server, etc).

Un caso particular es SQLite, que no es una aplicación externa sino una biblioteca en C que implementa un motor de bases de datos SQL. Eso quiere decir que PHP puede gestionar bases de datos directamente, sin necesidad de recurrir a SGBD externos.

Aunque PHP siempre ha permitido utilizar numerosos SGBD, la forma de hacerlo ha ido variando con el tiempo.

Acceso mediante bibliotecas específicas

En las primeras versiones de PHP, la única manera de acceder a un SGBD era a través de una biblioteca específica, que contenía las funciones necesarias. A esas bibliotecas se les suelen llamar también extensiones. En algunos casos esas extensiones se incluían en las distribuciones oficiales de PHP y para poder utilizarlas era suficiente con incluir la directiva correspondiente en el archivo de configuración php.ini. En otros casos esas extensiones no

estaban incluidas en las distribuciones oficiales de PHP, pero estaban incluidas en PEAR o PECL y para poder utilizarlas era necesario instalarlas por separado.

Con el paso del tiempo, este enfoque fue mostrando sus limitaciones. El principal inconveniente de este enfoque es que cada extensión está estrechamente vinculada a cada SGBD y si se quiere cambiar de SGBD es necesario reescribir completamente la aplicación (los nombres de cada función son diferentes, el orden de los argumentos distinto, las funcionalidades disponibles son diferentes, etc.). Además en algunos casos, las extensiones no están mantenidas adecuadamente (bugs de seguridad, etc.).

Actualmente, PHP se sigue distribuyendo con muchas de estas extensiones, pero en su lugar se recomienda utilizar la biblioteca PDO.

PHP y MySQL

PHP ha tenido tres extensiones para acceder a la base de datos MySQL:

- Extensión mysql: Esta extensión fue la primera extensión para MySQL incluida en PHP. En diciembre de 2012 se decidió eliminar la extensión. Desde PHP 5.5 (publicado en junio de 2013) la extensión estuvo desaconsejada y su uso generaba

  un aviso E_DEPRECATED. Desde PHP 7.0 (publicada en diciembre de 2015), ya no está incluida en PHP.
- Extensión mysqli: Esta extensión se incluyó por primera vez en PHP 5.0 (publicado en julio de 2004). Las principales ventajas de esta extensión son la posibilidad de utilizar una sintaxis orientada a objetos, sentencias preparadas, transacciones y otras.
- Extensión PDO: esta extensión, que permite trabajar con diferentes bases de datos, incluye un controlador para MySQL desde que fue incluida en PHP 5.1 (publicado en noviembre de 2005).

Aunque la extensión mysqli se sigue incluyendo en PHP, en su lugar se recomienda utilizar para acceder a MySQL la extensión PDO.

En esta unidad vamos a ver cómo acceder desde PHP a bases de datos MySQL utilizando PDO. Para el seguimiento de esta unidad se supone que conoces el lenguaje SQL utilizado en la gestión de bases de datos relacionales.

Además, para el acceso a las funcionalidades de la extensión vamos a utilizar objetos. Aunque más adelante veremos todas las características que nos ofrece PHP para crear programas orientados a objetos, debemos suponer también en este punto un cierto conocimiento de programación orientada a objetos. Básicamente, se debe saber cómo crear y utilizar objetos.

En PHP se utiliza la palabra new para crear un nuevo objeto instanciando una clase, y para acceder a los miembros de un objeto, se debe utilizar el operador flecha → .

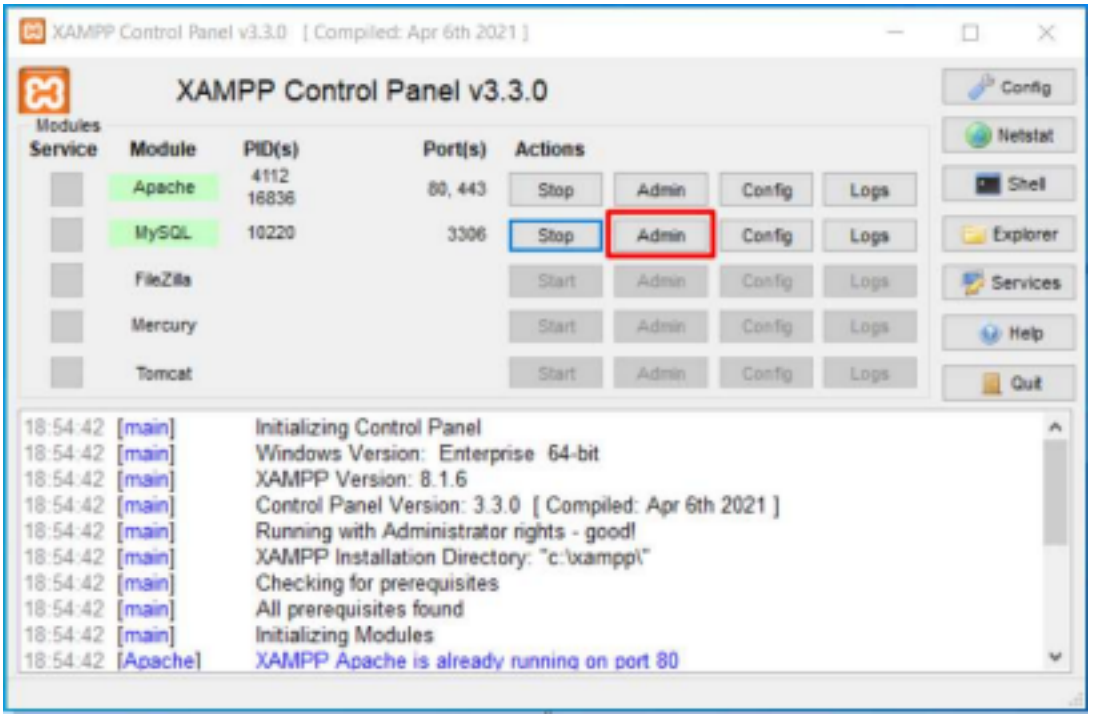
```php
<?php
 class ejemplo
 {
  function hacer_algo()
  {
   echo "<p>Haciendo algo.</p>";
  }
 }

 $valor = new ejemplo;
 $valor->hacer_algo();
?>
```

```
<p>Haciendo algo.</p>
```


XAMPP

Cuando se instaló el entorno de XAMPP en un principio para trabajar con PHP, se instaló también MySQL o MariaDB.

Para crear una base de datos, XAMPP instala también una aplicación codificada en PHP que nos permite interactuar con MySQL o MariaDB.
Este programa se llama PHPMyAdmin. Para iniciar el PHPMyAdmin debemos presionar el botón "Admin" de MySQL en el panel de control de XAMPP:



La interfaz de PHPMyAdmin es una aplicación que se ejecuta en la web y que ya hemos visto anteriormente.

Utilización de bases de datos MySQL en PHP

A la hora de establecer una conexión con MySQL desde PHP lo más habitual es elegir entre MySQLi (extensión nativa) y PDO.

Con cualquiera de ambas extensiones, se puede realizar acciones sobre las bases de datos como:

- Establecer conexiones.
- Ejecutar sentencias SQL.
- Obtener los registros afectados o devueltos por una sentencia SQL.
- Emplear transacciones.
- Ejecutar procedimientos almacenados.
- Gestionar los errores que se produzcan durante la conexión o en el establecimiento de la misma.

PDO y MySQLi (y también la antigua extensión MySQL) utilizan un driver de bajo nivel para comunicarse con el servidor MySQL. Hasta hace poco el único driver disponible para

realizar esta función era libmysql, que no estaba optimizado para ser utilizado desde PHP. A partir de la versión 5.3, PHP viene preparado para utilizar también un nuevo driver mejorado para realizar esta función, el Driver Nativo de MySQL, mysqlnd.

Extensión PDO

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, la extensión PDO permite acceder a distintas bases de datos utilizando las mismas funciones, lo que facilita la portabilidad. La extensión PDO no evalúa la corrección de las consultas SQL, aunque sí implementa algunas medidas de seguridad mediante las consultas preparadas.

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

```php
<?php

print_r(PDO::getAvailableDrivers());

?>
```

Conexión con la base de datos

Para conectar con la base de datos hay que crear una instancia de la clase PDO, que se utiliza en todas las consultas posteriores. En cada página PHP que incluya consultas a la base de datos es necesario conectar primero con la base de datos.

Si no se puede establecer la conexión con la base de datos, puede deberse a que la base de datos no esté funcionando, a que los datos de usuario no sean correctos, o a que no esté activada la extensión pdo.

Extensión PDO_MySQL

Para poder acceder a MySQL mediante PDO, debe estar activada la extensión php_pdo_mysql en el archivo de configuración php.ini.

```
extension=pdo_mysql ; Valor recomendado en este curso
```

XAMPP 8.1 tiene activada la extensión PDO MySQL, por lo que no es necesario modificarla, aunque conviene comprobar su valor.

Conexión con MySQL

Para establecer una conexión con una base de datos utilizando PDO, debes instanciar un objeto de la clase PDO. En el caso de MySQL, para crear el objeto PDO se necesita proporcionar el nombre del servidor, el nombre de usuario y la contraseña.

```php
<?php
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";

    $conexion = new PDO($mysql, $user, $password);
?>
```

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), el nombre de la base de datos (se puede especificar también el puerto) y el juego de caracteres.

Diferentes sistemas de bases de datos tienen distintos métodos para conectarse. La mayoría se conectan de forma parecida a como se conecta a MySQL.

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

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";

    $conexion = new PDO($mysql, $user, $password);

    echo "<p>Conectado a la BBDD</p>";
  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
  }
?>
```

Si la conexión falla, se ejecutará el catch para mostrar el error correspondiente y si la conexión es exitosa se muestra el mensaje de que estamos conectados a la base de datos. Además de usar la estructura de un try/catch para conectarnos a la base de datos se suele utilizar un cuarto parámetro que proporciona la clase PDO. Este parámetro es un array de opciones. La opción que se suele utilizar es:

- PDO::ERRMODE_EXCEPTION Con esta opción 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 una aplicación.

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);

    $conexion = new PDO($mysql, $user, $password, $opciones);

    echo "<p>Conectado a la BBDD</p>";
  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
```

```
  }
?>
```

Una vez establecida la conexión, puedes utilizar el método getAttribute para obtener información del estado de la conexión y setAttribute para modificar algunos parámetros que afectan a la misma. Por ejemplo, para obtener la versión del servidor puedes hacer:


```
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);

    $conexion = new PDO($mysql, $user, $password, $opciones);

    $version = $conexion->getAttribute(PDO::ATTR_SERVER_VERSION);
    // Mostramos mensaje de la versión
    echo "Versión: $version";

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
  }
?>
```

Operador de Resolución de Ámbito

El Operador de Resolución de Ámbito o, en términos simples, el doble dos-puntos, permite acceder a elementos estáticos y constantes de una clase.

```php
<?php

class MiClase {
        const VALOR_CONST = 'Un valor constante';

          public static function dobleDosPuntos() {
                print "<p>Doble dos puntos</p>";
        }
}

$clase = 'MiClase';
echo "<p>".$clase::VALOR_CONST ."</p>";
echo "<p>". MiClase::VALOR_CONST ."</p>";
echo "<p>". MiClase::dobleDosPuntos()."</p>";

?>
```


Desconexión con la base de datos

Para terminar la conexión con la base de datos, desconectar, hay que destruir el objeto PDO. Si no se destruye el objeto PDO, PHP lo destruye al terminar la página.

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);

    $conexion = new PDO($mysql, $user, $password, $opciones); echo

    "<p>Conectado a la BBDD</p>";

    // Cerrar la conexión
    $conexion = null;

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
  }
?>
```

Ejecución de consultas

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.


```php
<?php
  try {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION); }
  catch (PDOException $e) {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }
// PDO::exec() devuelve el número de filas modificadas o borradas  por la sentencia
SQL. Si no hay filas afectadas, devuelve 0.

// consultas de acción, como INSERT, DELETE o UPDATE

$registros = $conexion->exec('DELETE FROM mensajes WHERE
mensaje="Texto del Mensaje"');

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

$conexion = null;
?>
```

NOTA: No se recomienda usar exec para las consultas de acción. En su lugar, por motivos

de seguridad, se aconseja el uso de consultas preparadas.

Si la consulta genera un conjunto de datos, como es el caso de SELECT, debes utilizar el método query, que devuelve un objeto de la clase PDOStatement.

```php
<?php
  try {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

    $resultado = $conexion->query('select * FROM mensajes');
    $conexion = null;

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
  }
?>
```


Obtención y utilización de conjuntos de resultados

En PDO tienes varias posibilidades para tratar con el conjunto de resultados devuelto por el método query. La más utilizada es el método fetch de la clase PDOStatement. Este método devuelve un registro del conjunto de resultados, o false si ya no quedan registros por recorrer.

```php
<?php
try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $resultado = $conexion->query('select * FROM mensajes'); while
  ($registro = $resultado->fetch()) {
    echo "<p>Nombre: ".$registro['nombre']."</p>";
    // O también $registro[1];
  }

  $conexion = null;
?>
```

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_BOUND: asigna los valores de las columnas a las variables establecidas con el método PDOStatement::bindColumn.
- PDO::FETCH_OBJ. Devuelve un objeto cuyas propiedades se corresponden con los campos del registro.

PDO::FETCH_ASSOC

```php
<?php
  try
{
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $resultado = $conexion->query('select * FROM mensajes'); while ($registro
  = $resultado->fetch(PDO::FETCH_ASSOC)) { echo "<p>Nombre:
  ".$registro['nombre']."</p>";
  }

  $conexion = null;
?>
```

PDO::FETCH_NUM

```php
<?php
  try {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $resultado = $conexion->query('select * FROM mensajes'); while
  ($registro = $resultado->fetch(PDO::FETCH_NUM)) { echo "<p>Nombre:
  ".$registro[1]."</p>";
  }

  $conexion = null;
?>
```

PDO::FETCH_BOTH

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $resultado = $conexion->query('select * FROM mensajes'); while
  ($registro = $resultado->fetch(PDO::FETCH_BOTH)) { echo
  "<p>Nombre: ".$registro['nombre']."</p>";
  // O también $registro[1];
}

$conexion = null;
?>
```

PDO::FETCH_BOUND

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $resultado = $conexion->query('select * FROM mensajes');

  $resultado->bindColumn(1, $id);
  $resultado->bindColumn(2, $nombre);
```

```
$resultado->bindColumn(3, $email);
$resultado->bindColumn(4, $mensaje);
```



```
while ($registro = $resultado->fetch(PDO::FETCH_BOUND)) { echo
  "<p>Id: $id</p>";
  echo "<p>Nombre: $nombre</p>";
  echo "<p>Email: $email</p>";
  echo "<p>Mensaje: $mensaje</p>";
}

$conexion = null;
?>
```

PDO::FETCH_OBJ

```
<?php
 try
 {
  $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
  "dwes_tarde";
  $password = "dwes_2223";
  $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
  $conexion = new PDO($mysql, $user, $password);

 }
 catch (PDOException $e)
 {
  // Mostramos mensaje en caso de error
  echo "<p>" .$e->getMessage()."</p>";
  exit();
 }

 $resultado = $conexion->query('select * FROM mensajes'); while
 ($registro = $resultado->fetch(PDO::FETCH_OBJ)) { echo "<p>Nombre:
 ".$registro->nombre."</p>";
 }

$conexion = null;
?>
```

Es posible recoger todos los valores de una vez con el método fetchAll. En este caso se tiene un array de arrays, pero este método se desaconseja.

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
```



```
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $resultado = $conexion->query('select * FROM mensajes'); $registros =
  $resultado->fetchAll();

  // para mostrar todos los datos
  foreach ($registros as $registro) {
      print_r ($registro);
      foreach ($registro as $clave => $valor) {
          echo "<p> $clave: $valor </p>";
      }
  }

  $conexion = null;
?>
```

Una ventaja del query es que permite iterar sobre el conjunto de filas devueltos por una ejecución de una sentencia SELECT con éxito sin necesidad de recurrir al fetch.

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  foreach ($conexion->query('select * FROM mensajes') as $row) { echo
      "<p>Nombre: ".$row['nombre'] ."</p>";
      echo "<p>Email: ".$row['email'] ."</p>";
  }

  $conexion = null;
?>
```

Consultas preparadas

A la hora de ejecutar consultas de acción o de obtención de registros que requieran algún parámetro, por motivos de seguridad y para controlar los datos introducidos se deben

preparar las mismas. Con PDO podemos preparar consultas parametrizadas en el servidor para ejecutarlas de forma repetida.

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 o nombres.

Usando signos de interrogación:

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

$consulta = $conexion->prepare("insert into mensajes (nombre,  email, mensaje)
values (?,?,?)");

  $nombre = 'Nombre';
  $email = 'emanil@email.com';
  $mensaje = 'Texto mensaje PDO';

  $consulta->bindParam(1, $nombre);
  $consulta->bindParam(2, $email);
  $consulta->bindParam(3, $mensaje);

  $consulta->execute();

  $conexion = null;
?>
```


Utilizando parámetros con nombre, precediéndolos por el símbolo de dos puntos:

```php
<?php
try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

$consulta = $conexion->prepare("insert into mensajes (nombre,  email, mensaje)
values (:nombre,:email,:mensaje)");

  $nombre = 'Nombre';
  $email = 'email@email.com';
  $mensaje = 'Texto mensaje PDO';

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

$consulta->execute();

$conexion = null;
?>
```

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

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

$consulta = $conexion->prepare("insert into mensajes (nombre,  email, mensaje)
values (:nombre, :email, :mensaje)");
$nombre = 'Nombre';
$email = 'email@email.com';
$mensaje = 'Texto mensaje PDO';

$consulta->execute(array(':nombre' => $nombre, ':email' => $email, ':mensaje' =>
$mensaje));

$conexion = null;
?>
```

Es el método execute() el que realmente envía los datos a la base de datos 18

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():

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

$consulta = $conexion->prepare("insert into mensajes (nombre,  email, mensaje)
values (:nombre, :email, :mensaje)");

$nombre = 'Nombre';
$email = 'emanil@email.com';
$mensaje = 'Texto mensaje PDO';

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

// Si ahora cambiamos el valor de $nombre
$nombre = 'Otro Nombre';

$consulta->execute(); // Se insertará el nombre 'Otro Nombre'

$conexion = null;
?>
```


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

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

$consulta = $conexion->prepare("insert into mensajes (nombre,  email, mensaje)
values (:nombre, :email, :mensaje)");

$nombre = 'Nombre';
$email = 'emanil@email.com';
$mensaje = 'Texto mensaje PDO';

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

// Si ahora cambiamos el valor de $nombre
$nombre = 'Otro Nombre';

$consulta->execute(); // Se insertará el nombre 'Nombre'

$conexion = null;
?>
```

Transacciones con PDO

Cuando tenemos que ejecutar varias sentencias de vez, cómo varios 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 se llama, 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.

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  try {
    $conexion->beginTransaction();
    $conexion->query("INSERT INTO mensajes (nombre, email, mensaje) VALUES
('Nombre1', 'email1@email.com', 'Texto Mensaje 1')"); $conexion->query("INSERT
INTO mensajes (nombre, email, mensaje)  VALUES ('Nombre2', 'email2@email.com',
'Texto Mensaje 2')"); $conexion->query("INSERT INTO mensajes (nombre, email,
mensaje)  VALUES ('Nombre3', 'email3@email.com', 'Texto Mensaje 3')");
$conexion->query("INSERT INTO mensajes (nombre, email, mensaje)  VALUES
('Nombre4', 'email4@email.com', 'Texto Mensaje 4')");

    $conexion->commit();
    print "<p>Se han introducido los nuevos mensajes</p>"; } catch
  (Exception $e)
  {
    echo "<p>Ha habido algún error</p>";
    $conexion->rollback();
  }

  $conexion = null;
?>
```


Otras funciones de utilidad

Existen otras funciones en PDO que pueden ser de utilidad:

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

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

$consulta = $conexion->prepare("insert into mensajes (nombre,  email, mensaje)
values (:nombre, :email, :mensaje)");

$nombre = 'Nombre';
$email = 'email@email.com';
$mensaje = 'Texto mensaje PDO';

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

$consulta->execute();
// Mostramos el último Id insertado
print "<p>Último registro ".$conexion->lastInsertId() . "</p>";

$conexion = null;
?>
```


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

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }
  $consulta = $conexion->prepare("select * from mensajes");
  $consulta->execute();

  while ($registro = $consulta->fetchColumn(1)){
    echo "<p>Nombre: $registro</p>";
  }

  $conexion = null;
?>
```

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

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }
```

```php
  $consulta = $conexion->prepare("select * from mensajes");
  $consulta->execute();

  echo "<p>Número de registros: " .$consulta->rowCount(). "</p>";

  $conexion = null;
?>
```

PDOStatement::errorCode(). Devuelve el código de error de 5 cifras o 00000 en caso de que no haya error

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $consulta = $conexion->prepare("select * from mensajes");
  $consulta->execute();

  echo"<p>Código de error: " .$consulta->errorCode()."</p>";

$conexion = null;
?>
```


PDOStatement::errorInfo(). Devuelve un array con la información del error sobre la última operación realizada por el manejador de la base de datos. El array contiene los siguientes campos:

- En la posición 0: Código de error SQLSTATE (un identificador de cinco caracteres alfanuméricos definidos según el estándar ANSI SQL).
- En la posición 1: Código de error específico del driver.
- En la posición 2: Mensaje del error específico del driver.

```php
<?php
  try
  {
    $mysql =
"mysql:host=localhost;dbname=dwes_tarde_pruebas;charset=UTF8"; $user =
    "dwes_tarde";
    $password = "dwes_2223";
    $opciones = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
    $conexion = new PDO($mysql, $user, $password);

  }
  catch (PDOException $e)
  {
    // Mostramos mensaje en caso de error
    echo "<p>" .$e->getMessage()."</p>";
    exit();
  }

  $consulta = $conexion->prepare("select * from mensajes2");
  $consulta->execute();

  print_r ($consulta->errorInfo());

  $conexion = null;
?>
```

```
Array ( [0] => 42S02 [1] => 1146 [2] => Table
 'dwes_tarde_pruebas.mensajes2' doesn't exist )
```