# PHP
# INTRODUCCIÓN A PDO

# 1. Introducción a PDO

Las siglas PDO (PHP Data Objects) hacen referencia a una interfaz de PHP que nos permite acceder a bases de datos de cualquier tipo en PHP.

Cada controlador de bases de datos que implemente la interfaz PDO puede exponer características específicas de la base de datos, como las funciones habituales de la extensión. Obsérvese que no se puede realizar ninguna de las funciones de la bases de datos utilizando la extensión PDO por sí misma; se debe utilizar un controlador de PDO específico de la base de datos para tener acceso a un servidor de bases de datos.


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

Para saber los controladores PDO disponibles en nuestro sistema:

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

Información oficial sobre PDO en http://php.net/manual/es/book.pdo.php

## 2. Clases PDO, PDOStatement y PDOException

- Clase PDO: Se utiliza para representar la conexión entre PHP y un servidor de bases de datos. http://php.net/manual/es/class.pdo.php
- Clase PDOStatement: representa una sentencia preparada y también nos permite acceder al conjunto de resultados asociado. http://php.net/manual/es/class.pdostatement.php
- Clase PDOException: representa los errores generados PDO. http://php.net/manual/es/class.pdoexception.php

# 3. Acceso a bases de datos con PDO desde PHP

## 3.1. Conexión a la base de datos

$pdo=new PDO("mysql:host=$host;dbname=$dbname",$usuario,$password);
mysql: tipo de base de datos. Podría ser: mssql, sybase, sqlite, etc..

```
// Ejemplo de conexión a diferentes tipos de bases de datos.
# Conectamos a la base de datos
$host='www.veiga.local';
$dbname='cxbasex';
$user='cxbasex';
$pass='xxxxxx';
try {
  # MS SQL Server ySybase con PDO_DBLIB
  $pdo = new PDO("mssql:host=$host;dbname=$dbname, $user, $pass");
  $pdo = new PDO("sybase:host=$host;dbname=$dbname, $user, $pass");
   # MySQL con PDO_MYSQL
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
$user, $pass);
  # Para que genere excepciones a la hora de reportar errores.
  $pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
  # SQLite Database
  $pdo = new PDO("sqlite:my/database/path/database.db");
}
catch(PDOException $e) {
    echo $e->getMessage();
}
// Si todo va bien en $pdo tendremos el objeto que gestionará la
conexión con la base de datos.
```

## 3.2. Cerrar la conexión a la base de datos

Se recomienda cerrar siempre la conexión a la base de datos cuando no se vaya a utilizar más durante nuestro proceso. Hay que recordar que los recursos son limitados y cuando hay pocos usuarios no hay ningún problema, pero si tenemos muchos usuarios simultáneos entonces es cuando surgen problemas al haber alcanzado el número máximo de conexiones con el servidor, por ejemplo. Al cerrar la conexión de forma explícita aceleramos la liberación de recursos para que estén disponibles para otros usuarios.

```
// Si quisiéramos cerrar la conexión con la base de datos simplemente
// podríamos hacer al final del fichero.
$pdo = null;
```

## 4. Excepciones y PDO

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 para gestionar si produce algún error.

Podemos forzar PDO para que trabaje en cualquier de los tres modos siguientes:

- PDO::ERRMODE_SILENT -> Es el modo por defecto. Aquí tendremos que chequear los errores usando ->errorCode() y ->errorInfo().
- PDO::ERRMODE_WARNING -> Genera errores warning PHP pero permitiría la ejecución normal de la aplicación.
- PDO::ERRMODE_EXCEPTION -> Será la forma más utilizada en PDO. Dispara una excepción permitiéndonos gestionar el error de forma amigable.

```
# Activación del modo de trabajo de PDO
$pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT );
$pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING );
// Se recomienda activar esta opción para gestionar los errores con
PDOException
$pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

**Ejemplo de uso**:

```
<?php
# Conectamos a la base de datos
$host='www.veiga.local';
$dbname='cxbasex';
$user='cxbasex';
$pass='xxxxxx';
try {
  $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
$user, $pass);
  $pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
}
catch(PDOException $e) {
    echo "Se ha producido un error al intentar conectar al servidor
MySQL: ".$e->getMessage();
}
```

```php
try {
  # Otro Ejemplo de error ! DELECT en lugar de SELECT!
  $pdo->exec('DELECT name FROM people');
}
catch(PDOException $e) {
    echo "Se ha producido un error en la ejecucion de la consulta:
".$e->getMessage();

    # En este caso hemos mostrado el mensaje de error y además
almacenamos en un fichero los errores generados.
    file_put_contents('PDOErrors.txt', $e->getMessage(), FILE_APPEND);
}


?>
```

**Advertencia**

Si la aplicación no captura la excepción lanzada por el constructor de PDO, la acción predeterminada que toma el motor zend es la de finalizar el script y mostrar información de seguimiento. Esta información probablemente revelará todos los detalles de la conexión a la base de datos, incluyendo el nombre de usuario y la contraseña. Es su responsabilidad capturar esta excepción, ya sea explícitamente (con una sentencia *catch*) o implícitamente por medio de set_exception_handler().

## 5. INSERT, UPDATE y DELETE

Insertar nuevos datos, actualizarlos o borrarlos son algunas de las operaciones más comunes en una base de datos. Con PDO, se suele hacer en un proceso de 2 pasos.


Ejemplo de uso:

```
# $stmt sería un objeto de tipo PDOStatement (consulta preparada)
$stmt = $pdo->prepare("INSERT INTO alumnos( nombre, apellidos) values
( 'Taylor','Swift' )");
# Ejecutamos la consulta con ->execute() método del objeto
PDOStatement
# Este método devuelve true o false.
$stmt->execute();

// Como resultado de la ejecución tendríamos en $stmt un valor true o
false indicado si la instrucción se ha ejecutado correctamente.

// Se podría realizar lo anterior también con la sentencia query o
exec del objeto PDO ahorrándonos una instrucción:
// Es útil cuando son sentencias SQL que no reciben parámetros.

// $pdo->query("consulta SQL") -> Devolverá un objeto de tipo
PDOStatement. (recomendable para SELECT)
// $pdo->exec("consulta SQL") -> Devolverá el número de registros
afectados por la consulta (recomendable para INSERT, UPDATE o DELETE).

// Como resultado de la ejecución tendríamos en $numregistros el
número de filas afectadas por la instrucción.
$numregistros= $pdo->query("INSERT INTO alumnos( nombre, apellidos)
values ( 'Taylor','Swift' )");

# Si queremos borrar registros se podría hacer con:
$pdo->exec("DELETE FROM colegas WHERE apellidos='Swift'");
```

## 6. Consultas preparadas

Una consulta preparada es una sentencia SQL precompilada que se puede ejecutar múltiples veces simplemente enviando datos al servidor.

ATENCIÓN: EL USO DE CONSULTAS PREPARADAS CON PREPARE NOS AYUDARÁ A EVITAR LA INYECCIÓN SQL, CON LO QUE RECOMENDAMOS USAR ->prepare()

Para construir una sentencia preparada hay que hacerlo incluyendo unos marcadores en nuestra sentencia SQL.

3 ejemplos de cómo hacerlo:

```
# Marcadores anónimos
$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) values
(?, ?, ?)");

# Marcadores conocidos
$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) values
(:name, :addr, :city)");

# Aquí no lleva marcadores - ideal para una inyección SQL! ('''no usar
este método'''). !! Hay que usar los marcadores !!
$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) values
($name, $addr, $city)");
```

Deberás usar el primer o segundo método de los mostrados anteriormente. La elección de usar marcadores anónimos o conocidos afectará a cómo se asignan los datos a esos marcadores.

## 7. Asignación con marcadores anónimos

Para vincular los marcadores anónimos con su correspondiente valor se puede utilizar bindParam o bindValue:

ATENCION: $pdo->prepare() usando marcadores anónimos ? ,trata todas las variables como si fueran cadenas, por lo que usará comillas para delimitar sus valores por defecto.

```
# Asignamos variables a cada marcador, indexados del 1 al 3
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $addr);
$stmt->bindParam(3, $city);

# Insertamos una fila.
$name = "Daniel"
$addr = "1 Wicked Way";
$city = "Arlington Heights";
$stmt->execute();

# Insertamos otra fila con valores diferentes.
$name = "Steve"
$addr = "5 Circle Drive";
$city = "Schaumburg";
$stmt->execute();
```

Otra forma de asignación con marcadores anónimos a través de un array asociativo:

```
# Los datos que queremos insertar
$datos = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');

$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) values
(?, ?, ?)");
$stmt->execute($datos);
```

Diferencia entre el uso de bindParam y bindValue

- Con bindParam se vincula la variable al parámetro y en el momento de hacer el execute es cuando se asigna realmente el valor de la variable a ese parámetro.
- Con bindValue se asigna el valor de la variable a ese parámetro justo en el momento de ejecutar la instrucción bindValue.

Ejemplo de diferencia entre bindParam y bindValue:

```php
// Ejemplo con bindParam:
$sex = 'hombre';
$s = $dbh->prepare('SELECT name FROM studiantes WHERE sexo = :sexo');
$s->bindParam(':sexo', $sex);
$sex = 'mujer';
$s->execute(); // se ejecutó con el valor WHERE sexo = 'mujer'

// El mismo ejemplo con bindValue:
$sex = 'hombre';
$s = $dbh->prepare('SELECT name FROM students WHERE sexo = :sexo');
$s->bindValue(':sexo', $sex);
$sex = 'mujer';
$s->execute(); // se ejecutó con el valor WHERE sexo = 'hombre'
```

## 8. Asignación con marcadores conocidos

Los marcadores conocidos es la forma más recomendable de trabajar con PDO, ya que a la hora de hacer el bindParam o el bindValue se puede especificar el tipo de datos y la longitud máxima de los mismos.

- Información sobre bindParam
- Información sobre bindValue
- Constantes y Tipos de Datos utilizados en bindParam y bindValue

Formato de bindParam con marcadores conocidos:

```
   bindParam(':marcador', $variableVincular, TIPO DATOS PDO)
```

Ejemplo de uso de bindParam:

```
   $stmt->bindParam(':calorias', $misCalorias, PDO::PARAM_INT);
   $stmt->bindParam(':apellidos', $misApellidos, PDO::PARAM_STR, 35);
// 35 caracteres como máximo.
```

Con marcadores conocidos quedaría de la siguiente forma:

```
# El primer argumento de bindParam es el nombre del marcador y el
# segundo la variable que contendrá los datos.
# Los marcadores conocidos siempre comienzan con :
$stmt->bindParam(':name', $name);
$name='Pepito';
$stmt->execute();
# Otra forma es creando un array asociativo con los marcadores y sus
# valores:
# Los datos a insertar en forma de array asociativo
$datos = array( 'name' => 'Cathy', 'addr' => '9 Dark and Twisty',
'city' => 'Cardiff' );

# Fijarse que se pasa el array de datos en execute().
$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) value
(:name, :addr, :city)");
$stmt->execute($datos);
# La última instrucción se podría poner también así:
$stmt->execute(array(
 'name' => 'Cathy',
 'addr' => '9 Dark and Twisty',
 'city' => 'Cardiff'
```

```
 ));
```

Otra característica de los marcadores conocidos es que nos permitirán trabajar con objetos directamente en la base de datos, asumiendo que las propiedades de ese objeto coinciden con los nombres de los campos de la tabla en la base de datos.

Ejemplo de uso de marcadores conocidos y objetos:

```
# Un objeto sencillo

class person {
    public $name;
    public $addr;
    public $city;

    function __construct($n,$a,$c) {
        $this->name = $n;
        $this->addr = $a;
        $this->city = $c;
    }
    # etc ...
}

$cathy = new person('Cathy','9 Dark and Twisty','Cardiff');

# Preparación de la consulta
$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) value
(:name, :addr, :city)");

# Inserción del objeto
$stmt->execute((array)$cathy);
```

# 9. Ejemplos CRUD: INSERT, UPDATE y DELETE

## 9.1. INSERT

```
try {
  $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
$username, $password);
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $pdo->prepare('INSERT INTO someTable VALUES(:name)');
  $stmt->execute(array(
    ':name' => 'Justin Bieber'
  ));

  # Affected Rows?
  echo $stmt->rowCount(); // 1
} catch(PDOException $e) {
  echo 'Error: ' . $e->getMessage();
```

## 9.2. UPDATE

```
$id = 5;
$name = "Joe the Plumber";

try {
  $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
$username, $password);
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $pdo->prepare('UPDATE someTable SET name = :name WHERE id =
:id');
  $stmt->execute(array(
    ':id'   => $id,
    ':name' => $name
  ));

  echo $stmt->rowCount(); // 1
} catch(PDOException $e) {
  echo 'Error: ' . $e->getMessage();
```

## 9.3. DELETE

```php
$id = 5; // From a form or something similar
try {
  $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
$username, $password);
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $pdo->prepare('DELETE FROM someTable WHERE id = :id');
  $stmt->bindParam(':id', $id); // this time, we'll use the bindParam
method
  $stmt->execute();

  echo $stmt->rowCount(); // 1
} catch(PDOException $e) {
  echo 'Error: ' . $e->getMessage();
}
```

## 9.4. SELECT

Los datos se obtienen a través del método ->fetch() o ->fetchAll() (PDOStatement).

- ->fetch() : Obtiene la siguiente fila de un recordset (conjunto de resultados). http://php.net/manual/es/pdostatement.fetch.php
- ->fetchAll(): Devuelve un array que contiene todas las filas del conjunto de resultados (el tipo de datos a devolver se puede indicar como parámetro). http://php.net/manual/es/pdostatement.fetchall.php

Antes de llamar al método ->fetch() una buena idea es indicarle cómo queremos que nos devuelva los datos de la base de datos.

Tendremos las siguientes opciones en el método ->fetch():

- PDO::FETCH_ASSOC: devuelve un array indexado por el nombre de campo de la tabla.
- PDO::FETCH_BOTH: (por defecto): devuelve un array indexado por nombre de campo de la tabla y por número de campo.
- PDO::FETCH_BOUND: Asigna los valores devueltos a las variables asignadas con el método ->bindColumn().
- PDO::FETCH_CLASS: Asigna los valores de los campos a las propiedades de una clase. Si las propiedades no existen en esa clase, las creará.
- PDO::FETCH_INTO: Actualiza una instancia existente de una clase.
- PDO::FETCH_LAZY: Combina PDO::FETCH_BOTH/PDO::FETCH_OBJ, creando las variables del objeto a medida que se van usando.
- PDO::FETCH_NUM: Devuelve un array indexado por el número de campo.
- PDO::FETCH_OBJ: Devuelve un objeto anónimo con los nombres de las propiedades que se corresponden con los nombres de columnas.

Para ajustar el modo de respuesta:

```
$stmt->setFetchMode(PDO::FETCH_ASSOC);
```

## 9.5. FETCH_ASSOC

Para ejecutar la consulta SELECT si no tenemos parámetros en la consulta podremos usar ->query() del objeto PDO

Veamos un ejemplo de consulta SELECT:

```
try {
        # Para ejecutar la consulta SELECT si no tenemos parámetros en
la consulta podremos usar ->query()
        $stmt = $pdo->query('SELECT name, addr, city from colegas');
        # Indicamos en qué formato queremos obtener los datos de la
tabla en formato de array asociativo.
        # Si no indicamos nada por defecto se usará FETCH_BOTH lo que
nos permitirá acceder como un array asociativo o array numérico.
        $stmt->setFetchMode(PDO::FETCH_ASSOC);

        # Leemos los datos del recordset con el método ->fetch()
        while ($row = $stmt->fetch()) {
                echo $row['name'] . "<br/>";
                echo $row['addr'] . "<br/>";
                echo $row['city'] . "<br/>";
        }

        # Para liberar los recursos utilizados en la consulta SELECT
        $stmt = null;
} catch (PDOException $err) {
    // Mostramos un mensaje genérico de error.
        echo "Error: ejecutando consulta SQL.";
}
# Para ejecutar la consulta SELECT si tenemos parámetros lo haríamos
# así:
# Preparamos la consulta como siempre
$stmt = $pdo->prepare('SELECT name, addr, city from colegas where city
=:ciudad');
```

```
try{
        # Indicamos en qué formato queremos obtener los datos de la
tabla en formato de array asociativo.
        # Si no indicamos nada por defecto se usará FETCH_BOTH lo que
nos permitirá acceder como un array asociativo o array numérico.
        $stmt->execute(array('ciudad'=>'Santiago de Compostela');
        # Leemos los datos del recordset con el método ->fetch()
        # Por defecto ya los devuelve en forma de array asociativo si
       #no indicamos nada con setFetchMode()
        while ($row = $stmt->fetch())  {
                echo $row['name'] . "<br/>";
                echo $row['addr'] . "<br/>";
                echo $row['city'] . "<br/>";
        }
        # Para liberar los recursos de la consulta SELECT
        $stmt=null;
}
catch(PDOException $err)
        {
                // Mostramos un mensaje genérico de error.
                echo "Error: ejecutando consulta SQL.";
        }
```

## 9.6. FETCH_OBJ

En este tipo de modo de consulta se creará un objeto standard por cada fila que leemos del recordset.

Ejemplo:

```
try{
        # Creamos la consulta
        $stmt = $pdo->query('SELECT name, addr, city from colegas');
        # Ajustamos el modo de obtención de datos
        $stmt->setFetchMode(PDO::FETCH_OBJ);
        # Mostramos los resultados.
        # Fijaros que se devuelve un objeto cada vez que se lee una
         #fila del recordset.
        while($row = $stmt->fetch()) {
                echo $row->name . "<br/>";
                echo $row->addr . "<br/>";
                echo $row->city . "<br/>";
        }
        # Liberamos los recursos utilizados por $stmt
        $stmt=null;
}
```

```
catch(PDOException $err)
{
        // Mostramos un mensaje genérico de error.
        echo "Error: ejecutando consulta SQL.";
}
```

# 10. Otros métodos interesantes

La extensión PDO es muy grande para ver aquí todas sus opciones, pero aquí tenéis un para de métodos interesantes y que os podrán ser útiles en vuestras aplicaciones.

## 10.1. lastInsertId()

Devuelve el id del último registro insertado en la tabla. Es un método de PDO

```
$pdo->lastInsertId();
```

## 10.2. quote()

Si no usas consultas preparadas, la forma de protegerte contra la inyección SQL es usando este método:

```
$sqlSegura= $pdo->quote($sqlInsegura);
```

## 10.3. rowCount()

Devuelve un entero indicando el número de filas afectadas por la última operación.

```
$rows_affected = $stmt->rowCount();
```

# 11. Resumen del acceso a bases de datos con PDO

## 11.1. 1.- Conectar a la base de datos usando try/catch.

```
$pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $user,
$pass");
```

## 11.2. Preparar la consulta (insert,update,delete).

```
        $stmt = $pdo->prepare("INSERT INTO alumnos( nombre, apellidos)
values ( 'Taylor','Swift' )");
        $stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city)
values (?, ?, ?)");
        $stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city)
value (:name, :addr, :city)");
```

## 11.3. Asignar parámetros en la consulta:

```
        $stmt->bindParam(':name', $name);
        $name='Pepito';

        $datos = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');
        $stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city)
values (?, ?, ?)");

        $datos = array( 'name' => 'Cathy', 'addr' => '9 Dark and
Twisty', 'city' => 'Cardiff' );
        $stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city)
value (:name, :addr, :city)");
```

## 11.4. Ejecutar la consulta

```
try{
        $stmt->execute();

        // o bien
        $stmt->execute($datos);
}
catch(PDOException $err) {
   // Mostramos un mensaje genérico de error.
   echo "Error: ejecutando consulta SQL.";
}
```

## 11.5. Preparar la consulta (select).

```
try{
       // ATENCION: si no tenemos parámetros en la consulta, la
//podemos ejecutar con ->query (recomendable en SELECT) o con ->exec
//(para INSERT, UDPATE, DELETE)
        $stmt = $pdo->query('SELECT name, addr, city from colegas');
}
catch(PDOException $err) {
   // Mostramos un mensaje genérico de error.
   echo "Error: ejecutando consulta SQL.";
}
$stmt = $pdo->prepare('SELECT name, addr, city from colegas where city
=:ciudad'); (con parámetros)
$datos = array( ':ciudad' => 'Santiago');

try{
        $stmt->execute($datos);
}
catch(PDOException $err) {
   // Mostramos un mensaje genérico de error.
   echo "Error: ejecutando consulta SQL.";
}
```

## 11.6. Leemos los datos del recordset

Leemos los datos del recordset (conjunto de registros) que nos devuelve SELECT en el objeto PDOStatement.

Se puede leer cada fila del recordset con ->fetch() del objeto PDOStatement o mediante ->fetchAll() (obtiene todas las filas del recordset).

```
        while($row = $stmt->fetch()) {
            echo $row['name'] . "<br/>";
            echo $row['addr'] . "<br/>";
            echo $row['city'] . "<br/>";
        }
        $row = $sql->fetchAll();
        foreach($data as $row){
            $id = $row['id'];
            $content = $row['content'];
        }
```

## 11.7. Cerrar la conexión.

```
// Liberamos los recursos utilizados por el recordset $stmt
    $stmt=null;
// Se recomienda cerrar la conexión para liberar recursos de forma más
// rápida.
    $pdo = null;
```