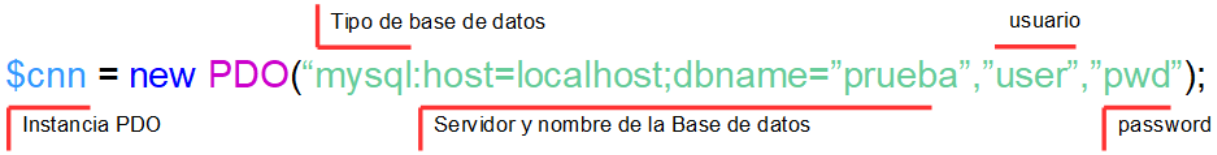
Consultas a bases de datos MySql desde PHP con PDO

Hasta ahora, para conectarnos a MySql desde Php, hemos utilizado el API de mysqli que ya conoces.

Podemos utilizar este API claro, o mejor aún PDO (PHP Data Objects). Utilizando PDO podemos solventar muchas dificultades que surgen al utilizar la API de mysqli. 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 mysqli_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.

PDO se basa en las características de orientación a objetos de PHP pero, al contrario que la extensión mysqli, no ofrece un *interface* de programación dual. Para acceder a las funcionalidades de la extensión tenemos que emplear los objetos que ofrece, con sus métodos y propiedades. No existen funciones alternativas. Aunque después de la unidad anterior, esto no te debe suponer ningún problema.

Veamos como podemos utilizar PDO en una aplicación para recuperar datos de una base de datos MySql, y lo sencillo que es usar esta API. Lo primero que tenemos que hacer, como siempre, es realizar la conexión a la base de datos. La conexión la realizaremos con el constructor de la clase de la siguiente forma:



Con la llamada anterior ya tenemos creada la conexión a la base de datos. Antes de continuar veamos **como tratar los posibles errores** con PDO. 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*** (seguro que os suenan de Java). De esta forma cada vez que ocurra un error saltará una excepción que capturaremos y podremos tratar correctamente para mostrarle un mensaje al usuario (o más bien para hacer lo que nos interese). Para realizar esta tarea utilizaremos la función setAttribute() de la siguiente forma:

```
$conn = new PDO('mysql:host=localhost;dbname=nombreBaseDatos', 'user', 'pass');
$conn→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 mysqli 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 utilizamos los bloques try-catch: (en este ejemplo simplemente se muestra por pantalla un mensaje que incluye el error devuelto…. En un caso real esto no es aconsejable, haremos lo que nos interese como programadores….redirigir a index con algún mensaje??)

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

Ahora que sabemos como conectarnos a la base de datos, vamos a crear una sentencia para poder recuperar datos. Para ejecutar sentencias podemos utilizar query() o bien prepare(). Aunque tenemos disponibles las dos llamadas **es mucho más seguro utilizar el método prepare()** ya que esta **se encarga de escapar por nosotros los parámetros y nos asegura que no sufriremos problemas de SQL Injection**. El método **query()** se suele utilizar cuando la sentencia que vamos a ejecutar no contiene parámetros que ha enviado el usuario. Veamos un ejemplo utilizando **query()**:

```php
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();
}
```

¿Has visto el ejemplo del uso de query()? Pues ya te puedes olvidar, nosotros utilizaremos siempre **prepare().**

La forma de utilizar la función **prepare() ,** la opción más recomendada para nosotros, es la siguiente:

```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 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 PDOStatement sobre el cual ejecutaremos la función execute() para que consulte los datos. A continuación simplemente los tenemos que recorrer con ayuda del método fetch() para poder mostrar su valor.

Si necesitamos pasarle valores a la sentencia sql (cosa de lo más habitual, claro) utilizaremos los **parámetros**. Los parámetros los indicamos en la misma sentencia sql y los podemos escribir de dos formas distintas. Mediante el signo ? o mediante un nombre de variable precedido por el símbolo ':' , algo como *:nombreParam*. Esta segunda forma,

nos permite una identificación más fácil de los parámetros y es la que utilizaremos, pero cualquiera de las dos formas es correcta. Veamos un ejemplo:

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

Como podéis ver hemos llamado al parámetro *:apellidos* y posteriormente en la llamada a la función **execute()** indicamos con un array asociativo el nombre del parámetro y su valor. Y ahí tenemos nuestras ***"consultas preparadas"***. Otra forma de indicar los parámetros, muy utilizada y aconsejable para ganar claridad, es utilizando la función bindParam, de esta manera:

```php
$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->bindParam(':apellidos', $ape, PDO::PARAM_INT);
  $stmt->execute();

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

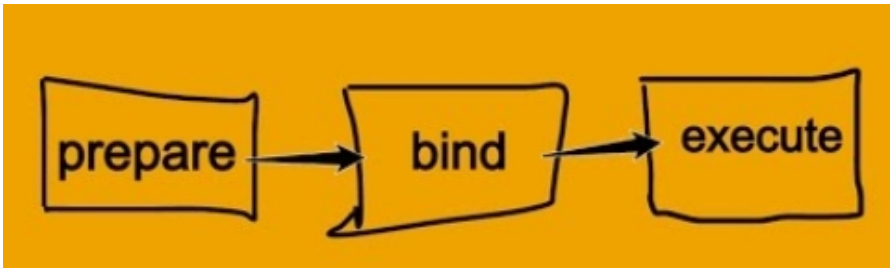
A la función bindParam() le pasamos el nombre del parámetro, su valor y finalmente el tipo que es. Los tipos de parámetros que le podemos pasar los podemos ver en las constantes predefinidas de PDO y son:

•PDO::PARAM_BOOL

•PDO::PARAM_NULL

•PDO::PARAM_INT

•PDO::PARAM_STR



Del mismo modo que las sentencias select, podemos utilizar las funciones query() y **prepare()** (prepare(), siempre prepare()) para ejecutar **inserts**, **updates** y **deletes**. La forma de hacerlo es igual que lo que hemos estado viendo hasta ahora:

Ejemplo de insert:

```php
$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)'
  );
  $rows = $stmt->execute(array(':nombre'    => $nom, ':apellidos' => $ape));

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

Ejemplo de update:

```php
$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'
  );
  $rows = $stmt->execute( array( ':nombre'    => $nom,
                                 ':apellidos' => $ape));
  if( $rows > 0 )
    echo 'Actualización correcta';
} catch(PDOException $e) {
  echo 'Error: ' . $e->getMessage();
}
```

Ejemplo de delete:

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

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

Para acabar con esta breve introducción 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.

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

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

Hay que tener en cuenta que para que funcione correctamente, el nombre de los atributos en nuestra clase tienen que ser iguales que los que tienen las columnas en nuestra tabla de la base de datos.

Con esto claro, vamos a realizar la consulta.

```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');

  while($usuario = $stmt->fetch())
    echo $usuario->nombreApellidos() . '<br />';

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

La novedad que podemos ver en este último ejemplo 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 en un array los obtendremos en el objeto indicado.