**Conectarse a una base de datos.**

Para empezar, crearemos una base de datos llamada “Biblioteca”, yo la crearé con código, pero ustedes pueden hacerlo como quieran:

CREATE DATABASE Biblioteca;

Ahora que ya tenemos nuestra base de datos podemos comenzar intentando conectar PHP con la misma mediante un objeto PDO. Para ello necesitaremos 5 datos: El nombre de la base de datos con la que nos vamos a conectar, el servidor, el nombre de usuario y su contraseña. Pero también debemos indicar a qué tipo de base de datos nos conectaremos, en mi caso mysql, aunque también podría ser otra como por ejemplo oracle. Probemos conectarnos de esta forma:

<?php

$pdo = new PDO('mysql:host=localhost;dbname=Biblioteca', 'root', '');

?>

Por empezar, la clase PDO no necesita ser incluida, la misma está disponible desde la versión 5 de PHP y existe en memoria. Al crear el objeto necesitará tres parámetros.

El primer parámetro será una cadena donde tendré que ingresar el tipo de base de datos, mysql por ejemplo; el host o servidor, acá será localhost o 127.0.0.1 y el nombre de la base de datos, en este caso “Biblioteca”. También puede agregarse el puerto, separado por punto y coma.

El segundo parámetro es el nombre del usuario y el tercero la contraseña. En mi caso el nombre de usuario es **root** y la contraseña es una cadena vacía.

Existe un cuarto parámetro no obligatorio que es un array con opciones para la conexión.

Hay ocasiones en que nuestro código puede lanzar errores, aunque el código esté correcto. Por ejemplo, nosotros subimos nuestro sitio a un hosting y de pronto la base de datos deja de funcionar, algún dato cambia de nombre, etcétera. Para solucionar este problema usaremos la sentencia try catch, y de esta manera tendremos un control por si nuestro script pudiese tener problemas al conectarse con la base de datos:

<?php

try{

$pdo = new PDO('mysql:host=localhost;dbname=biblioteca', 'root', '');

}catch(PDOException $e){

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

exit;

}

?>

Para probar el error pueden por ejemplo cambiar algún dato, como en el nombre de la base de datos poniendo un nombre que no existe, o apagar la base de datos y comprobarlo.

Ahora que ya sabemos cómo usar la clase PDO para conectarse a una base podríamos crear una clase propia que herede todos los métodos de la misma, pero que incluya dentro los datos necesarios para conectarse. De esta manera no tendríamos que ingresar todos los datos cada vez que en nuestro proyecto quisiéramos conectarnos a la base de datos. Creemos una clase llamada **Conexion**:

<?php

class Conexion extends PDO {

private $tipodb = 'mysql';

private $host = 'localhost';

private $nombre\_de\_base = 'biblioteca';

private $user = 'root';

private $psw = '';

public function \_\_construct() {

//Sobreescribo el método constructor de la clase PDO.

try{

parent::\_\_construct($this->tipobd.':host='.$this->host.';dbname='.$this->nombre\_de\_base, $this->user, $this->psw);

}catch(PDOException $e){

echo 'Ha surgido un error y no se puede conectar a la base de datos. Detalle: ' . $e->getMessage();

exit;

}

}

}

?>

Cómo ven la clase es muy corta, hereda de PDO, y sobrescribe el constructor padre ingresando los datos para conectarse que están guardados en propiedades.

Entonces nosotros podemos conectarnos con:

<?php

require\_once 'Conexion.php';

$conn = new Conexion();

?>

Para trabajar con la misma utilizamos dos clases: La clase Conexion que hereda de PDO y permite conectarse a la base de datos y la clase **Biblioteca**, con todo lo relacionado a la tabla **Biblioteca**, y que además usa la clase **Conexion** para hacer consultas con la base de datos.

<?php

require\_once 'Conexion.php';

class Personaje {

private $isbn;

private $nombre;

private $autor;

const TABLA = 'libro';

public function getIsbn() {

return $this->isbn;

}

public function getNombre() {

return $this->nombre;

}

public function getAutor() {

return $this->autor;

}

public function setNombre($nombre) {

$this->nombre = $nombre;

}

public function setDescripcion($descripcion) {

$this->descripcion = $descripcion;

}

public function \_\_construct($nombre, $descipcion, $id=null) {

$this->nombre = $nombre;

$this->descripcion = $descipcion;

$this->id = $id;

}

}

?>

El método **constructor** es el encargado de inicializar un nuevo objeto libro, que con estos valores lo vamos a emplear en un futuro en el método de creación o actualización.

Como cada atributo de la clase es privada, solo se puede acceder desde una función de la misma clase, por este motivo se declaró cada método SET y GET de cada atributo y a través de estos métodos podemos implementar.

Método **buscarPorId**.

El método buscará un personaje por su id, si lo encuentra devolverá un objeto con sus datos guardados en las propiedades correspondientes, pero sino devolverá false. Este método es declarado estático porque solo se desea

Ahora analicemos el código.

Creamos una nueva conexión para realizar consultas, nada nuevo:

$conexion = new Conexion();

Escribimos la consulta que buscará un personaje por su id:

$consulta = $conexion->prepare('SELECT nombre, descripcion FROM ' . self::TABLA . ' WHERE id = :id');

Agregamos los valores a la consulta:

$consulta->bindParam(':id', $id);

La ejecutamos:

$consulta->execute();

En la siguiente línea lo que hacemos es guardar la respuesta de la consulta mediante un método llamado fetch():

$registro = $consulta->fetch();

Lo que hace el método fetch() es devolvernos un array asociativo con los datos que le pedimos, en caso de encontrar un resultado (fetch() se usa para consultas que deberían devolver un solo registro) Pero en caso de no encontrar nada nos devolverá false.

Preguntamos si nos devolvió algún resultado. Si así fue retornamos un objeto Personaje con las propiedades $id, $nombre y $descripcion cargadas con sus valores correspondientes. Pero si la consulta no encontró nada con ese id nos devolverá false, osea nada.

if($registro){

return new self($registro['nombre'], $registro['descripcion'], $id);

}else{

return false;

}

Intenten probarlo con un id que exista en la base de datos:

<?php

require\_once 'ejemplo2.php';

$personaje = Personaje::buscarPorId(1);

if($personaje){

echo $personaje->getNombre();

echo '<br />';

echo $personaje->getAutor();

}else{

echo 'El personaje no ha podido ser encontrado';

}

?>

Y ahora que tenemos un objeto creado y recuperado por su id, podríamos cambiarle por ejemplo la descripción mediante el método setAutor()