***CONEXIÓN A BASE DE DATOS EN POO***

Para empezar crearemos una base de datos llamada ‘SENA’, yo la crearé con código pero ustedes pueden hacerlo como quieran:

CREATE DATABASE SENA;

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=SENA', '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 ‘SENA’. 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, pero en este posteo lo omitiré.

Bueno, ahora siguiendo con el código, voy a aclarar algo que está faltando. 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=SENA', '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 **Conexión**:

<?php

class Conexion extends PDO {

   private $tipo\_de\_base = 'mysql';

   private $host = 'localhost';

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

   private $usuario = 'root';

   private $contrasena = '';

   public function \_\_construct() {

      //Sobreescribo el método constructor de la clase PDO.

      try{

         parent::\_\_construct($this->tipo\_de\_base.':host='.$this->host.';dbname='.$this->nombre\_de\_base, $this->usuario, $this->contrasena);

      }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 'clases/Conexion.php';

$conexion = new Conexion();

?>

***GUARDAR REGISTROS EN BASE DE DATOS.***

En primer lugar, seguiré con la misma base de datos, con el nombre **SENA**, de esta forma:

CREATE DATABASE SENA;

Ahora crearé una tabla para guardar los personajes de esta serie:

CREATE TABLE personaje(

id tinyint(3) unsigned not null auto\_increment primary key,

nombre varchar(255) not null,

descripcion text not null

)

Bueno, hasta acá nada nuevo, creé una base de datos con una sola tabla que guardará los nombres y una descripción de los personajes, y por supuesto tendrá un id o clave primaria.

Ahora, para probar mis consultas crearé dos clases. Una será una clase para conectarse a la base de datos, la misma que usé en el posteo anterior:

<?php

class Conexion extends PDO {

   private $tipo\_de\_base = 'mysql';

   private $host = 'localhost';

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

   private $usuario = 'root';

   private $contrasena = '';

   public function \_\_construct() {

      //Sobreescribo el método constructor de la clase PDO.

      try{

         parent::\_\_construct($this->tipo\_de\_base.':host='.$this->host.';dbname='.$this->nombre\_de\_base, $this->usuario, $this->contrasena);

      }catch(PDOException $e){

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

         exit;

      }

   }

}

?>

Y luego crearé una segunda clase que se encargará de todo lo referido con consultas con la tabla de personajes que se llame **Personaje**, en la misma carpeta de la clase **Conexión**. Pero antes de mostrarles el código paremos aquí.

Primero pensemos en qué propiedades va a necesitar nuestra clase **Personaje**. Por cada registro necesitará guardar el id, el nombre y la descripción del personaje, pero también para realizar consultas necesitará a qué tabla apuntará. Así que crearemos tres propiedades para guardar los datos de cada personaje y una constante donde guardará el nombre de la tabla. Probemos de esta forma:

<?php

require\_once 'Conexion.php';

class Personaje {

private $id;

private $nombre;

private $descripcion;

const TABLA = 'personaje';

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

$this->nombre = $nombre;

$this->descripcion = $descipcion;

$this->id = $id;

}

}

?>

Ok, echemos un vistazo a esta clase. En primer lugar, importé la clase **Conexion**, ya que más adelante tendré que hacer consultas a la base y necesitaré mucho de la misma. Luego creé tres propiedades para guardar los datos de cada personaje y una constante con el nombre de la tabla. Y por último creé un constructor donde tendré que setear el valor de las propiedades. Si se han fijado bien, el último parámetro, **$id**, es un parámetro que por defecto es **null**, esto significa que en el momento de crear el constructor si yo no ingreso ese último parámetro tomará ese valor, nulo.  Esto se debe a que no siempre el id tendrá un valor definido, por ejemplo, si yo estoy creando un nuevo usuario no ingresaré el valor del id ya que eso será auto incrementable y el valor lo recuperaré una vez insertado el registro.

Continuemos, necesitaremos un método que nos permita insertar y modificar datos en la tabla **personaje**, para ello crearemos un método llamado **guardar()**, que nos servirá para insertar y modificar registros. Para ello crearemos un objeto de la clase **Conexion** que por medio de los métodos que heredó de la clase **PDO** podrá realizar  esto. La clase **Personaje** ahora quedaría así:

<?php

require\_once 'Conexion.php';

class Personaje {

   private $id;

   private $nombre;

   private $descripcion;

   const TABLA = 'personaje';

   public function getId() {

      return $this->id;

   }

   public function getNombre() {

      return $this->nombre;

   }

   public function getDescripcion() {

      return $this->descripcion;

   }

   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;

   }

   public function guardar(){

      $conexion = new Conexion();

      if($this->id) /\*Modifica\*/ {

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

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

         $consulta->bindParam(':descripcion', $this->descripcion);

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

         $consulta->execute();

      }else /\*Inserta\*/ {

         $consulta = $conexion->prepare('INSERT INTO ' . self::TABLA .' (nombre, descripcion) VALUES(:nombre, :descripcion)');

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

         $consulta->bindParam(':descripcion', $this->descripcion);

         $consulta->execute();

         $this->id = $conexion->lastInsertId();

      }

      $conexion = null;

   }

}

?>

Ok, analicemos los cambios. En primer lugar agregué los métodos getters y setters que me estaban faltando, luego creé un método **guardar()** que primero crea un objeto **Conexion**, para así luego por medio de ese objeto poder conectarse a la base de datos:

$conexion = new Conexion();

Luego pregunté por el valor del id mediante un **if**, si el valor es cualquier cosa que no sea 0, **false** o **null**entonces entrará en el lado verdadero del condicional. Esto sirve para indicarle a nuestro método si tendrá que hacer un **update** o un **insert**, si el valor del id es **algo**significa existe un id, por ende se está modificar un usuario con ese valor, de lo contrario se está intentando insertar un usuario nuevo.

Analicemos el lado verdadero del condicional, osea el **update**:

Primero escribimos la consulta que modificará los datos del usuario:

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

Como ven creamos un nuevo objeto **$consulta**, que nos devuelve el método **prepare()**. Aquí es donde se define la consulta que se está haciendo en la base.

Luego por medio del método **bindParam()** agregamos los valores de la consulta, por ejemplo **:nombre**será el valor de la propiedad **$nombre**:

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

$consulta->bindParam(':descripcion', $this->descripcion);

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

Y finalmente ejecutamos la consulta:

$consulta->execute();

El lado falso del **if**, osea el que sirve para insertar un nuevo registro es muy parecido al del update, sólo cambia la consulta (obviamente) y que no debe pasarle el valor del id, porque como dijimos antes, en el **INSERT** no hay id, sino que se auto incrementa en la consulta.

Por eso al finalizar de ejecutar la consulta se recupera el id que se acaba de insertar:

$this->id = $conexion->lastInsertId();

El método **lastInsertId()** me permite recuperar la última clave primaria o id insertada en la base de datos, es un método de la clase **PDO**.

Finalmente igualo el objeto **$conexion** a **null** para cerrar la conexión:

$conexion = null;

Ahora para comprobar que todo ha salido bien probaremos crear  un objeto **Personaje** para insertar un registro en la base de datos:

<?php

require\_once 'clases/Personaje.php';

$persona = new Personaje('Batman', 'Encapuchado, disfrazado de murciélago que sale por las noches a combatir el mal.', 1);

$persona->guardar();

echo $persona->getNombre() . ' se ha guardado correctamente con el id: ' . $persona->getId();

?>

Al no ingresarle en el constructor el tercer parámetro no obligatorio **$id** el método **guardar()**interpretará que se está intentado insertar un nuevo registro. Prueben ejecutar este script y verán que se ha insertado un nuevo registro en la tabla **personaje**.

***BUSCAR REGISTROS.***

Repasando un poco lo que vimos, primero creamos una base de datos:

CREATE DATABASE SENA;

Luego hicimos una tabla para guardar los personajes de la saga Batman:

CREATE TABLE personaje(

id tinyint(3) unsigned not null auto\_increment primary key,

nombre varchar(255) not null,

descripcion text not null

)

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 **Personaje**, con todo lo relacionado a la tabla **personaje**,  y que además usa la clase **Conexion** para hacer consultas con la base de datos.

Conexion:

<?php

class Conexion extends PDO {

   private $tipo\_de\_base = 'mysql';

   private $host = 'localhost';

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

   private $usuario = 'root';

   private $contrasena = '';

   public function \_\_construct() {

      //Sobreescribo el método constructor de la clase PDO.

      try{

         parent::\_\_construct($this->tipo\_de\_base.':host='.$this->host.';dbname='.$this->nombre\_de\_base, $this->usuario, $this->contrasena);

      }catch(PDOException $e){

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

         exit;

      }

   }

}

?>

Personaje:

<?php

require\_once 'Conexion.php';

class Personaje {

   private $id;

   private $nombre;

   private $descripcion;

   const TABLA = 'personaje';

   public function getId() {

      return $this->id;

   }

   public function getNombre() {

      return $this->nombre;

   }

   public function getDescripcion() {

      return $this->descripcion;

   }

   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;

   }

   public function guardar(){

      $conexion = new Conexion();

      if($this->id) /\*Modifica\*/ {

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

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

         $consulta->bindParam(':descripcion', $this->descripcion);

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

         $consulta->execute();

      }else /\*Inserta\*/ {

         $consulta = $conexion->prepare('INSERT INTO ' . self::TABLA .' (nombre, descripcion) VALUES(:nombre, :descripcion)');

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

         $consulta->bindParam(':descripcion', $this->descripcion);

         $consulta->execute();

         $this->id = $conexion->lastInsertId();

      }

      $conexion = null;

   }

}

?>

Esta última clase nos permitía crear un objeto con los datos del personaje y luego por medio de un método **guardar ()** podíamos insertar o modificar un registro dependiendo de si le pasábamos un id o no:

<?php

require\_once 'clases/Personaje.php';

$persona = new Personaje('batman', 'Encapuchado, disfrazado de murciélago que sale por las noches a combatir el mal.');

$persona->guardar();

echo $persona->getNombre() . ' se ha guardado correctamente con el id: ' . $persona->getId();

?>

Con este código el método **guardar()** insertaba un nuevo registro ya que nosotros no le pasamos ningún id, si no existe id significa que el registro es nuevo, pero si la propiedad **$id** no está vacía intentará modificar el registro con dicho valor.

Ahora crearemos un nuevo método que buscará un registro en la base de datos con un id específico con un nuevo método, **buscarPorId()**:

<?php

require\_once 'Conexion.php';

class Personaje {

private $id;

private $nombre;

private $descripcion;

const TABLA = 'personaje';

public function getId() {

return $this->id;

}

public function getNombre() {

return $this->nombre;

}

public function getDescripcion() {

return $this->descripcion;

}

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;

}

public function guardar(){

$conexion = new Conexion();

if($this->id) /\*Modifica\*/ {

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

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

$consulta->bindParam(':descripcion', $this->descripcion);

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

$consulta->execute();

}else /\*Inserta\*/ {

$consulta = $conexion->prepare('INSERT INTO ' . self::TABLA .' (nombre, descripcion) VALUES(:nombre, :descripcion)');

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

$consulta->bindParam(':descripcion', $this->descripcion);

$consulta->execute();

$this->id = $conexion->lastInsertId();

}

$conexion = null;

}

public static function buscarPorId($id){

$conexion = new Conexion();

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

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

$consulta->execute();

$registro = $consulta->fetch();

if($registro){

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

}else{

return false;

}

}

}

?>

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**.

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. En mi caso el id 1 es Batman:

<?php

require\_once 'clases/Personaje.php';

$personaje = Personaje::buscarPorId(1);

if($personaje){

echo $personaje->getNombre();

echo '<br />';

echo $personaje->getDescripcion();

}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 **setDescripcion()** y probar si lo modifica correctamente con el método **guardar()**:

<?php

require\_once 'clases/Personaje.php';

$personaje = Personaje::buscarPorId(1);

if($personaje){

$personaje->setDescripcion('En realidad es el millonario Bruno Diaz');

$personaje->guardar();

echo 'El personaje ha sido modificado';

}else{

echo 'El personaje no ha podido ser encontrado';

}

?>

Bueno, ahora, si quisiéramos por ejemplo recuperar una lista, en este caso de varios personajes, deberíamos usar el método **fetchAll()**. Este método a diferencia de **fetch()** no nos devolverá un array asociativo sino que nos devolverá un array de arrays asociativos.

Ahora insertaremos varios registros a mano para poder probarlo rápidamente esto:

INSERT INTO personaje(nombre, descripcion)

VALUES

('Robin', 'El compañero de Batman'),

('El Guasón', 'El que se ríe'),

('Dos Caras', 'El que tiene dos caras'),

('El Acertijo', 'El de verde con un signo de preguntas'),

('Alfred', 'El mayordomo'),

('Bane', 'El que usa máscara');

El nuevo método para recuperar todos los personajes que se llamará **recuperarTodos()**. La clase ahora quedará así:

<?php

require\_once 'Conexion.php';

class Personaje {

private $id;

private $nombre;

private $descripcion;

const TABLA = 'personaje';

public function getId() {

return $this->id;

}

public function getNombre() {

return $this->nombre;

}

public function getDescripcion() {

return $this->descripcion;

}

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;

}

public function guardar(){

$conexion = new Conexion();

if($this->id) /\*Modifica\*/ {

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

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

$consulta->bindParam(':descripcion', $this->descripcion);

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

$consulta->execute();

}else /\*Inserta\*/ {

$consulta = $conexion->prepare('INSERT INTO ' . self::TABLA .' (nombre, descripcion) VALUES(:nombre, :descripcion)');

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

$consulta->bindParam(':descripcion', $this->descripcion);

$consulta->execute();

$this->id = $conexion->lastInsertId();

}

$conexion = null;

}

public static function buscarPorId($id){

$conexion = new Conexion();

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

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

$consulta->execute();

$registro = $consulta->fetch();

if($registro){

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

}else{

return false;

}

}

public static function recuperarTodos(){

$conexion = new Conexion();

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

$consulta->execute();

$registros = $consulta->fetchAll();

return $registros;

}

}

?>

Analicemos el código del método **recuperarTodos()**:

Conectamos a la base de datos:

$conexion = new Conexion();

Creamos la consulta:

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

La ejecutamos:

$consulta->execute();

Recuperamos la respuesta de la consulta y la retornamos:

$registros = $consulta->fetchAll();

return $registros;

Ahora vemos cómo funciona:

<?php

require\_once 'clases/Personaje.php';

$personajes = Personaje::recuperarTodos();

?>

<html>

<head></head>

<body>

<ul>

<?php foreach($personajes as $item): ?>

<li> <?php echo $item['nombre'] . ' - ' . $item['descripcion']; ?> </li>

<?php endforeach; ?>

</ul>

</body>

</html>

Simplemente guardamos en la variable **$personajes** el resultado de la consulta y la mostramos en pantalla con código html.