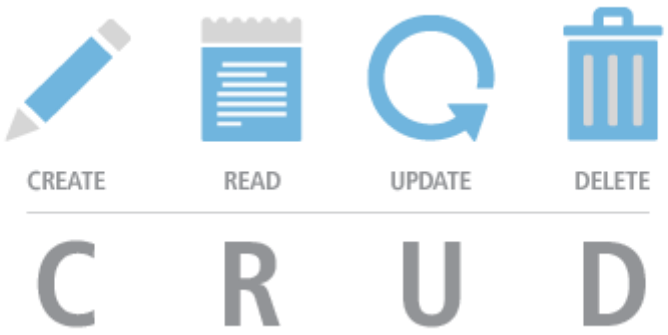
Creando CRUD en PHP: operaciones básicas para la gestión de bases de datos



Cualquier portal web utiliza algún sistema de base de datos para almacenar la información que luego mostrará en sus páginas. Para ello es necesario ejecutar una serie de acciones básicas para su buen funcionamiento, que quedan referenciadas bajo el acrónimo de CRUD: Create, Read, Update y Delete.

Para la creación de nuestro CRUD utilizaremos PDO, una capa abstracta para PHP que nos permite realizar consultas a bases de datos y que soporta una gran variedad de motores diferentes como MySQL, SQLServer, Oracle…

Crear tabla base de datos

Lo primero de todo será crear la tabla de la base de datos que vamos a gestionar con nuestro CRUD. En nuestro caso usaremos como ejemplo una tabla donde almacenaremos información sobre vehículos como puede ser la marca, el modelo o el número de kilómetros que tiene. Las instrucciones para crear esta tabla son las siguientes:

```
CREATE TABLE `vehiculo` (
`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`marca` VARCHAR( 255 ) NOT NULL ,
`modelo` VARCHAR( 255 ) NOT NULL ,
`kilometros` INT NOT NULL
) ENGINE = INNODB;
```

Una vez creada la tabla también crearemos nuestra entidad "**vehiculo**", que será la data a mapear ya sea para listar, registrar, eliminar o modificar sobre la base de datos. En este caso el código PHP que utilizaremos será:

```
class Vehiculo
{
    private $id;
    private $marca;
    private $modelo;
    private $kilometros;

    public function __GET($k){ return $this->$k; }
    public function __SET($k, $v){ return $this->$k = $v; }
}
```

El método **__GET($k)** se encarga de devolver el valor del atributo indicado. En cambio, el método **__SET($k, $v)** asignará el valor $v al atributo indicado en el parámetro $k.

Clase de la lógica de negocio

Una vez completados estos pasos será el momento de empezar con la lógica de negocio, que nos permitirá realizar las acciones sobre nuestra base de datos. Esta clase estará formada por varios métodos, cada uno realiza una acción del CRUD. Veamos a continuación cada uno de los métodos que forman parte de esta clase de negocio.

Constructor

El constructor se encargará de crear el objeto PDO que luego utilizaremos para ejecutar nuestras instrucciones en la base de datos. El código para esto es:

```
public function __CONSTRUCT()
{
    try
    {
        $this->pdo = new PDO('mysql:host=localhost;dbname=vehiculo', 'root',
'');
        $this->pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    }
    catch(Exception $e)
    {
        die($e->getMessage());
    }
}
```

En el código anterior lo más reseñable es la siguiente línea:

```
$this->pdo = new PDO('mysql:host=localhost;dbname=vehiculo', 'root', 'xxxxxx');
```

Aquí lo que debemos hacer es indicar los datos que utilizaremos para conectarnos a la base de datos. Estos datos son:

- **host**: nombre del servidor en el que está ubicada la base de datos. Normalmente suele ser "localhost".
- **dbname**: Nombre de la base de datos. En nuestro ejemplo la hemos llamado "vehiculo".
- **dbuser**: Se trata del nombre del usuario que tendrá acceso a la base de datos. En nuestro caso el usuario es "root".
- **dbpass**: Sería la contraseña asociada al usuario de base de datos. En nuestro ejemplo le hemos puesto "xxxxxx".

Función listar resultados

Una vez que hemos visto el constructor podemos empezar a analizar cada una de las operaciones que forman parte del CRUD. La primera de ellas será la que nos permita sacar un listado de los registros guardados en base de datos. El código encargado de esto es el siguiente:

```
public function Listar()
{
    try
    {
        $result = array();

        $stm = $this->pdo->prepare("SELECT * FROM vehiculo");
        $stm->execute();

        foreach($stm->fetchAll(PDO::FETCH_OBJ) as $r)
        {
            $alm = new Vehiculo();

            $alm->__SET('id', $r->id);
            $alm->__SET('marca', $r->marca);
            $alm->__SET('modelo', $r->modelo);
            $alm->__SET('kilometros', $r->kilometros);

            $result[] = $alm;
        }

        return $result;
    }
    catch(Exception $e)
    {
        die($e->getMessage());
    }
}
```

Lo primero que haremos será crearnos un array de nombre $result donde almacenaremos los diferentes vehículos que forman parte de nuestra base de datos. Esto se hace mediante la siguiente instrucción:

```
$result = array();
```

A continuación indicamos la instrucción que queremos ejecutar sobre la base de datos:

```
$stm = $this->pdo->prepare("SELECT * FROM vehiculo");
$stm->execute();
```

En este caso hemos indicado mediante la instrucción SQL "**SELECT * FROM vehiculo**" que nos devuelva todos los registros almacenados en la base de datos. El encargado de ejecutar esta instrucción SQL será el método "execute()".

Una vez ejecutada la instrucción tendremos que recorrer todos esos registros para crear los diferentes vehículos, a fin de almacenarlos en nuestro array $result.

```
foreach($stm->fetchAll(PDO::FETCH_OBJ) as $r)
{
    $alm = new Vehiculo();
    $alm->__SET('id', $r->id);
    $alm->__SET('marca', $r->marca);
    $alm->__SET('modelo', $r->modelo);
    $alm->__SET('kilometros', $r->kilometros);

    $result[] = $alm;
 }
```

Por último, sólo falta devolver el array que contiene la información para poder mostrarla de la forma que queramos en nuestra página web.

```
return $result;
```

Función recuperar un vehículo determinado

La siguiente operación que vamos a mostrar es aquella que permite recuperar un único resultado de la base de datos. El funcionamiento es muy similar al caso anterior, pero con la diferencia de que en la consulta SQL se le indicará el id del vehículo que queremos recuperar:

```
public function recuperarVehiculo($id)
{
    try
    {
        $stm = $this->pdo
                  ->prepare("SELECT * FROM vehiculo WHERE id = ?");
        $stm->execute(array($id));
        $r = $stm->fetch(PDO::FETCH_OBJ);

        $alm = new Vehiculo();

        $alm->__SET('id', $r->id);
        $alm->__SET('marca', $r->Nombre);
        $alm->__SET('modelo', $r->Apellido);
        $alm->__SET('kilometros', $r->Sexo);

        return $alm;
    } catch (Exception $e)
    {
        die($e->getMessage());
    }
}
```

Debemos fijarnos en que a este método se le pasa como argumento el id del vehículo que queremos recuperar de la base de datos.

```
public function recuperarVehiculo($id)
```

Como hemos dicho anteriormente, lo que se hace en este caso es indicar en la instrucción SQL que se quiere recuperar un vehículo en concreto identificado mediante su "**id**". Una vez ejecutada la sentencia, se almacena en un objeto $r donde después podremos sacar los valores de cada uno de los atributos del objeto vehículo.

```
$stm = $this->pdo->prepare("SELECT * FROM vehiculo WHERE id = ?");
$stm->execute(array($id));
$r = $stm->fetch(PDO::FETCH_OBJ);
```

Después, creamos un objeto vehículo vacío donde colocaremos la información recuperada de la base de datos.

```
$alm = new Vehiculo();
$alm->__SET('id', $r->id);
$alm->__SET('marca', $r->marca);
$alm->__SET('modelo', $r->modelo);
$alm->__SET('kilometros', $r->kilometros);
```

Por último, devolvemos el objeto vehículo que hemos creado y donde hemos almacenado la información recuperada de la base de datos.

```
return $alm;
```

Este objeto podrá ser utilizado en nuestra página web para mostrar los datos de ese vehículo.

Función guardar

Vamos a seguir ahora con la función que nos permite guardar nuevos vehículos en nuestra base de datos. El código sería el siguiente.

```
public function Registrar(Vehiculo $data)
{
    try
    {
    $sql = "INSERT INTO vehiculo  (marca,modelo,kilometros)
            VALUES (?, ?, ?)";

    $this->pdo->prepare($sql)
         ->execute(
        array(
            $data->__GET('marca'),
            $data->__GET('modelo'),
            $data->__GET('kilometros')
            )
        );
    } catch (Exception $e)
    {
        die($e->getMessage());
    }
}
```

Al igual que en el caso anterior, a esta función se le pasa como parámetro un objeto vehículo que contiene la información que vamos almacenar en base de datos. Esta información la hemos podido sacar de un formulario, un archivo xml o incluso otra base de datos.

```
public function Registrar(Vehiculo $data)
```

Lo siguiente será crear la instrucción SQL que insertará la información en la base de datos y que corresponde al siguiente trozo de código.

```
$sql = "INSERT INTO vehiculo  (marca,modelo,kilometros) VALUES (?, ?, ?)";

    $this->pdo->prepare($sql)
         ->execute(
        array(
            $data->__GET('marca'),
            $data->__GET('modelo'),
            $data->__GET('kilometros')
            )
        );
```

En el código anterior se observa que se hace uso del método "**__GET()**" para conseguir la información de cada uno de los atributos del objeto vehículo que se le ha pasado por parámetro.

Función modificar

La función modificar es muy similar al caso anterior donde hemos visto el proceso de registro.

```
public function modificar(vehiculo $data)
{
    try
    {
        $sql = "UPDATE vehiculo SET
                    marca          = ?,
                    modelo        = ?,
                    kilometros             = ?
                WHERE id = ?";

        $this->pdo->prepare($sql)
             ->execute(
            array(
                $data->__GET('marca'),
                $data->__GET('modelo'),
                $data->__GET('kilometros'),
                $data->__GET('id')
                )
            );
    } catch (Exception $e)
    {
        die($e->getMessage());
    }
}
```

Como en el anterior, al método se le pasaría como argumento un objeto vehículo con los datos a modificar.

```
public function modificar(vehiculo $data)
```

A continuación, se crearía la instrucción SQL que se encargaría de modificar la información, de forma similar a lo visto en el caso anterior.

```
$sql = "UPDATE vehiculo SET
                    marca = ?,
                    modelo = ?,
                    kilometros = ?
                WHERE id = ?";

        $this->pdo->prepare($sql)
             ->execute(
            array(
                $data->__GET('marca'),
                $data->__GET('modelo'),
                $data->__GET('kilometros'),
                $data->__GET('id')
                )
            );
```

Función eliminar

La última de las funciones sería la encargada de eliminar los registros de la base de datos. El código sería el siguiente:

```
public function eliminar($id)
{
    try
    {
        $stm = $this->pdo
                  ->prepare("DELETE FROM vehiculo WHERE id = ?");

        $stm->execute(array($id));
    } catch (Exception $e)
    {
        die($e->getMessage());
    }
}
```

Lo que se pasa por parámetros a la función es el id del vehículo que vamos a eliminar.

```
public function eliminar($id)
```

Ese id, será el que utilicemos para crear la consulta SQL encargada de realizar el borrado del registro.

```
$stm = $this->pdo->prepare("DELETE FROM vehiculo WHERE id = ?");
$stm->execute(array($id));
```

Al igual que en todos los casos anteriores, el método "**execute**" será el encargado de ejecutar la instrucción SQL.

Código completo de nuestro CRUD

A continuación, os dejamos el código completo de la clase que contendrá todas las funciones del CRUD que hemos explicado.

```
class VehiculoModel
{
    private $pdo;

    public function __CONSTRUCT()
    {
        try
        {
            $this->pdo = new PDO('mysql:host=localhost;dbname=vehiculo', 'root',
'xxxxxx');
            $this->pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        }
        catch(Exception $e)
        {
            die($e->getMessage());
        }
    }

    public function Listar()
    {
        try
        {
            $result = array();

            $stm = $this->pdo->prepare("SELECT * FROM vehiculo");
            $stm->execute();

            foreach($stm->fetchAll(PDO::FETCH_OBJ) as $r)
            {
                $alm = new Vehiculo();

                $alm->__SET('id', $r->id);
                $alm->__SET('marca', $r->marca);
                $alm->__SET('modelo', $r->modelo);
                $alm->__SET('kilometros', $r->kilometros);

                $result[] = $alm;
            }

            return $result;
        }
        catch(Exception $e)
        {
            die($e->getMessage());
        }
    }

    public function recuperarVehiculo($id)
    {
        try
        {
            $stm = $this->pdo
                      ->prepare("SELECT * FROM vehiculo WHERE id = ?");
```

```
            $stm->execute(array($id));
            $r = $stm->fetch(PDO::FETCH_OBJ);

            $alm = new Vehiculo();

            $alm->__SET('id', $r->id);
            $alm->__SET('marca', $r->marca);
            $alm->__SET('modelo', $r->modelo);
            $alm->__SET('kilometros', $r->kilometros);

            return $alm;
        } catch (Exception $e)
        {
            die($e->getMessage());
        }
    }

    public function eliminar($id)
    {
        try
        {
            $stm = $this->pdo
                      ->prepare("DELETE FROM vehiculo WHERE id = ?");

            $stm->execute(array($id));
        } catch (Exception $e)
        {
            die($e->getMessage());
        }
    }

    public function modificar(Vehiculo $data)
    {
        try
        {
            $sql = "UPDATE vehiculo SET
                        marca          = ?,
                        modelo         = ?,
                        kilometros             = ?
                    WHERE id = ?";

            $this->pdo->prepare($sql)
                 ->execute(
                array(
                    $data->__GET('marca'),
                    $data->__GET('modelo'),
                    $data->__GET('kilometros'),
                    $data->__GET('id')
                    )
                );
        } catch (Exception $e)
        {
            die($e->getMessage());
        }
    }

    public function Registrar(Vehiculo $data)
    {
        try
        {
```

```
        $sql = "INSERT INTO vehiculo (marca,modelo,kilometros)
                VALUES (?, ?, ?)";

        $this->pdo->prepare($sql)
             ->execute(
            array(
                $data->__GET('marca'),
                $data->__GET('modelo'),
                $data->__GET('kilometros')
                )
            );
        } catch (Exception $e)
        {
            die($e->getMessage());
        }
    }
}
```

Estas mismas operaciones que hemos creado para nuestra tabla vehículo, la podemos utilizar para cualquier otra, modificando el nombre de los campos. El pasó final, sería utilizar algo de código HTML para crear la estructura que permitiese elegir entre las diferentes acciones que hemos explicado a lo largo de nuestro White Paper.