

## **UNIDAD DIDÁCTICA 8:**

### **Sistema MVC.**

**Módulo profesional:**  
**Desarrollo Web en Entorno Servidor**

## Contenido

Resumen Introductorio .....	3
Introducción.....	3
Caso Introductorio .....	3
1. Patrón MVC .....	4
1.1. Porqué utilizar el Patrón MVC .....	4
1.2. Patrón MVC.....	5
1.3. Modelo .....	7
1.4. Vista .....	7
1.5. Controlador.....	8
1.6. Patrón MVC. Ejemplo de Modelo con PHP .....	9
2. Vista y Formularios.....	11
2.1. Primer paso. Estructura de ficheros modelo, vista de resultados	13
2.2. Segundo paso. Formularios .....	16
3. CRUD.....	18
3.1. Primer paso. Modificación del modelo para la incorporación de la inserción.....	18
3.2. Segundo paso. Incluimos los formularios necesarios para enviar la información .....	20
3.3. Tercer paso. Update y delete.....	21
4. Funciones, POO y MVC .....	24
4.1. Parámetros por defecto .....	24
4.2. MVC y funciones .....	26
5. Herencia y MVC.....	29
5.1. Accediendo a métodos padre.....	32
5.2. Aplicándolo al Modelo .....	33
Resumen final: .....	37

## Resumen Introductorio

En esta unidad aprenderemos los conceptos básicos del Patrón MVC, y del acrónimo CRUD, bases para los frameworks de desarrollo modernos y de desarrollo en equipo.

## Introducción

La programación independiente, la programación procedimental, la programación tradicional basada en fichero hoy en día deja de tener sentido en un momento en el que las aplicaciones cada vez son más grandes, más dependientes del trabajo de más personas y en general necesitadas de manejar mucha información.

PHP desde su desarrollo hacia a un mundo más orientado a objetos y por lo tanto orientado hacia una programación más mantenible y escalable no es menos en introducirse en el mundo del desarrollo mediante Frameworks como Symfony y Laravel.

Con la orientación a objetos damos un salto hacia modelos, arquitecturas y patrones de desarrollo como es el patrón MVC, Modelo-Vista-Controlador, que nos proporciona un grado importante de mejora en la organización de nuestro código, una mejora en la separación de las funciones de las clases y por lo tanto una impresionante mejora en las posibilidades de escalabilidad y mantenimiento de nuestras aplicaciones.

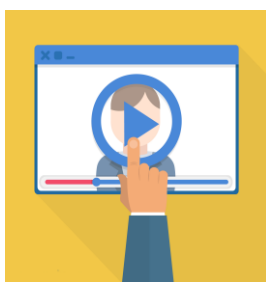
## Caso Introductorio

Queremos realizar una aplicación que maneje diversas tablas dentro de una base de datos. Interactuar con el usuario y mantener la información mediante diversas pantallas de entrada y listado de la información.

# 1. Patrón MVC

## 1.1. Porqué utilizar el Patrón MVC

¿Por qué debemos programar orientado a objetos? ¿Por qué debemos utilizar un patrón tipo Modelo Vista Controlador (MVC)? Creo que es importante contestar a estas preguntas antes de comenzar a utilizar un patrón y un cambio de metodología de programación como el MVC ya que en un principio resulta más aparatoso o costoso utilizarlo. Y qué mejor que utilizar un ejemplo para poder ver el por qué, para ello utilizaremos el código que os presento en github y una base de datos que también os proporciono:



### VIDEO DE INTERÉS

En el vídeo encontrarás una explicación de porqué utilizar el patrón MVC.



<https://youtu.be/jhw1zCec6VI>



### ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/gtVwKq>

¿Qué ha ocurrido en el ejemplo que hemos visto? Si desarrollamos una ÚNICA página que se conecta a una ÚNICA tabla dentro de la base datos, de una ÚNICA base de datos, seguramente no tendrá ningún sentido ni la utilización del patrón MVC ni la utilización de objetos.

Pero actualmente el desarrollo de software ha crecido y se ha hecho muy muy complejo, y una aplicación no se basa en una sola página, ni en 10 ni en 20, tampoco se basa en una base de datos de 2 o 3 tablas, y por último tampoco nos encontramos habitualmente.

El patrón MVC da respuesta a las problemáticas planteadas:

- Reutilización de código de una forma ordenada y clara.
- Trabajo en equipo de trabajo con la división de tareas.
- Mantenimiento y escalabilidad del código.
- Eliminación de dependencias fuertes entre código.

## 1.2. Patrón MVC

Muchas y variadas son las definiciones que podemos encontrar al respecto de la arquitectura MVC. Según (Fowler, 2002), el *Model View Controller* (MVC) es uno de los patrones más citados y utilizados alrededor del mundo. Comenzó como un marco desarrollado por Trygve Reenskaug para la plataforma *Smalltalk* a finales de los años setenta. Desde entonces, ha desempeñado un papel influyente en la mayoría de los marcos de la interfaz de usuario y en el pensamiento sobre el diseño de la interfaz de usuario.

El MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

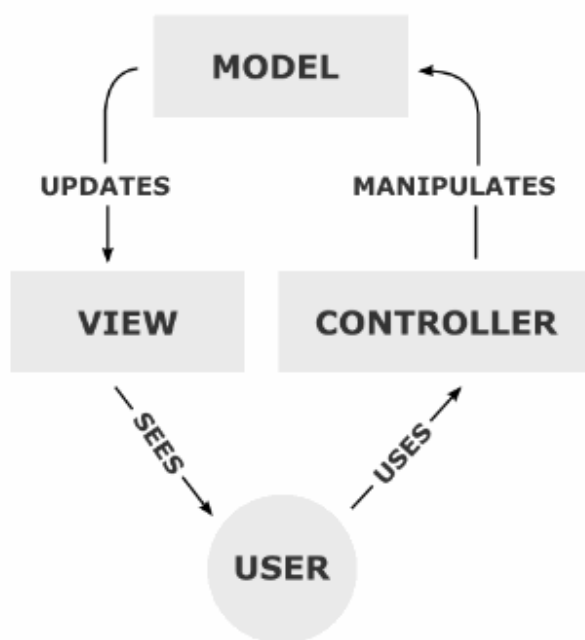
## BIBLIOGRAFÍA RECOMENDADA

***Libro recomendado, Patterns of Enterprise Application Architecture*** (Fowler, 2002)



*En el MVC, tenemos dos separaciones principales: separar la presentación del modelo y separar el controlador de la vista. De estos la separación de la presentación y el modelo es uno de los principios de diseño más importantes en el software, y la única vez que no debe seguirlo es en sistemas muy simples, donde el modelo no tiene ningún comportamiento real de todos modos. Tan pronto como obtenga alguna lógica no visual debe aplicar la separación. Por desgracia, muchos de los marcos de la interfaz de usuario hacen que sea difícil, y los que no lo son a menudo se enseña sin una separación.*

*La separación de la vista y el controlador es menos importante, por lo que sólo recomendaría hacerlo cuando sea realmente útil.*



**Imagen 1: Patrón MVC.** Fuente de la imagen: propia.

### 1.3. Modelo

Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

En esta parte nos encontraremos principalmente todo el desarrollo que interactúe con la base de datos.

### 1.4. Vista

Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

En esta parte nos encontraremos con toda la parte que tenga que ver con visualización y "pintar" la información.

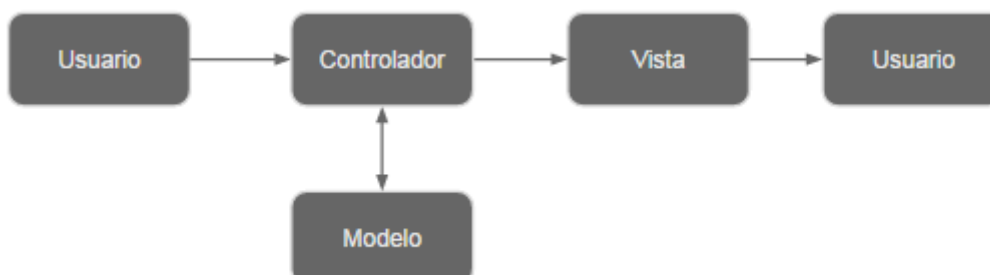
## 1.5. Controlador

Es quizá la parte más compleja tanto de entender como de programar, ya que es la parte que más complicada se hace abstraer. En un principio, podríamos pensar que únicamente realiza las labores de enrutador, es decir, recibe las peticiones de los usuarios y redirige hacia donde corresponda:



**Imagen 2: Controlador como enrutador.** Fuente de la imagen: propia.

Y aunque es una de las funciones del controlador, no es la única función. Siguiendo con el anterior esquema, lo que podríamos ampliar como funciones del controlador sería:



**Imagen 3: Controlador.** Fuente de la imagen: propia.

Esta ya es una aproximación más certera de la funcionalidad de un controlador:

- El controlador es quien recibe la petición del usuario.
- Será quien sepa qué información se necesita y por lo tanto de la necesidad o no de un Modelo.
- Por último, enrutará la información a través de una vista que le llegará al usuario.



## 1.6. Patrón MVC. Ejemplo de Modelo con PHP

Como siempre, un ejemplo vale mucho más que mil explicaciones teóricas, y para ello a partir del primer código realizado, modificaremos los tres ficheros y crearemos una clase que será el núcleo del cambio ya que nos permitirá esa distinción entre la Vista y el Modelo.



### VIDEO DE INTERÉS

En el vídeo encontrarás un ejemplo de creación de modelo siguiendo el patrón MVC.



<https://youtu.be/CNws4qBji1M>



### ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/AMEDf5>

Poco a poco iremos mejorando la estructura de ficheros, pero ya en un primer momento lo que observamos es que:

- Abstraemos todas las operaciones que realizábamos en múltiples ficheros contra nuestras bases de datos en un único fichero.
- Es mucho más sencillo la reutilización ya que únicamente deberemos incluir nuestra nueva clase.
- Es mucho más mantenible, ya que realizar aplicaciones se realizarán en un único fichero.

### COMPRUEBA LO QUE SABES




Una vez estudiado qué es una arquitectura MVC, ¿serías capaz de encontrar 3 frameworks escritos con PHP que utilicen arquitectura MVC?

Coméntalo en el foro.

### ARTÍCULO DE INTERÉS



Interesantísimo artículo sobre diferentes frameworks PHP.

 <https://www.genbetadev.com/frameworks/y-6-frameworks-php-mas-que-te-haran-la-vida-mas-simple>

### COMPRUEBA LO QUE SABES



Una vez estudiado qué es una arquitectura MVC ¿serías capaz utilizar el servidor PostgreSQL (<https://www.postgresql.org/>)?

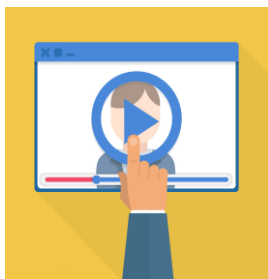
Coméntalo en el foro.

## 2. Vista y Formularios

Hemos aprendido a trabajar con el patrón Modelo, Vista y Controlador, y en concreto hemos comenzado con la parte de Modelo para poder encapsular y después generalizar nuestro objeto para posteriores páginas.

También aprendimos en una UD anterior, Formularios y objetos, cómo trabajar con formularios para poder interactuar con el usuario y así recoger información de forma dinámica.

Ahora pretendemos unir ambos aspectos, el de patrón MVC y el de Formularios y objetos para poder avanzar en el desarrollo de nuestras páginas. En este caso estamos hablando de la **VISTA**.



### VIDEO DE INTERÉS

En el vídeo encontrarás el paso a paso de la creación de la vista y la incorporación de los formularios.



<https://youtu.be/dcbX44yzSiE>

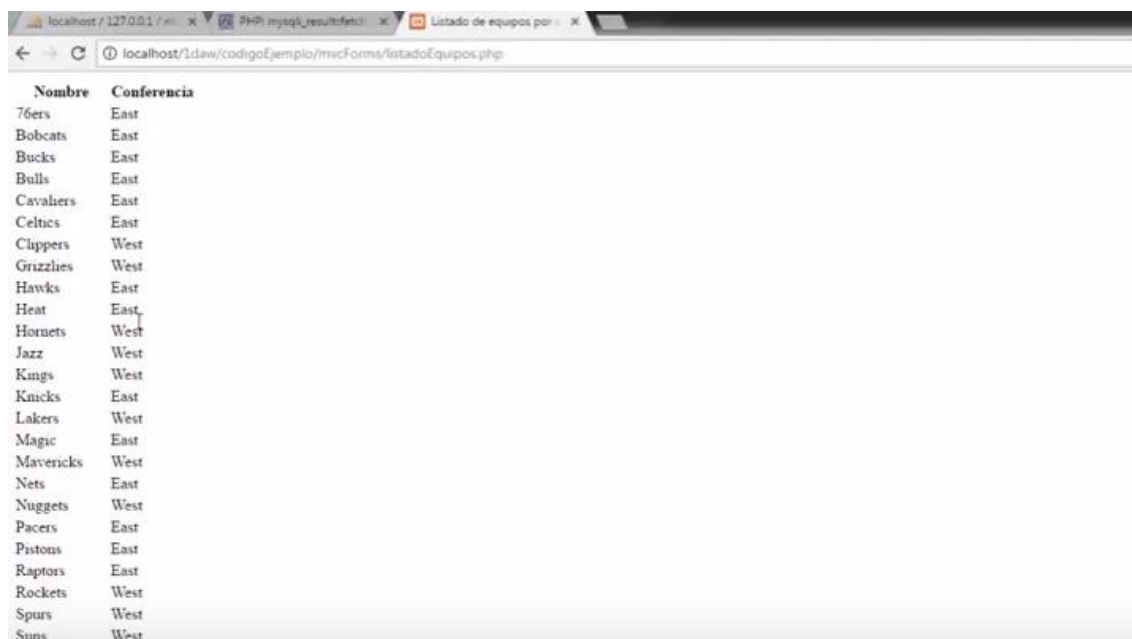


### ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/MzaAzi>



Nombre	Conferencia
76ers	East
Bobcats	East
Bucks	East
Bulls	East
Cavaliers	East
Celtics	East
Clippers	West
Grizzlies	West
Hawks	East
Heat	East
Hornets	West
Jazz	West
Kings	West
Knicks	East
Lakers	West
Magic	East
Mavericks	West
Nets	East
Nuggets	West
Pacers	East
Pistons	East
Raptors	East
Rockets	West
Spurs	West
Suns	West

**Imagen 5: Resultado de ejecución.** Fuente de la imagen: propia.

## 2.1. Primer paso. Estructura de ficheros modelo, vista de resultados

Usando como referencia el vídeo y el código propuesto, vemos que en primer lugar generaremos en la parte de modelo para interactuar con la base de datos.

### Modelo

```
<?php
/**
 * Permitir la conexion contra la base de datos
 */
class dbNBA
{
    //Atributos necesarios para la conexion
    private $host="localhost";
    private $user="root";
    private $pass="";
    private $db_name="nba";

    //Conector
    private $conexion;

    //Propiedades para controlar errores
    private $error=false;

    function __construct()
    {
        $this->conexion = new mysqli($this->host, $this->user, $this->pass, $this->db_name);
        if ($this->conexion->connect_errno) {
            $this->error=true;
        }
    }

    //Funcion para saber si hay error en la conexion
    function hayError() {
        return $this->error;
    }

    //Funcion para devolver los equipos y su conferencia
    public function devolverEquiposConf($conferencia) {
        if($this->error==false) {
            $resultado = $this->conexion->query("SELECT nombre,conferencia
FROM equipos WHERE conferencia='".$conferencia."'");
            //echo "SELECT nombre,conferencia FROM equipos WHERE
conferencia='".$conferencia."'";
            return $resultado;
        }else{
            return null;
        }
    }
}
```

Como podemos observar:

1. Tenemos todo lo necesario para la administración de la conexión contra la base de datos:
  - a. Los parámetros
  - b. El control de errores
2. En segundo lugar, tenemos todo el código necesario para poder interactuar con la base de datos como es la parte de consultas

Una vez tenemos este código podemos realizar la parte de VISTA, y en primer lugar implementaremos la visualización de los datos sacados por el listado contra la base de datos:

#### Vista

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Listado de equipos por conferencia</title>
  </head>
  <body>
    <!-- Esqueleto de info-->
    <?php
      include "dbNBA.php";
      //Crear el objeto de conexion
      $nba=new dbNBA();
      //Comprobar que llega la variable conferencia
      if(isset($_POST["conferencia"])){
        //Recuperar los equipos y sus conferencias
        $resultado=$nba->devolverEquiposConf($_POST["conferencia"]);
        ?>
        <table>
          <tr>
            <th>Nombre</th>
            <th>Conferencia</th>
          </tr>
          <?php
            while ($fila = $resultado->fetch_assoc()) {
              echo "<tr>";
              echo "<td>".$fila["nombre"]."</td>";
              echo "<td>".$fila["conferencia"]."</td>";
              echo "</tr>";
            }
            ?>
          </table>
          <?php }?>
        <!-- Esqueleto de info-->
      </body>
    </html>
```

## EJEMPLO PRÁCTICO

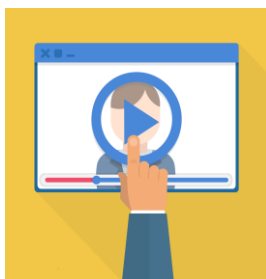
Una vez que hemos visto el inicio de la creación de la estructura mvc, vamos a crear un ejemplo sobre una base de datos de alumnos:

- 1) Para nuestro ejemplo necesitaremos una nueva base de datos:
  - a. Con la ayuda de Phpmyadmin crearemos una base de datos denominada escuela.
  - b. Crearemos una nueva tabla denominada alumnos con los siguientes campos:
    - i. Nombre, varchar 64
    - ii. Apellidos, varchar 128
    - iii. Edad, int
- 2) Crearemos una nueva carpeta para nuestro proyecto, y dentro de este:
  - a. listado.php
  - b. lib/escuela.php
- 3) Utilizando como modelo el ejemplo anterior llamado dbNBA, generaremos una nueva clase denominada Escuela.



## 2.2. Segundo paso. Formularios

¿Cómo incorporar interacción con el usuario? Los formularios son la clave en esto. Veámoslo con el vídeo explicativo:



### VIDEO DE INTERÉS

En el vídeo encontrarás la incorporación del formulario a nuestro patrón mvc.

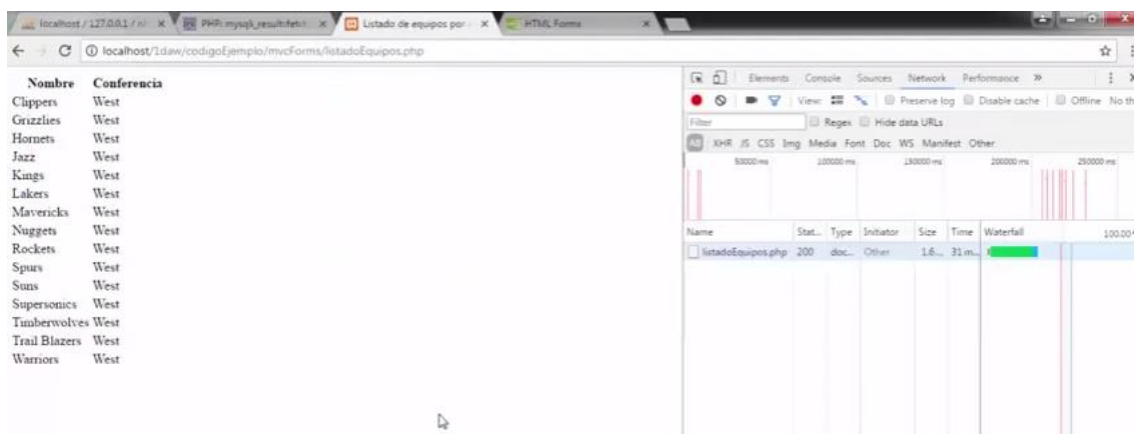
 <https://youtu.be/zzEWhHE0vXk>



### ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:

 <https://goo.gl/B1pnQ8>



**Imagen 6: Resultado de ejecución.** Fuente de la imagen: propia.



Tal y como se explica, hemos incorporado un formulario que recoge nuestras peticiones:

### Formulario

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Pagina principal</title>
  </head>
  <body>
    <h2>MOSTRAR LISTADO DE TODOS LOS EQUIPOS</h2>
    <form action="listadoEquipos.php" method="POST">
      Conferencia:<br>
      <input type="text" name="conferencia"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

### COMPRUEBA LO QUE SABES



Una vez estudiado como incluir una vista y formulario a partir de una tabla de una base de datos, ¿puedes crear un ejemplo propio que presente los datos de la forma vista?

Coméntalo en el foro.

Esto permitirá que nos llegue el parámetro elegido a nuestro listadoEquipos.php, lo que nos permitirá seleccionar la conferencia, y por lo tanto deberíamos cambiar el método tal y como el vídeo nos indica.

### 3. CRUD

El término CRUD proviene del acrónimo, CREATE, READ, UPDATE y DELETE, es decir, todas aquellas operaciones básicas y esenciales que podemos realizar contra una base de datos:

- Crear un nuevo registro con INSERT.
- Leer registros con SELECT.
- Actualizar un registro almacenado con UPDATE.
- Borrar un registro con DELETE.

Después de haber aprendido a realizar consultas con una base de datos, y comprender y crear un patrón MVC, es muy sencillo comprender y utilizar el resto de acciones. Pero lo mejor, igual que en otras ocasiones, será realizar ejemplos que nos ayuden a comprender cada uno de las acciones.

#### 3.1. Primer paso. Modificación del modelo para la incorporación de la inserción

La inserción de nuevos registros en una base de datos, después de la lectura de información, es la acción casi más importante para comenzar a interactuar con nuestras tablas. No resulta complejo ni complicado, pero será importante puesto que el proceso es un poco más largo realizarlo poco a poco y comprobando cada uno de los pasos:

##### VIDEO DE INTERÉS



En el vídeo encontrarás la explicación de la modificación del modelo para la incorporación del INSERT.



<https://youtu.be/dMqKVxvLz1I>



## ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:

 <https://goo.gl/wk3sg3>



**Imagen 7: Resultado de ejecución del INSERT.** Fuente de la imagen: propia.

En concreto, el código que incluimos nuevo en nuestro modelo es:

### INSERT

```
//function insercion contra la tabla usuarios
public function insertarUsuario($nombre,$apellidos,$edad){
    if($this->error==false)
    {
        $insert_sql="INSERT INTO usuario (id, nombre, apellidos, edad)
VALUES (NULL,'".$nombre."', '>".$apellidos."', '>".$edad."");
        if (!$this->conexion->query($insert_sql)) {
            echo "Falló la insercion de la tabla: (" . $this->conexion->errno . ") " . $this->conexion->error;
            return false;
        }
        return true;
    }else{
        return false;
    }
}
```

### 3.2. Segundo paso. Incluimos los formularios necesarios para enviar la información

El siguiente paso una vez que tenemos creada una función que nos permite insertar información en la base de datos consiste en generalizar la función y hacerla más flexible. Para ello en primer lugar añadiremos los parámetros necesarios en la función para recibir los datos y en segundo lugar incluir un formulario que nos permita enviar la información (OJO, aún no realizaremos comprobación de la información):



#### VIDEO DE INTERÉS

En el vídeo encontrarás la explicación de la inclusión del formulario.



<https://youtu.be/NMonB4hVPs4>



#### ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/rTm2U1>

## EJEMPLO PRÁCTICO

Seguiremos con el ejemplo creado sobre la base de datos de alumnos:



- 1) Usaremos la base de datos alumnos
- 2) En escuela.php incluiremos una nueva función que realice la inserción de un nuevo alumno en la tabla alumnos.
- 3) Crearemos un nuevo documento denominado alumno.php. En este fichero crearemos un formulario con 3 campos: nombre, apellidos y edad. Este formulario enviará la información a alumnoNuevo.php que recibirá la información y utilizará la nueva función para la inserción.

### 3.3. Tercer paso. Update y delete

La actualización de registros va a ser muy, muy parecida a la inserción. Aquí la gran dificultad es actualizar el registro correcto, ya que el resto de comandos y código es exactamente igual a los vistos:



#### VIDEO DE INTERÉS

En el vídeo encontrarás la explicación del update.



<https://youtu.be/XWLqGN5Y3II>



**Imagen 8: Resultado de ejecución.** Fuente de la imagen: propia.

En concreto, el código que incluimos nuevo en nuestro modelo es:

#### UPDATE

```
//function actualizar contra la tabla usuarios
public function actualizarUsuario($id,$nombre,$apellidos,$edad){
    if($this->error==false)
    {
        $insert_sql="UPDATE usuario SET nombre='".$nombre."',
apellidos='".$apellidos."', edad='".$edad.'" WHERE id='".$id;
        if (!$this->conexion->query($insert_sql)) {
            echo "Falló la actualizacion de la tabla: (" . $this-
>conexion->errno . ") " . $this->conexion->error;
            return false;
        }
        return true;
    }else{
        return false;
    }
}
```

Por último, el borrado de un determinado registro será muy parecido a la actualización, salvo que no es necesario todos los datos para realizar la consulta:

En concreto, el código que incluimos nuevo en nuestro modelo es:

#### DELETE

```
//function borrar contra la tabla usuarios
public function borrarUsuario($id){
    if($this->error==false)
    {
        $insert_sql="DELETE FROM usuario WHERE id=".$id;
        if (!$this->conexion->query($insert_sql)) {
            echo "Falló el borrado del usuario: (" . $this->conexion-
>errno . ") " . $this->conexion->error;
            return false;
        }
        return true;
    }else{
        return false;
    }
}
```

#### COMPRUEBA LO QUE SABES



Una vez estudiado el CRUD a partir de la información, ¿puedes crear un ejemplo propio que contenga un CRUD siguiendo el modelo estudiado?

Coméntalo en el foro.

## 4. Funciones, POO y MVC

### 4.1. Parámetros por defecto

Han sido ya muchas las entradas donde hemos hablado de funciones, su uso dentro de las páginas, su uso dentro de las clases, sus variantes, el ámbito y visibilidad, ... Cuando comenzamos a trabajar la conexión a base de datos embebida con la Programación orientada a objetos y la arquitectura MVC, se nos plantean situaciones donde necesitaríamos tener mayor flexibilidad en el uso de las funciones. Este es el punto en el que nos puede ayudar muchas reglas de php en este sentido.

#### RECUERDA...

##### Un método dentro de una clase tiene... (UD5)



- La palabra reservada *function*.
- El nombre de la función definida por el usuario (en este caso se denomina *foo*).
- Los argumentos de entrada de parámetros, los cuales pueden ser tantos como se necesiten o se requieran para el buen funcionamiento de la función.
- El cuerpo de la función.
- La devolución del resultado mediante la palabra reservada *return*.



Los parámetros por defecto en una función nos permiten definir funciones donde una, varias, o todos los parámetros de entrada puedan coger un parámetro por defecto cuando este no se defina. Vamos a verlo con el siguiente código y ejemplo:

#### Parámetros por defecto

```
<?php

function devolverUsuarioId($id,$ultimo=false) {
    if($this->error==false){
        if($ultimo==true) $resultado = $this->conexion->query("SELECT
* FROM usuario ORDER BY id DESC LIMIT 1");
        else $resultado = $this->conexion->query("SELECT * FROM
usuario WHERE id=".$id);
        return $resultado;
    }else{
        return null;
    }
}

//USOS
$resultado=$usuarios->devolverUsuarioId(4);
$resultado=$usuarios->devolverUsuarioId(null,true);

?>
```



#### VIDEO DE INTERÉS

En el vídeo encontrarás la explicación sobre los parámetros por defecto.



<https://youtu.be/JdJiIlubXBA>

### EJEMPLO PRÁCTICO



Seguiremos con el ejemplo creado sobre la base de datos de alumnos:

- 1) Usaremos la base de datos alumnos.
- 2) En escuela.php modificaremos nuestra función de inserción de datos, haciendo que el parámetro de entrada, edad, no sea obligatorio y que sea un parámetro con una edad 18 por defecto.

## 4.2. MVC y funciones

Hasta la fecha, nuestras funciones dentro de la clase devolvían objetos del tipo mysqli, algo un poco incoherente respecto a la arquitectura MVC, donde queremos que la parte de Modelo se quede en las clases y la parte de vista en el frontal. Para cambiar esta dinámica simplemente cambiaremos el resultado de nuestras funciones dentro de la definición de las clases.



### VIDEO DE INTERÉS

En el vídeo encontrarás la explicación de la inclusión del formulario.



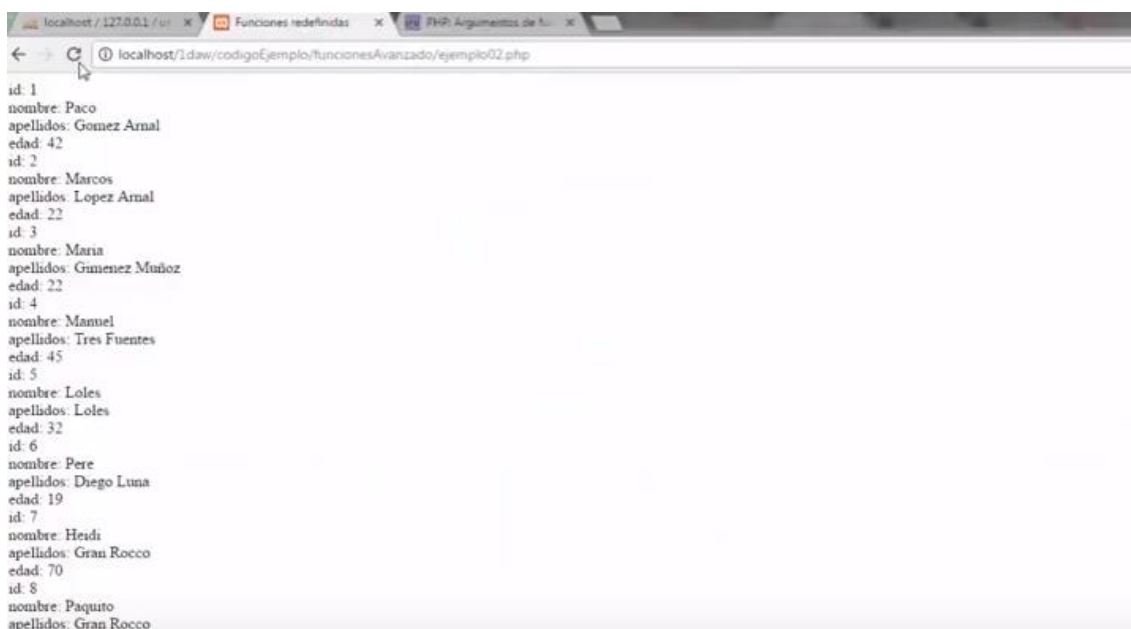
<https://youtu.be/4oBjcNmdM9w>



## ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:

 <https://goo.gl/6LEzgG>



```
id: 1
nombre: Paco
apellidos: Gomez Arnal
edad: 42
id: 2
nombre: Marcos
apellidos: Lopez Arnal
edad: 22
id: 3
nombre: Maria
apellidos: Gimenez Munoz
edad: 22
id: 4
nombre: Manuel
apellidos: Tres Fuentes
edad: 45
id: 5
nombre: Loles
apellidos: Loles
edad: 32
id: 6
nombre: Pere
apellidos: Diego Luna
edad: 19
id: 7
nombre: Heidi
apellidos: Gran Rocco
edad: 70
id: 8
nombre: Paquito
apellidos: Gran Rocco
```

**Imagen 9: Resultado de ejecución.** Fuente de la imagen: propia.

Cuando en la vista representábamos los datos de nuestra base de datos, utilizábamos el siguiente código:

Vista

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Listado de equipos por conferencia</title>
  </head>
  <body>
    <!-- Esqueleto de info-->
    <?php
      include "dbNBA.php";
      //Crear el objeto de conexion
      $nba=new dbNBA();
      //Comprobar que llega la variable conferencia
      if(isset($_POST["conferencia"])){
        //Recuperar los equipos y sus conferencias
        $resultado=$nba->devolverEquiposConf($_POST["conferencia"]);
      }
      <table>
        <tr>
          <th>Nombre</th>
          <th>Conferencia</th>
        </tr>
        <?php
          while ($fila = $resultado->fetch_assoc()) {
            echo "<tr>";
            echo "<td>".$fila["nombre"]."</td>";
            echo "<td>".$fila["conferencia"]."</td>";
            echo "</tr>";
          }
        <?>
      </table>
    <?php }?>
    <!-- Esqueleto de info-->
  </body>
</html>

```

Como vemos, utilizamos la función `fetch_assoc` de la librería `mysqli`. Esto provoca un "acople" y una gran dependencia de la base de datos que estamos utilizando.

Justamente lo que estamos intentando es producir un código en la vista lo más independiente del modelo, de la base de datos posible. Por este motivo es muy interesante “desacoplar”, y no hacerlo dependiente mediante la devolución de un array:

#### Devolviendo un array

```
<?php
//Devolver arrays
function devolverUsuarios() {
    $tabla=[];
    if($this->error==false){
        $resultado = $this->conexion->query("SELECT * FROM usuario");
        while($fila=$resultado->fetch_assoc()){
            $tabla[]=$fila;
        }
        return $tabla;
    }else{
        return null;
    }
}
?>
```

## 5. Herencia y MVC

Una de las propiedades más importantes y características de la programación orientada a objetos es la Herencia.

En POO, mediante la herencia, los diseñadores pueden crear nuevas clases partiendo de una clase o de una jerarquía de clases preexistente evitando con ello el rediseño, la modificación y verificación de la parte ya implementada. La herencia facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento (métodos) y eventualmente los atributos (variables) de su superclase.

## PARA SABER MÁS...

### *Herencia*



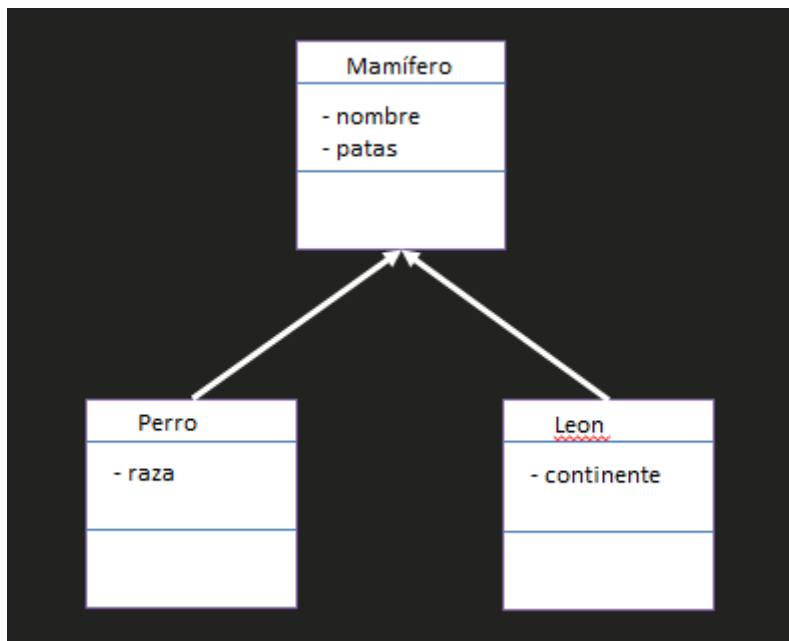
El fantástico libro gratuito sobre Java, **Thinking in Java**, 3rd ed. Revision 4.0, (Eckel, 2002), aunque verse sobre el lenguaje Java, nos proporciona una visión muy bien documentada de lo que es y significa la programación orientada a objetos.

Una de las propiedades más interesantes de la POO, la Herencia, la cual permite la reutilización del código.

Por sí mismo, la idea de un objeto es una herramienta conveniente. Nos permite empaquetar los datos y la funcionalidad por concepto. Estos conceptos se expresan como unidades fundamentales en el lenguaje de programación utilizando la palabra clave class (en el caso de PHP también).

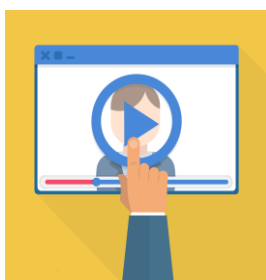
La herencia, como muy bien introducíamos, permite la reutilización de esa unidad fundamental, sus datos y sus funcionalidades, creando una estructura modular y arborescente de padres e hijos.

Utilizando una imagen que puede clarificar este concepto:



**Imagen 4: Herencia.** Fuente de la imagen: propia.

Son conceptos que en 1º y con otros lenguajes de programación como Java, están muy marcados. Pero no olvidemos que PHP pivotó hacia un modelo y por lo tanto orientado a objetos.



#### VIDEO DE INTERÉS

En el vídeo encontrarás la explicación la independización de código entre vista y modelo.

 <https://youtu.be/ct7pReCvz0g>



### ENLACE DE INTERÉS

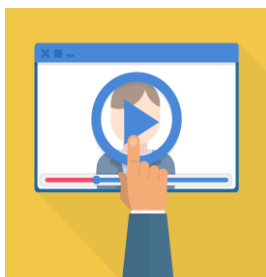
En el siguiente enlace tienes el código utilizado:

 <https://goo.gl/8X8tW6>

La identificación la realizamos con la palabra reservada `extends`, el resto, su funcionamiento y significado será el mismo que con cualquier otro lenguaje de programación.

## 5.1. Accediendo a métodos padre

Cuando realizamos una estructura de clases padre-hijo, una estructura con diferentes propiedades, métodos, constantes, ..., aparecen ocasiones donde se pueden redefinir los métodos (esto se denomina sobrecarga), ahora quedémonos con que un mismo método del padre se puede redefinir en el hijo, y sería imposible poder acceder al del padre a no ser que utilicemos el operador de ámbito:: (doble punto), seguido de la palabra reservada `parent`. Como siempre lo mejor es que lo veamos con un ejemplo:



### VIDEO DE INTERÉS

En el vídeo encontrarás la explicación sobre el acceso a los métodos padre.

 <https://youtu.be/370MMacFhzc>



## 5.2. Aplicándolo al Modelo

La herencia tiene una aplicación directa y muy fácil de entender en el modelo, ya que podemos realizar la siguiente división de las tareas:

- Una clase padre encargada de la conexión e interacción con la base de datos.
- Unas clases encargadas de la interacción con las tablas y el manejo de la información.

En esta tipología muy clásica dentro de los frameworks basados en MVC, las clases encargadas de la interacción con las tablas, normalmente tienen el mismo nombre que tienen las tablas con las que están relacionadas, mientras que la clase padre suele tener un nombre más generalista como db y esta clase es la encargada en última instancia de estar relacionada con la tecnología de base de datos elegida para nuestra aplicación.

## Modificando db

```
<?php
/**
 * Permitir la conexion contra la base de datos
 */
class db
{
    //Atributos necesarios para la conexion
    private $host="localhost";
    private $user="root";
    private $pass="";
    private $db_name="usuarios";
    //Conector
    private $conexion;
    //Propiedades para controlar errores
    private $error=false;
    private $error_msj="";
    function __construct()
    {
        $this->conexion = new mysqli($this->host, $this->user, $this->pass, $this->db_name);
        if ($this->conexion->connect_errno) {
            $this->error=true;
            $this->error_msj="No se ha podido realizar la conexion a la bd. Revisar base de datos o parámetros";
        }
    }
    //Funcion para saber si hay error en la conexion
    function hayError(){
        return $this->error;
    }
    //Funcion que devuelve mensaje de error
    function msjError(){
        return $this->error_msj;
    }
    //Metodo para la realización de consultas a la bd
    public function realizarConsulta($consulta){
        if($this->error==false){
            $resultado = $this->conexion->query($consulta);
            return $resultado;
        }else{
            $this->error_msj="Imposible realizar la consulta: ".$consulta;
            return null;
        }
    }
}
?>
```

Vemos en el anterior código dos nuevas modificaciones:

- No aparece código con las consultas específicas SQL, esto aparecerá en las clases hijas.
- Aparece una nueva función genérica `realizarConsulta($consulta)`, que es la encargada de realizar las consultas y comprobar si se puede realizar.

#### Clase hija

```
<?php
include "db.php";
/**
 *
 */
class Usuario extends db
{
    function __construct()
    {
        //De esta forma realizamos la conexion a la base de datos
        parent::__construct();
    }
    //Devolvemos todos los usuarios
    function devolverUsuarios(){
        //Construimos la consulta
        $sql="SELECT * from usuario";
        //Realizamos la consulta
        $resultado=$this->realizarConsulta($sql);
        if($resultado!=null){
            //Montamos la tabla de resultados
            $tabla=[];
            while($fila=$resultado->fetch_assoc()){
                $tabla[]=$fila;
            }
            return $tabla;
        }else{
            return null;
        }
    }
}
?>
```

Aquí aparece nuestras consultas SQL, ya que tiene sentido que sea en la clase que va a estar relacionada con la tabla en cuestión la que tenga la información de cómo manejar dicha información.

Por otro lado, necesita de la clase padre db, para poder realizar las conexiones a través del conector y de una conexión ya abierta.

### COMPRUEBA LO QUE SABES



Una vez estudiada la Herencia, ¿puedes modificar tu ejemplo anterior para que contenga al menos una clase padre y otra hija de acuerdo a los ejemplos presentados?

Coméntalo en el foro.

### EJEMPLO PRÁCTICO

Seguiremos con el ejemplo creado sobre la base de datos de alumnos:



- 1) Usaremos la base de datos alumnos.
- 2) Crearemos una nueva clase denominada Alumnos.php dentro del directorio lib/alumnos.php.
- 3) De acuerdo a lo aprendido con herencia y PHP, modificaremos nuestros ficheros para que:
  - a. Escuela.php solo tenga los métodos de conexión.
  - b. Alumnos.php contenga los métodos CRUD.

## Resumen final:

En esta importantísima unidad hemos descubierto y aprendido el funcionamiento principal de los *frameworks* de desarrollo modernos. Actualmente el desarrollo ha evolucionado, tanto en tecnología como metodología.

La POO y la metodología MVC, proporcionan esos mecanismos modernos de trabajo en equipo para el desarrollo de aplicaciones web, pero sobre todo mejoran unos aspectos importantísimo en el desarrollo software:

- Mejora la relación con el cliente, ya que se puede recibir un feedback continuo
- Mejora el diseño y desarrollo ya que permite modularizar las tareas
- Mejora el mantenimiento y mejoras del desarrollo ya que no es necesario un cambio de todo el desarrollo sino de determinadas partes

La metodología MVC, Modelo-Vista-Controlador, puede ser implementada de múltiples formas, la vista en esta unidad es una de ellas, y es el inicio para entender otras opciones desarrolladas o implementar el propio desarrollo a partir de las necesidades del producto.