



UNIDAD III

**MANEJO DE FORMULARIOS Y CONEXIÓN A UNA
BASES DE DATOS EN PHP CON MYSQL.**

MANEJO DE FORMULARIO

TRABAJANDO CON FORMULARIOS EN PHP

Los formularios son una parte esencial de cualquier aplicación web que permita a los usuarios interactuar y enviar información al servidor. Php es un lenguaje de programación ampliamente utilizado para procesar formularios en aplicaciones web.

MÉTODOS GET Y POST

En php, los formularios pueden enviar datos al servidor utilizando dos métodos principales: **GET** y **POST**. Ambos métodos son utilizados para transmitir información desde el navegador del cliente al servidor, pero tienen diferencias importantes.

GET

- ✓ Los datos se envían a través de la URL y son visibles en la barra de direcciones del navegador.
- ✓ Se utiliza principalmente para solicitudes que no modifican datos en el servidor, como búsquedas o filtros.
- ✓ La información se recupera en php utilizando la variable super-global **\$_GET**

POST

- ✓ Los datos se envían de manera más discreta y no son visibles en la URL.
- ✓ Se utiliza para enviar datos sensibles o para realizar cambios en el servidor, como enviar un formulario de registro.
- ✓ La información se recupera en php utilizando la variable super-global **\$_POST**

MANEJO DE FORMULARIO

TRABAJANDO CON FORMULARIOS EN PHP

Para crear un formulario que utilice uno de estos métodos, simplemente establece el atributo **method** del formulario en **GET** o **POST** según sea necesario, y en el atributo **action** ponemos la URL del archivo php que va a recibir y procesar los datos que envía el usuario.

RECUPERANDO INFORMACIÓN

Una vez que el formulario se ha enviado al servidor, es necesario recuperar la información ingresada por el usuario. Esto se hace utilizando las variables super-globales **\$_GET** y **\$_POST**, según el método que se haya utilizado.



MANEJO DE FORMULARIO

TRABAJANDO CON FORMULARIOS EN PHP

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <form action="./procesar.php" method="GET">
    <input type="text" placeholder="Usuario" name="user">
    <input type="password" placeholder="Contraseña" name="pass">
    <input type="submit" value="Enviar">
  </form>
</body>

</html>
```

```
<?php
$usuario = $_GET['user'];
$password = $_GET['pass'];

echo "Los datos recibidos son,
usuario: ".$usuario." y contraseña ".$password;
```

TRATAMIENTO DE CADENAS

CADENA DE TEXTO EN PHP

Una cadena de texto la componen una serie de caracteres (letras, números y/o símbolos) delimitados normalmente entre comillas dobles o simples.

CONVERTIR MAYÚSCULAS Y MINÚSCULAS

Para convertir una cadena de texto en php, usaremos las funciones **strtoupper()** para convertir una cadena a mayúscula y **strtolower()** para convertir una cadena a minúscula.

También disponemos de **ucfirst()** para obtener una cadena de texto con el *primer carácter en mayúscula*, y de **ucwords()** para **convertir a mayúsculas el primer carácter de cada palabra**. La función **lcfirst()** nos devuelve una cadena de texto con el **primer carácter en minúscula**.

```
$nombre = "Felisa nchama";  
//convirtiendo en minuscula  
echo strtolower($nombre)."<br/>";  
//convirtiendo en mayuscula  
echo strtoupper($nombre)."<br/>";  
//primer caracter en mayuscula  
echo ucfirst($nombre)."<br/>";  
//convierte a mayuscula a cada palabra de la cadena  
echo ucwords($nombre)."<br/>";  
//convierte a minuscula el primer caracter  
echo lcfirst($nombre)."<br/>";
```

TRATAMIENTO DE CADENAS

CADENA DE TEXTO EN PHP

OBTENER LA LONGITUD DE UNA CADENA

La función de php **strlen()** nos devuelve la longitud de una cadena de caracteres.

```
$nombre = "Felisa nchama";  
echo strlen($nombre);
```

CONCATENAR O UNIR CADENAS

Podemos concatenar (unir) cadenas de caracteres usando el **punto (.)** como operador de concatenación.

BUSCAR UNA CADENA DENTRO DE OTRA

Si necesitamos averiguar si una cadena de caracteres contiene parte de otra cadena usaremos la funciones de php **strpos()** (*distingue entre mayúsculas y minúsculas*) y **stripos()** (*no distingue entre mayúsculas y minúsculas*), que devolverán un valor numérico indicando la posición en la que ha sido hallada la cadena buscada dentro de la cadena de texto principal, o bien **false** si no se ha encontrado la coincidencia.

```
$cadena = "Aprendiendo a programar en PHP";  
$encontrar = strpos($cadena, "programar");  
echo $encontrar;
```

TRATAMIENTO DE CADENAS

CADENA DE TEXTO EN PHP

BUSCAR UNA CADENA DENTRO DE OTRA

Si necesitamos saber **cuántas veces aparece una cadena dentro de otra**, usaremos la función de php **substr_count()** teniendo en cuenta que distingue entre mayúsculas y minúsculas.

```
$cadena = "Aprendiendo a programar en PHP";  
$encontrar = substr_count($cadena,  
"programar");  
echo $encontrar;
```

EXTRAER PARTE DE UNA CADENA

Para extraer parte de una cadena en php usaremos la función de php **substr()**, indicando como primer parámetro la cadena de texto en la que se realizará la búsqueda, y como segundo y tercer parámetros las posiciones inicial y final de la subcadena a extraer.

```
$cadena = "Andrés";  
echo "[".substr($cadena, 3)."] <br/>";  
echo "[".substr($cadena, 0,1)."] <br/>";
```

TRATAMIENTO DE CADENAS

CADENA DE TEXTO EN PHP

ELIMINAR ESPACIOS EN BLANCO A PRINCIPIO Y FINAL DE UNA CADENA DE TEXTO

Para **quitar los espacios en blanco** al principio y al final de una cadena disponemos de las siguientes funciones de php.

- ✓ **trim()**: elimina los espacios tanto a principio como a final de una cadena de texto.
- ✓ **ltrim()**: quita los espacios al principio de la cadena.
- ✓ **rtrim()**: quita los espacios al final de la cadena.

```
<?php
$cadena1 = " Curso de PHP - ";
$cadena2 = " Curso de PHP";
$cadena3 = "Curso de PHP ";
echo "<pre>";
echo "\$cadena1 : [".trim($cadena1)."]<br />"; // Devuelve: "$cadena1 : [Curso de PHP -]"
echo "\$cadena2 : [".ltrim($cadena2)."]<br />"; // Devuelve: "$cadena2 : [Curso de PHP]"
echo "\$cadena3 : [".rtrim($cadena3)."]<br />"; // Devuelve: "$cadena3 : [Curso de PHP]"
echo "</pre>";
?>
```


TRATAMIENTO DE NUMEROS

COMPROBAR SI UN VALOR ES NUMÉRICO

Cuando necesitemos comprobar si un determinado valor es numérico usaremos las funciones de php `is_int()`, `is_float()` e `is_numeric()`.

```
$int = 4;  
$float = 4.5;  
echo is_int($int);  
echo is_float($float);
```

Convertir a número

Para convertir a tipo numérico disponemos de la función `intval()`.

```
$valor = "4";  
var_dump($valor);  
$convertido = intval($valor);  
var_dump($convertido);
```

TRATAMIENTO DE NUMEROS

REDONDEAR NÚMEROS

Utilizaremos las siguientes funciones para redondear números en php.

- ✓ **round()**: redondea un número decimal hacia arriba, indicando como segundo parámetro la **precisión del número** obteniendo (si no indicamos nada quedará sin decimales).
- ✓ **ceil()**: redondea un número decimal hacia arriba, devolviendo sólo la parte entera.
- ✓ **floor()**: redondea un número decimal hacia abajo, devolviendo sólo la parte entera.

TRATAMIENTO DE NÚMERO

REDONDEAR NÚMEROS

```
<?php
echo round("8.34", 1,). "<br />";
echo round("8.35", 1,). "<br />";
echo round("8.36", 1,). "<br />";
echo ceil("55.4"). "<br />";
echo ceil("55.5"). "<br />";
echo ceil("55.50"). "<br />";
echo ceil('55.6'). "<p />";
echo floor("55.4"). "<br />";
echo floor("55.5"). "<br />";
echo floor("55.50"). "<br />";
echo floor("55.6")

?>
```

VALIDACIÓN DE FORMULARIOS EN PHP

Tal y como hemos visto en los anteriores cursos, usaremos el lenguaje HTML para crear formularios y JavaScript para validarlos localmente en el ordenador del usuario antes de enviar los datos al servidor web.

Cuando el servidor web recibe los datos, estos deben ser validados nuevamente en él para asegurarnos de que han llegado todos y que son correctos.

Dado que php es un lenguaje de programación que se ejecuta en el servidor web podemos utilizarlo para realizar dicha validación.

Consideraciones sobre la validación de formularios

Cuando un usuario envía un formulario los datos contenidos en cada componente (definidos mediante las etiquetas HTML **<input>**, **<select>** y **<textarea>**) llegan al servidor web y son almacenados en un array asociativo (**\$_POST** o **\$_GET**, dependiendo de si el formulario está definido con el atributo HTML **method="post"** o **method="get"**): por cada componente del formulario se insertará un elemento en el array cuya **clave** será el atributo HTML **name** de dicho componente, y el valor el contenido en su atributo HTML **value**.

Sin embargo, debes tener en cuenta que tras enviarse el formulario no se guardará nada en dicho array asociativo si:

- Una **casilla de verificación** (creada con **<input type="checkbox">**) o un **botón radio** (creado con **<input type="radio">**) no estuviesen seleccionados.

Otras condiciones:

- Si un **cuadro de texto** (creada con **<input type="text">** o **<textarea>**) se ha enviado sin contenido alguno, se insertará un elemento en el array asociativo pero su valor será el de una cadena vacía.

VALIDACIÓN DE FORMULARIOS EN PHP

CONSIDERACIONES SOBRE LA VALIDACIÓN DE FORMULARIOS

Otras condiciones:

- Si una **casilla de verificación** o **botón de tipo radio** estaban marcados pero no tenían definido el atributo HTML **value**, su valor en el array asociativo será **on**.

Por último, si un formulario que contiene un componente para **seleccionar un archivo** creado con `<input type=file">` (usado para **subir archivos al servidor**) no contiene el atributo HTML **enctype="multipart/form-data"**, no quedará registrado en el array asociativo **\$_FILE** (lo explicaremos con más detalle más adelante) sino en **\$_GET** o **\$_POST** (aunque sólo el nombre del archivo).

MANEJO DE BASES DE DATOS EN PHP CON MYSQL

Con php, puedes conectarte y manipular bases de datos. MySQL es el sistema de base de datos más popular utilizado con php.

¿Qué es MySQL?

- ✓ **MySQL** es un sistema de base de datos utilizado en la web.
- ✓ **MySQL** es un sistema de base de datos que se ejecuta en un servidor.
- ✓ **MySQL** es ideal para aplicaciones pequeñas y grandes.
- ✓ **MySQL** es muy rápido, confiable y fácil de usar.
- ✓ **MySQL** utiliza SQL estándar.
- ✓ **MySQL** se compila en varias plataformas.
- ✓ **MySQL** se puede descargar y usar gratis.

Los datos de una base de datos MySQL se almacenan en tablas. Una tabla es una colección de datos relacionados y consta de columnas y filas.

Conexión MySQL

Php 5 y versiones posteriores pueden trabajar con una base de datos MySQL usando:

- ✓ **Extensión MySQLi** (la “i” viene del ingles **improved** que significa mejorado).
- ✓ **PDO** (php Data Objects que en español seria Objetos de Datos PHP).

Las versiones anteriores de php usaban la extensión MySQL. Sin embargo, la extensión esa quedó en desuso en 2012.

Conexión a una base de datos

A continuación vamos a ver dos maneras de conectar a una base de datos.

- ✓ **MySQLi** (procedimental).
- ✓ **MySQLi** (Orientado a objetos).

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

Antes de que podamos acceder a los datos de la base de datos MySQL, debemos poder conectarnos al servidor donde está alojada nuestra base de datos.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_connect():** Antes de realizar cualquier operación relacionada con la base de datos, debe establecer una conexión con el servidor de la base de datos MySQL. Si la conexión se establece correctamente, devuelve un identificador de recurso de conexión de base de datos. Si se produce un error en la conexión, simplemente arroja un error.

```
<?php
// Configuración de la base de datos
$host = "localhost";
$dbuser = "root";
$dbpass = "";
$dbname = "prueba";

// Creando la conexión a la base de datos.
mysqli_connect($host, $dbuser, $dbpass, $dbname);

?>
```


ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CHEQUEANDO LA CONEXIÓN.

- **mysqli_connect_error()**: la función MySQLi arroja un error cuando la conexión no se realiza correctamente y la función almacena el error en la llamada anterior a **mysqli_connect()**. Si no se encuentra ningún error, devuelve **NULL**. Si se encuentra algún error, devuelve un mensaje de error.

```
// Chequeando la conexion
if(mysqli_connect_error())
{
    echo "Se produjo un error en la conexion!";
}
else
{
    echo "Conexión establecida exitosamente.";
}
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_close():** Esta función MySQLi se utiliza para cerrar una base de datos previamente conectada. Esta función devolverá **TRUE** al cerrarse correctamente, de lo contrario, devolverá **FALSE**.

```
<?php
// Creando la conexión a la base de datos
$conn = mysqli_connect($host, $dbuser, $dbpass, $dbname);
//algún código php
if(mysqli_close($conn))
echo "Conexión cerrada exitosamente.";
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_query():** Esta función MySQLi realiza o ejecuta la consulta en la base de datos dada.

```
<?php
require_once "../conexion.php";
$query = "INSERT INTO empleado (ID_Emple, Nombre, Apellidos, Telefono, Profesion)

VALUES ('', 'Juana', 'MBASOGO ONDO', '222385038', 'Contable')";
$resul = mysqli_query($conn, $query);

if ($resul) {
    echo "Datos insertados exitosamente";
}
mysqli_close($conn);
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_fetch_row():** Esta función MySQLi se utiliza para obtener una fila del conjunto de resultados como una matriz enumerada. Cada llamada a la función anterior devolverá la siguiente fila del conjunto de resultados. Si no se obtiene ninguna fila, devolverá **FALSE**.

```
<?php
require_once "../conexion.php";
$query = "SELECT * FROM empleado";
$resul = mysqli_query($conn, $query);
while ($fila = mysqli_fetch_row($resul)) {
    echo "ID: ". $fila[0].", Nombre: ".$fila[1].", Apellidos: ".$fila[2].", Telefono:
    ".$fila[3].", Profesion: ".$fila[4]."</br>";
}
mysqli_close($conn);
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_fetch_array():** Esta función MySQLi se utiliza para obtener una fila como una matriz asociativa, numérica o ambos tipos de matriz del conjunto de resultados.

```
<?php
require_once "../conexion.php";
$query = "SELECT * FROM empleado";
$resul = mysqli_query($conn, $query);
while ($fila = mysqli_fetch_array($resul, MYSQLI_NUM)) {
    echo "ID: ". $fila[0].", Nombre: ".$fila[1].", Apellidos: ".$fila[2].", Telefono:
    ".$fila[3].", Profesion: ".$fila[4]."</br>";
}
mysqli_close($conn);
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_fetch_array():** Esta función MySQLi se utiliza para obtener una fila como una matriz asociativa, numérica o ambos tipos de matriz del conjunto de resultados.

```
<?php
require_once "../conexion.php";
$query = "SELECT * FROM empleado";
$resul = mysqli_query($conn, $query);

while ($fila = mysqli_fetch_array($resul, MYSQLI_ASSOC)) {
    echo "ID: ". $fila['ID_Emple'].", Nombre: ".$fila['Nombre'].", Apellidos:
    ".$fila['Apellidos'].", Telefono: ".$fila['Telefono'].", Profesion:
    ".$fila['Profesion']. "</br>";
}
mysqli_close($conn);
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL. CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_fetch_assoc():** Esta función MySQLi obtiene todas las filas y devuelve el conjunto de resultados como una matriz asociativa.

```
<?php
require_once "../conexion.php";
$query = "SELECT * FROM empleado";

$resul = mysqli_query($conn, $query);

while ($fila = mysqli_fetch_assoc($resul)) {
    echo "ID: ". $fila['ID_Emple'].", Nombre: ".$fila['Nombre'].", Apellidos:
    ".$fila['Apellidos'].", Telefono: ".$fila['Telefono'].", Profesion:
    ".$fila['Profesion']. "</br>";
}
mysqli_close($conn);
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_num_rows():** Esta función MySQLi se utiliza para devolver el numero de filas del conjunto de resultados.

```
<?php
require_once "../conexion.php";
$query = "SELECT * FROM empleado";
$resul = mysqli_query($conn, $query);

$totalRegistro = mysqli_num_rows($resul);

echo "Tenemos $totalRegistro Empleados registrados </br>";

mysqli_close($conn);
```


ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_real_escape_string()**: Esta función MySQLi se utiliza para escapar de los caracteres especiales de una cadena para su uso en consultas MySQL.

```
<?php
```

```
$variable1 = mysqli_real_escape_string($conn,$_POST['var1']);
```

```
$variable2 = mysqli_real_escape_string($conn, $_POST['var2']);
```

```
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS PROCEDIMENTAL.

CONEXIÓN A UNA BASE DE DATOS DE FORMA PROCEDIMENTAL.

- **mysqli_affected_rows():** Esta función MySQLi se utiliza para devolver el número total de filas afectadas de la consulta anterior de MySQL INSERT, SELECT, UPDATE o DELETE.

```
<?php
require_once "../conexion.php";
$query = "INSERT INTO empleado (ID_Emple, Nombre, Apellidos, Telefono,
Profesion) VALUES ('','Maria', 'MANGUE ESONO OWONO', '222385498','Recursos
Humano')";
$resul = mysqli_query($conn, $query);

if ($resul) {
    echo "Datos insertados exitosamente </br>";
}

echo "Total de filas afectadas: ".mysqli_affected_rows($conn);

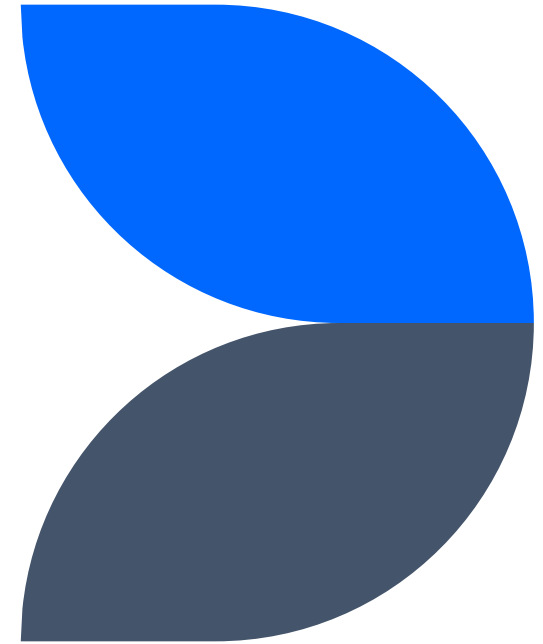
mysqli_close($conn);
```

Integrando código php dentro de html

```
conexion > tabla.php
1  <!doctype html>
2  <html lang="en">
3  <?php
4      require_once "../conexion.php";
5      $query = "SELECT * FROM empleado";
6
7      $resul = mysqli_query($conn, $query);
8  ?>
9
10 <head>
11     <meta charset="utf-8">
12     <meta name="viewport" content="width=device-width, initial-scale=1">
13     <title>Tabla Empleados</title>
14     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
15         integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH" crossorigin="anonymous">
16 </head>
17
18 <body>
19     <div class="container mt-3">
20         <h2>Empleados</h2>
21         <table class="table table-hover">
22             <thead>
23                 <tr>
24                     <th>ID</th>
25                     <th>NOMBRE</th>
26                     <th>APELLIDOS</th>
27                     <th>TELEFONO</th>
28                     <th>PROFESION</th>
29                 </tr>
30             </thead>
31             <tbody>
32                 <?php while ($fila = mysqli_fetch_assoc($resul)) {
33                     ?>
34                     <tr>
35                         <td><?php echo $fila['ID_Emple'];></td>
36                         <td><?php echo $fila['Nombre'];></td>
37                         <td><?php echo $fila['Apellidos'];></td>
38                         <td><?php echo $fila['Telefono'];></td>
39                         <td><?php echo $fila['Profesion'];></td>
40                     </tr>
41                 </tr>
42                 <?php } ?>
43             </tbody>
44         </table>
45     </div>
46     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
47         integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous">
48     </script>
49 </body>
50
51
52 </html>
```

SUBIR UN ARCHIVO AL SERVIDOR

En php



Subir un archivo al servidor en php

¿Cómo subir un archivo al servidor en php?

En este apartado vamos a explicar los conceptos básicos de la carga de archivos al servidor en php, en primer lugar, veremos las opciones de configuración de php que se deben implementar para que los archivos se carguen correctamente.

Configurar las opciones de php

Hay un par de ajustes de configuración de php que querrás verificar de antemano para cargar archivos correctamente. Veremos todas y cada una de las opciones que son importantes en los que respecta a la carga de archivos php. Estas opciones se pueden configurar en el archivo **php.ini**.

Si no estás seguro de dónde encontrar tu archivo **php.ini**, puedes usar la función *php_ini_loaded_file()* para localizarlo. Simplemente crear un archivo php en su servidor con la siguiente línea y ábrelo desde el navegador.

```
<?php echo php_ini_loaded_file(); ?>
```

Subir un archivo al servidor en php

Configurar las opciones de php

A continuación se muestra algunos valores de configuración predeterminados útiles.

```
1  ; Whether to allow HTTP file uploads.
2  file_uploads = On
3
4  ; Temporary directory for HTTP uploaded files.
5  ; Will use system default if not set.
6  ;upload_tmp_dir =
7
8  ; Maximum allowed size for uploaded files.
9  upload_max_filesize = 16M
10
11 ; Maximum number of files that can be uploaded via a single request
12 max_file_uploads = 20
13
14 ; Maximum size of POST data that PHP will accept.
15 post_max_size = 20M
16
17 max_input_time = 60
18 memory_limit = 128M
19 max_execution_time = 30
```

Subir un archivo al servidor en php

Configuraciones clave

- ✓ **file_uploads:** el valor de la directiva **file_uploads** se debe establecer en *on* para permitir la carga de archivos. El valor predeterminado de esta directiva es *on*.
- ✓ **upload_max_filesize:** la directiva **upload_max_filesize** te permite configurar el tamaño máximo del archivo cargado. De forma predeterminada, se establece en **2M** (*dos megabytes*), y puede anular esta configuración utilizando el archivo **.htaccess** también.

Dos megabytes no son mucho para los estándares de hoy, por lo que es posible que tenga que aumentar esto. Si recibes un error de cuando el *archivo excede upload_max_filesize* cuando intenta cargar un archivo, debes aumentar este valor. Si lo hace, asegúrate de aumentar también **post_max_size** (que se vera mas adelante).

- ✓ **upload_tmp_dir:** Establece un directorio temporal que se utilizará para almacenar los archivos cargados. En la mayoría de los casos, no necesita preocuparse por esta configuración. Si no lo configuras, se utilizará el directorio temporal predeterminado del sistema.
- ✓ **post_max_size:** La directiva **post_max_size** te permite configurar el tamaño máximo de los datos POST. Dado que los archivos se cargan con solicitudes POST, este valor debe ser mayor que lo que has establecido para la directiva **upload_max_filesize**.

Por ejemplo, si tu **upload_max_filesize** es **16M** (*16 megabytes*), es posible que desee establecer **post_max_size** en **20M**.

Subir un archivo al servidor en php

Configuraciones clave

- ✓ **max_file_uploads:** Te permite establecer el número máximo de archivos que se pueden cargar a la vez. El valor predeterminado es 20, una cantidad razonable.
- ✓ **max_input_time:** Es el número máximo de segundos que un script puede analizar los datos de entrada. Debe establecerlo en un valor razonable si está tratando con cargas de archivos grandes. **60 (60 segundos)** es un buen valor para la mayoría de las aplicaciones.
- ✓ **memory_limit:** La directiva **memory_limit** indica la cantidad máxima de memoria que puede consumir un script. Si tienes problemas durante la carga de archivos grandes, debe asegurarse de que el valor de esta directiva sea mayor que lo que ha establecido para la directiva **post_max_size**. El valor predeterminado es **128M (128 megabytes)**, así que, a menos que tenga un **post_max_size** y **upload_max_filesize** muy grande, no debe preocuparse por eso.
- ✓ **max_execution_time:** Es el número máximo de segundos que se permite ejecutar un script. Si tiene problemas durante la carga de archivos grandes, puede considerar aumentar este valor **30 (30 segundos)** deberían funcionar bien para la mayoría de las aplicaciones.

Subir un archivo al servidor en php

Crear el formulario HTML

Una vez que hayas configurado la configuración de php, estarás listo para probar las capacidades de carga de archivos de php.

Para que un formulario sea capaz de enviar archivos al servidor, al formulario **<form>** tendremos que agregarle el atributo **enctype**, con su valor **multipart/form-data**, y al formulario le agregamos un campo de tipo archivo **“file”**.

El atributo **enctype**, especifica el tipo de codificación que se debe usar cuando se envía el formulario, y toma uno de los siguientes tres valores.

- ✓ **application/x-www-form-urlencoded:** Este es el valor predeterminado cuando no establece el valor del atributo **enctype** explícitamente. En este caso, los caracteres se codifican antes de que se envíen al servidor. Si no tienes el campo de archivo en su formulario, debes usar este valor para el atributo **enctype**.
- ✓ **multipart/form-data:** Cuando utiliza el valor **multipart/form-data** para el atributo **enctype**, te permite cargar archivos utilizando el método **POST**. Además, se asegura de que los caracteres no estén codificados cuando se envíe el formulario.
- ✓ **text/plain:** Generalmente no se usa. Con esta configuración, los datos se envían sin codificar.

SUBIR UN ARCHIVO AL SERVIDOR EN PHP

CREAR LA LÓGICA DE CARGA

En la sección anterior, creamos el formulario HTML que se muestra en el lado del cliente y te permite cargar un archivo desde su computadora. En esta sección, veremos la contraparte del lado del servidor que le permite manejar el archivo cargado.

En php, cuando se carga un archivo, la variable superglobal **\$_FILES** se rellena con toda la información sobre el archivo cargado. Se inicializa como un arreglo y puede contener la siguiente información para la carga exitosa de archivos.

- ✓ **tmp_name:** La ruta temporal donde se carga el archivo se almacena en esta variable.
- ✓ **name:** El nombre real del archivo se almacena en esta variable.
- ✓ **size:** Indica el tamaño del archivo cargado en bytes.
- ✓ **type:** Contiene el tipo mime del archivo cargado.
- ✓ **error:** Si hay un error durante la carga del archivo, esta variable se rellena con el mensaje de error correspondiente. En el caso de que se cargue correctamente el archivo, contiene **0**, que puede comparar utilizando la constante **UPLOAD_ERR_OK**.

Después de validar la solicitud POST, verificamos que la carga del archivo se realizó correctamente.

```
if (isset($_FILES['uploadedFile']) && $_FILES['uploadedFile']['error']== UPLOAD_ERR_OK) {  
    # code...  
}
```

SUBIR UN ARCHIVO AL SERVIDOR EN PHP

CREAR LA LÓGICA DE CARGA

Se puede ver que la variable `$_FILES` es un arreglo multidimensional, el primer elemento es el nombre del campo de archivo, y el segundo elemento tiene la información sobre el archivo cargado.

Si la carga del archivo es exitosa, inicializamos algunas variables con información sobre el archivo cargado.

```
$dirTemporal = $_FILES['NomArchivo']['tmp_name'];  
$NombreArchivo = $_FILES['NomArchivo']['name'];  
$TamañoArchivo = $_FILES['NomArchivo']['size'];  
$tipoArchivo = $_FILES['NomArchivo']['type'];  
$arrayNombreArchi = explode(".", $NombreArchivo);  
$extensionArchivo = strtolower(end($arrayNombreArchi));
```

En el fragmento anterior, también hemos descubierto la extensión del archivo cargado y lo hemos almacenado en la variable `$extensionArchivo`.

SUBIR UN ARCHIVO AL SERVIDOR EN PHP

CREAR LA LÓGICA DE CARGA

Es importante que restrinjas el tipo de archivo que se puede cargar a ciertas extensiones.

```
$exten = array("jpg", "gif", "png");
```

Finalmente, usamos la función **move_uploaded_file()** para mover el archivo cargado a la ubicación específica de nuestra elección.

```
<?php
if (isset($_FILES['NomArchivo']) && $_FILES['NomArchivo']['error']== UPLOAD_ERR_OK) {
    $destino = "img/";
    $dirTemporal = $_FILES['NomArchivo']['tmp_name'];
    $NombreArchivo = $_FILES['NomArchivo']['name'];
    $TamañoArchivo = $_FILES['NomArchivo']['size'];
    $tipoArchivo = $_FILES['NomArchivo']['type'];

    $fileNameComps = explode(".", $NombreArchivo);
    $fileExtension = strtolower(end($fileNameComps));

    $exten = array("jpg", "gif", "png");

    if (in_array($fileExtension, $exten)) {

        echo "El archivo es admitido";

        if (move_uploaded_file($dirTemporal, $destino.$NombreArchivo)) {

            echo "Archivo guardado con exito";
        }

    }else{

        echo "No se admite ese tipo de archivo";
    }
}
?>
```

LAS EXCEPCIONES

En php



MANEJO DE EXCEPCIONES EN PHP

¿QUÉ SON LAS EXCEPCIONES EN PHP?

Las excepciones son una forma de manejar errores que ocurren durante la ejecución de un programa. Una excepción es un objeto que representa un error o una condición inusual que ha ocurrido en el código y que puede interrumpir el flujo normal de ejecución. Php proporciona para la gestión de errores las palabras reservadas “try”, “catch”, “throw” y “finally”.

Una excepción es un objeto que representa un error o evento inusual durante la ejecución del código. Estos objetos contienen información valiosa sobre el problema, facilitando su resolución.

¿CUÁNDO USAR EXCEPCIONES?

Debemos usar excepciones cuando:

- ✓ **Pueda ocurrir un error que interrumpe el flujo normal del programa:** ¿algunos ejemplos? La imposibilidad de conectarse a una base de datos o la falta de un archivo requerido.

¿CUÁNDO NO USAR EXCEPCIONES?

- ✓ **Validación simple de datos:** Para errores de validación, es más eficiente devolver **false** o un mensaje de error en lugar de usar excepciones. En ocasiones, una estructura **if/else** puede ser suficiente y eficiente para resolver problemas simples.

MANEJO DE EXCEPCIONES EN PHP

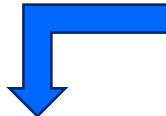
INICIAR UNA EXCEPCIÓN

La instrucción **throw** permite que un usuario defina función o método para iniciar una excepción. Cuando se produce una excepción, el código siguiente no se ejecutará.

Si no se detecta una excepción, se producirá un error irre recuperable con un mensaje “**No detectado**” excepción.

Intentemos lanzar una excepción sin detectarla.

El resultado se verá así:



```
Fatal error: Uncaught Exception: Division by zero in C:\webfolder\test.php:4
Stack trace: #0 C:\webfolder\test.php(9):
divide(5, 0) #1 {main} thrown in C:\webfolder\test.php on line 4
```

```
<?php
function divide($dividendo, $divisor) {
    if($divisor == 0) {
        throw new Exception("Division entre cero no
está admitido");
    }
    return $dividendo / $divisor;
}
echo divide(5, 0);
?>
```

MANEJO DE EXCEPCIONES EN PHP

Para evitar el error del ejemplo anterior, podemos usar la declaración **try....catch** para detectar excepciones y continuar el proceso.

ESTRUCTURA BÁSICA DEL MANEJO DE EXCEPCIONES EN PHP

El manejo de excepciones en php sigue el siguiente patrón básica.

- ✓ **try:** Aquí colocamos el código que puede generar una excepción.
- ✓ **catch:** En esta parte capturamos la excepción y definimos cómo manejarla.
- ✓ **finally:** Este bloque es opcional y se ejecuta siempre, tanto si ocurre una excepción como si no.

```
try {  
    //Código que puede generar una excepcion.  
} catch (Exception $e) {  
    //Código para manejar la excepcion.  
}finally{  
    //Código que se ejecuta siempre, independientemente de si ocurrió una excepcion o no  
}
```


MANEJO DE EXCEPCIONES EN PHP

MÉTODOS

Al detectar una excepción, en la siguiente tabla se muestran algunos de los métodos que se pueden usar para obtener información sobre la excepción.

Método	Descripción
getMessage()	Devuelve una cadena que describe por qué se lanzó la excepción.
getPrevious()	Si esta excepción fue activada por otra, este método devuelve la excepción anterior. Si no, devuelve null
getCode()	Devuelve el código de excepción
getFile()	Devuelve la ruta completa del archivo en el que se lanzó la excepción.
getLine()	Devuelve el número de línea de la línea de código que generó la excepción.

MANEJO DE EXCEPCIONES EN PHP

MÉTODOS `getCode()`

El método `getCode()` devuelve un número entero que se puede utilizar para identificar la excepción.

SINTAXIS

`$excepcion->getCode()`.

```
<?php
try {
    throw new Exception("se produjo un error", 120);
} catch(Exception $e) {
    echo "Error code: " . $e->getCode();
}
?>
```

MANEJO DE EXCEPCIONES EN PHP

MÉTODOS `getFile()`

El método `getFile()` devuelve la ruta de acceso absoluta al archivo en el que se produjo una excepción.

SINTAXIS

`$excepcion->getFile()`.

```
<?php
try {
    throw new Exception("se produjo un error", 120);
} catch(Exception $e) {
    echo "Error en este archivo: " . $e->getFile();
}
?>
```

MANEJO DE EXCEPCIONES EN PHP

MÉTODOS getMessage()

El método **getMessage()** devuelve una descripción del error o comportamiento que provocó que se produjera la excepción.

SINTAXIS

`$excepcion->getMessage()`.

```
<?php
try {
    throw new Exception("se produjo un error", 120);
} catch(Exception $e) {
    echo $e->getMessage();
}
?>
```

MANEJO DE EXCEPCIONES EN PHP

MÉTODOS `getLine()`

El método `getLine()` devuelve el número de línea de código que produjo la excepción.

SINTAXIS

`$excepcion->getLine()`.

```
<?php
try {
    throw new Exception("se produjo un error", 120);
} catch(Exception $e) {
    echo $e->getLine();
}
?>
```

MANEJO DE EXCEPCIONES EN PHP

MÉTODOS `getPrevious()`

Si la excepción fue desencadenada por otra, el método `getPrevious()` devuelve la otra excepción. De lo contrario, devuelve **null**.

SINTAXIS

`$excepcion->getPrevious()`.

```
<?php
try {
    try {
        throw new Exception("Se produjo un error", 1);
    } catch(Exception $e1) {
        throw new Exception("Ocurrió otro error", 2, $e1);
    }

} catch (Exception $e2) {
    echo $previous = $e2->getPrevious();
    echo $previous->getMessage();
}
?>
```

MANEJO DE EXCEPCIONES EN PHP

Ejemplo básico

```
try {  
    $archivo = fopen("archivo_inexistente.txt", "r");  
    if (!$archivo) {  
        throw new Exception("El archivo no se pudo abrir", 1);  
    }  
} catch (Exception $e) {  
    echo "Excepcion capturada: ".$e->getMessage();  
}finally{  
    echo "Este bloque se ejecuta siempre";  
}
```

MANEJO DE EXCEPCIONES EN PHP

EXCEPCIONES PERSONALIZADAS

En php, es posible crear excepciones personalizadas que extienden la clase base **Exception**. Esto nos permite capturar tipos específicos de errores y manejarlos de manera diferente.

EJEMPLO DE EXCEPCIONES PERSONALIZADA:

```
class MiExcepcionPersonalizada extends Exception{}  
try {  
    $edad = 15;  
    if ($edad < 18) {  
        throw new MiExcepcionPersonalizada("El Usuario es menor de edad", 1);  
    }  
} catch (MiExcPer $e) {  
    echo "Excepción personalizada capturada: ".$e->getMessage();  
}finally{  
    echo "Este bloque se ejecuta siempre";  
}
```


MANEJO DE EXCEPCIONES EN PHP

CONEXIÓN A UNA BASE DE DATOS

Uno de los escenarios más comunes donde utilizamos excepciones es al trabajar con base de dato.

Ejemplo:

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost; dbname=realrebola","root","");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch ( PDOExveption $e) {
    echo "Error en la conexion: ".$e->getMessage();
}finally{
    echo "Intento de conexion completado.";
}
```

MANEJO DE EXCEPCIONES EN PHP

LECTURA DE ARCHIVO

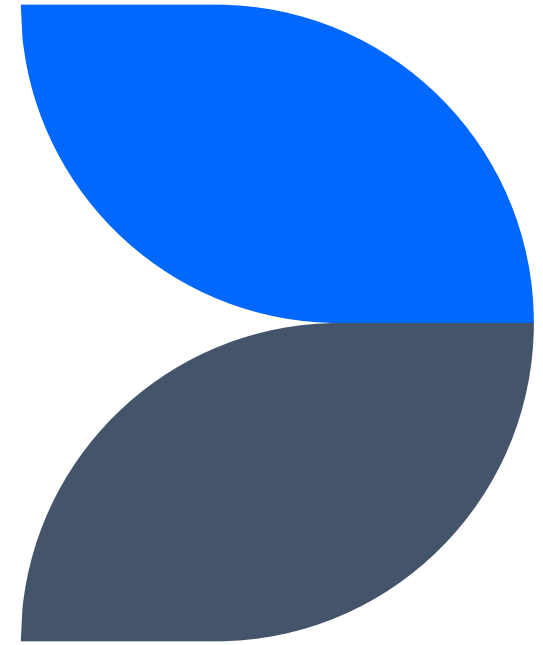
Si necesitamos leer un archivo que podría no existir o no estar disponible, también es útil manejar las excepciones.

Ejemplo:

```
<?php
try {
    if (!file_exists("datos.txt")) {
        throw new Exception("El archivo no existe </br>", 1);
    }
    $contenido = file_get_contents("datos.txt");
} catch (Exception $e) {
    echo "Error: ".$e->getMessage();
}finally{
    echo "</br> Operacion de la lectura de archivo finalizada.";
}
```

MySQLi

ORIENTADO A OBJETOS



ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

MySQLi es una clase que viene por defecto en MySQL, la cual nos ayuda a acceder a funcionalidades en MySQL, como la conexión a BD.

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Para poder abrir una conexión, tenemos que instanciar la clase **mysqli()**.

Sintaxis.

```
$con = new mysqli (host, usuario, contraseña, basedatos);
```

Parametros	Descripción
Host	Especifique el nombre del servidor o la dirección IP.
Usuario	Especifique el nombre del usuario MySQL
Contraseña	Especifique la contraseña de MySQL
Base de datos	Especifique la base de datos que vas a utilizar en el servidor.

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

MySQLi es una clase que viene por defecto en MySQL, la cual nos ayuda a acceder a funcionalidades en MySQL, como la conexión a BD.

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Para poder abrir una conexión, tenemos que instanciar la clase **mysqli()**.

ejemplo.

```
<?php
    $mysqli = new mysqli("localhost","my_user","my_password","my_db");

// Comprobando la conexión
if ($mysqli -> connect_errno) {
    echo "Fallo en la conexión: " . $mysqli -> connect_error;
    exit();
}

?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función `connect_errno()` de php MySQLi

La función `connect_errno` / `mysqli_connect_errno()`, devuelve el código de error del último error de conexión, si lo hubiera.

Sintaxis orientado a objetos:

`$con->connect_errno`

Sintaxis procedimental:

`Mysqli_connect_errno()`

```
<?php
    $mysqli = new mysqli("localhost","my_user","my_password","my_db");

    // Check connection
    if ($mysqli -> connect_errno) {
        echo "No se pudo conectar al servidor: " . $mysqli -> connect_error;
        exit();
    }
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función close() de php MySQLi

La función `close()` / `mysqli_close()`, cierra una conexión de base de datos previamente abierta.

Sintaxis orientado a objetos:

`$con->close()`

Sintaxis procedimental:

`Mysqli_close($conn)`

```
<?php
    $mysqli = new mysqli("localhost","my_user","my_password","my_db");

    if ($mysqli -> connect_errno) {
        echo "No se pudo conectar al servidor: " . $mysqli -> connect_error;
        exit();
    }

    // ....algún código de PHP...

    $mysqli -> close();
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función `set_charset()` de php MySQLi

La función `set_charset()` / `mysqli_set_charset()`, se utiliza para especifica el juego de caracteres que vamos a utilizar. Para los caracteres hispanos, utilizaremos “**utf8**”

Sintaxis orientado a objetos:

`$conn->set_charset(charset)`

Sintaxis procedimental:

`Mysqli_set_charset($conn, charset)`

```
<?php
    $mysqli = new mysqli("localhost","my_user","my_password","my_db");

    if ($mysqli -> connect_errno) {
        echo " No se pudo conectar al servidor: " . $mysqli -> connect_error;
        exit();
    }

    echo " El juego de carácter inicial es: " . $mysqli -> character_set_name();

    // Cambiar el conjunto de carácter a utf8
    $mysqli -> set_charset("utf8");

    echo " El conjunto de caracteres actuales es: " . $mysqli -> character_set_name();

    $mysqli -> close();
?>
```


ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función query() de php MySQLi

La función **query()** / **mysqli_query()**, realiza una consulta en una base de datos.

Sintaxis orientado a objetos:

`$conn->query(sentencia)`

Sintaxis procedimental:

`Mysqli_query($conn, sentencia)`

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

// Comprobar la conexión
if ($mysqli -> connect_errno) {
    echo "No se pudo conectarse al servidor: " . $mysqli -> connect_error;
    exit();
}

// Realizar consulta
if ($result = $mysqli -> query("SELECT * FROM Persons")) {
    echo "Las filas devueltas son: " . $result -> num_rows;
}
$mysqli -> close();
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función `fetch_all()` de php MySQLi

La función `fetch_array()` / `mysqli_fetch_array()`, obtiene una fila de resultados como una matriz asociativa, una matriz numérica o ambas.

Sintaxis orientado a objetos:

`$result->fetch_array(tipo de resultado)`

Sintaxis procedimental:

`Mysqli_fetch_array($result, tipo de resultado)`

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");
if ($mysqli -> connect_errno) {
    echo "No se pudo conectar al servidor: " . $mysqli -> connect_error;
    exit();
}
$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result = $mysqli -> query($sql);
// Array indexado
$row = $result -> fetch_array(MYSQLI_NUM);
printf ( $row[0], $row[1]);
// Array Asociativo
$row = $result -> fetch_array(MYSQLI_ASSOC);
printf ($row["Lastname"], $row["Age"]);
$mysqli -> close();
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función `fetch_assoc()` de php MySQLi

La función `fetch_assoc()` / `mysqli_fetch_assoc()`, obtiene una fila de resultados como una matriz asociativa.

Sintaxis orientado a objetos:

`$result->fetch_assoc()`

Sintaxis procedimental:

`mysqli_fetch_assoc($result)`

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

if ($mysqli -> connect_errno) {
    echo "No se pudo conectar al servidor: " . $mysqli -> connect_error;
    exit();
}
$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result = $mysqli -> query($sql);
// Array Asociativo
$row = $result -> fetch_assoc();
printf (" $row["Lastname"], $row["Age"]);

$mysqli -> close();
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función insert_id() de php MySQLi

La función `insert_id()` / `mysqli_insert_id()`, devuelve el id (generado con **AUTO_INCREMENT**) de la última consulta.

Sintaxis orientado a objetos:

`$conn->insert_id()`

Sintaxis procedimental:

`Mysqli_insert_id($conn)`

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

if ($mysqli -> connect_errno) {
    echo "No se pudo conectar al servidor: " . $mysqli -> connect_error;
    exit();
}

$mysqli -> query("INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Glenn', 'Quagmire', 33)");

// Imprimir identificación generada automáticamente.
echo "El nuevo registro tiene el ID: " . $mysqli -> insert_id;

$mysqli -> close();
?>
```

ABRIR UNA CONEXIÓN A LA BASE DE DATOS (MySQLi POO)

CONEXIÓN A UNA BASE DE DATOS CON MySQLi ORIENTADO A OBJETO

Función de `affected_rows` de php MySQLi

La función `affected_rows` / `mysqli_affected_rows()`, devuelve el numero de filas afectadas en la consulta **SELECT**, **INSERT**, **UPDATE**, **DELETE**.

Sintaxis orientado a objetos:

`$con->affected_rows`

Sintaxis procedimental:

`Mysqli_affected_rows($conn)`

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

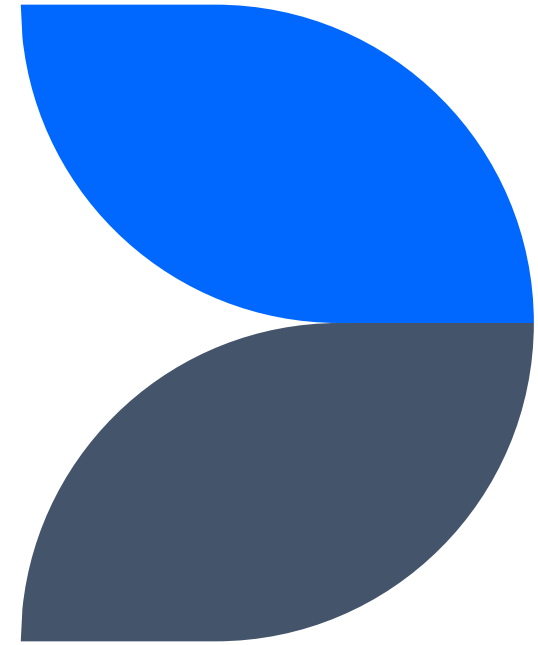
if ($mysqli -> connect_errno) {
    echo "No se pudo conectar al servidor: " . $mysqli -> connect_error;
    exit();
}

// Realizar consultas e imprimir las filas afectadas
$mysqli -> query("SELECT * FROM Persons");
echo "Filas afectadas: " . $mysqli -> affected_rows;

$mysqli -> query("DELETE FROM Persons WHERE Age>32");
echo "Filas afectadas: " . $mysqli -> affected_rows;

$mysqli -> close();
?>
```

*INYECCIONES SQL, Y
SENTENCIAS PREPARADAS
EN PHP Y MySQL*



INYECCIÓN DE SQL

¿Qué es la inyección de SQL?

La **inyección SQL** consiste en **insertar código SQL para alterar la base de datos** de la página web. Se suele hacer en los formularios de login o en los formularios de alta de usuario o cuando se sabe que se va a interactuar con una base de datos.

Ejemplo:

Supongamos que tenemos un formulario de login donde se pide el nombre de usuario y damos al boton de enviar (o submit) y tenemos la siguiente sentencia en php.

```
$query = "SELECT * FROM usuarios WHERE nombre = '$nombreusuario.'";"
```

Se supone que la variable **nombre de usuario** contiene el nombre del usuario que se va a logear en nuestra aplicación. Pero alguien malintencionado podría introducir en el formulario otra cosa como por ejemplo **juan';DROP TABLE usuarios; SELECT * FROM datos WHERE nombre LIKE '%';** y después hacer la consulta *select* borra la tabla de usuarios.

Una solución sencilla sería usar la función **mysqli_real_escape_string()** y nos podría evitar muchos problemas.

INYECCIÓN DE SQL

La función `real_escape_string()`

La función `real_escape_string()` / `mysqli_real_escape_string()` escapa los caracteres especiales en una cadena de caracteres para su uso en una consulta SQL, teniendo en cuenta el conjunto de caracteres actuales de la conexión.

SINTAXIS.

Estilo orientado a objetos:

```
$conn->real_escape_string($variable);
```

Estilo procedimental:

```
mysqli_real_escape_string($conn, $variable);
```

```
$nombre = $conn -> real_escape_string($_POST['nombre']);
```

```
$apellidos = mysqli_real_escape_string($con, $_POST['apellidos']);
```

¿Cómo prevenir un ataque de inyección SQL?

Existen distintos tipos de medidas que se deben considerar cuando se necesita comunicar un sistema con una base de datos relacional a través del uso de SQL. Estas son algunas medidas que se deberían implementar.

- Uso de consultas parametrizadas.
- Validación de entrada de datos de usuarios.
- *Escapar* todas las entradas permitidas de los usuarios.

INYECCIÓN DE SQL

¿Qué es una prepared Statements?

Las *prepared statements*, también llamadas *consultas*, *comandos* o *sentencias preparadas*, son plantillas para **consultas a sistemas de bases de datos en lenguaje SQL** cuyos parámetros están desprovistos de valores. Para reemplazar dichos valores, estas plantillas trabajan con variables o marcadores de posición, que no son sustituidos por los valores reales hasta estar dentro del sistema.

¿por qué conviene utilizar prepared statements en MySQL y sistemas similares?

La principal razón para utilizar sentencias preparadas cuando se trabaja con sistemas de gestión de bases de datos como MySQL no es otra que la **seguridad**. El mayor problema de los métodos convencionales de **acceso a las bases de datos basadas en lenguaje SQL** es la facilidad con la que pueden ser manipuladas. Este tipo de ataque se denomina **inyección SQL**.

¿Cómo se utiliza exactamente una prepared statements?

Sin entrar en los detalles de la sintaxis del lenguaje de programación ni de las características de cada sistema de gestión de base de datos, **la incorporación y el uso de sentencias preparadas** suele dividirse en las siguientes fases.



INYECCIÓN DE SQL

¿Cómo se utiliza exactamente una prepared statements?

Fase 1: preparación

El primer paso es generar una plantilla de sentencia en php, la función correspondiente es *prepare()*. En lugar de los valores, a los parámetros relevantes se les asignan los ya mencionados **marcadores de posición**, también llamados **parámetros de sustitución posicionales** o **variables bind**. En general, estos marcadores se caracterizan por un signo de interrogación (?),

```
$query = "INSERT INTO Productos (Nombre, Precio) VALUES (?,?)";
```

Fase 2: procesamiento de la plantilla en el DBMS

El sistema de gestión de base de datos (DBMS) *parsea*, es decir, analiza sintácticamente la plantilla de sentencia para que en un siguiente paso se pueda compilar, es decir, convertirse en una orden ejecutable. Durante este proceso, además, se optimiza la *prepared statement*.

Fase 3: ejecución

En un momento posterior, la aplicación vincula los valores a los parámetros y la base de datos ejecuta la instrucción. La aplicación puede ejecutar la sentencia tantas veces como quiera con diferentes valores.

SENTENCIAS PREPARADAS PHP Y MySQL

las **sentencias preparadas** son muy útiles frente a inyecciones SQL, ya que los valores de los parámetros, que son transmitidos después usando un protocolo diferente, no necesitan ser escapados.

Tabla de ejemplo

En este ejemplo partimos de los siguientes datos en nuestra tabla.

ID	NOMBRE	APELLIDOS	EMAIL
1	Juan Antonio	NKISOGO OWONO	juanan@gmail.com
2	Felisa	MANGUE ESONO	felisamangue@hotmail.com
3	Juan Manuel	BOTE BAITA	jmanuel2b@gmail.com
4	Marcelina	BANCH BELOPE	marcelina2b@hotmail.com

SENTENCIAS PREPARADAS PHP Y MySQL

PARÁMETROS

Una cadena que contiene uno o más caracteres que especifican los tipos para el correspondiente enlazado de variables.

CARÁCTER	DESCRIPCIÓN
i	La variable correspondiente es de tipo entero
d	La variable correspondiente es de tipo double
s	La variable correspondiente es de tipo string
b	La variable correspondiente es un blob

Al decirle a mysql qué tipo de datos espera, minimizamos el riesgo de inyección de SQL.

SENTENCIAS PREPARADAS PHP Y MySQL

FUNCIONES REQUERIDAS

Para realizar una sentencia preparada, necesitamos las siguientes funciones:

- ✓ **mysqli_prepare():** Prepara la consulta SQL y devuelve un identificador de declaración que se utilizará para operaciones posteriores en la declaración.
- ✓ **mysqli_stmt_bind_param():** Es usada para enlazar variables para los marcadores de parámetros en la sentencia SQL que fue pasada a *mysqli_prepare()*.
- ✓ **mysqli_stmt_execute():** Ejecuta una consulta que ha sido previamente preparada usando la función *mysqli_prepare()*.
- ✓ **mysqli_stmt_get_result():** Obtiene un conjunto de resultados de una sentencia preparada.

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “*SELECT*” CON MySQL PROCEDIMENTAL

Ejecutaremos una sentencia de tipo **SELECT** en que filtraremos por campo *Nombre*.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";
$usuario = "root";
$pass = "";
$basedatos = "dawes";

$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);

mysqli_set_charset($conn, "utf8");

if (!$conn) {
    die("Conexion fallada: ".mysqli_connect_error());
}
```

Consulta preparada de selección.

```
require "../conexion.php";
$buscar = "juan";
$sql = "SELECT * FROM empleado WHERE Nombre = ?";
$stmt = mysqli_prepare($conn, $sql);
mysqli_stmt_bind_param($stmt, "s", $buscar);
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);
if (mysqli_num_rows($result) > 0) {
    while ($fila = mysqli_fetch_assoc($result)) {
        echo "ID: ".$fila['ID_Emple']." - Nombre: ".$fila['Nombre']." - Apellidos ".$fila['Apellidos'];
    }
} else {
    echo "0 resultado...";
}
//cerramos la conexion
mysqli_close($conn);
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “*SELECT*” CON MySQL PROCEDIMENTAL

Otro ejemplo seria utilizar **LIKE** en la consulta, en cuyo caso sería de esta forma.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";
$usuario = "root";
$pass = "";
$basedatos = "dawes";

$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);

mysqli_set_charset($conn, "utf8");

if (!$conn) {
    die("Conexion fallada: ".mysqli_connect_error());
}
```

Consulta preparada de selección.

```
require "../conexion.php";
$buscar = "%juan%";
$sql = "SELECT * FROM empleado WHERE Nombre LIKE ?";
$stmt = mysqli_prepare($conn, $sql);
mysqli_stmt_bind_param($stmt, "s", $buscar);
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);
if (mysqli_num_rows($result) > 0) {
    while ($fila = mysqli_fetch_assoc($result)) {
        echo "ID: ".$fila['ID_Emple']." - Nombre: ".$fila['Nombre']." - Apellidos ".$fila['Apellidos'];
    }
} else {
    echo "0 resultado...";
}
//cerramos la conexion
mysqli_close($conn);
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “INSERT” CON MySQL PROCEDIMENTAL

Realizaremos un INSERT en la tabla empleado.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";  
$usuario = "root";  
$pass = "";  
$basedatos = "dawes";  
  
$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);  
  
mysqli_set_charset($conn, "utf8");  
  
if (!$conn) {  
    die("Conexion fallada: ".mysqli_connect_error());  
}
```

Consulta preparada de inserción.

```
require "../conexion.php";  
  
$sqlInser = "INSERT INTO empleado (Nombre, Apellidos, Telefono, Profesion)  
VALUES(?,?,?,?)";  
$stmtInsert = mysqli_prepare($conn, $sqlInser);  
mysqli_stmt_bind_param($stmtInsert, "ssss", $nom, $apell, $tlf, $prof);  
$nom = "Ernesto";  
$apell = "ROCA BELOPE";  
$tlf = "222 304 384";  
$prof = "Contable";  
mysqli_stmt_execute($stmtInsert);
```


SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “UPDATE” CON MySQL PROCEDIMENTAL

Realizaremos un UPDATE sobre un campo de tipo texto.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";  
$usuario = "root";  
$pass = "";  
$basedatos = "dawes";  
  
$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);  
  
mysqli_set_charset($conn, "utf8");  
  
if (!$conn) {  
    die("Conexion fallada: ".mysqli_connect_error());  
}
```

Consulta preparada de actualización.

```
require "../conexion.php";  
  
$antiguoNom = "Juana";  
$nuevoNom = "Martha";  
  
$sqlUpdate = "UPDATE empleado SET Nombre = ? WHERE Nombre = ?";  
$stmtUpdate = mysqli_prepare($conn, $sqlUpdate);  
mysqli_stmt_bind_param($stmtUpdate, "ss", $nuevoNom, $antiguoNom);  
mysqli_stmt_execute($stmtUpdate);
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “DELETE” CON MySQL PROCEDIMENTAL

Eliminaremos un registro filtrando por su campo *Nombre*.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";
$usuario = "root";
$pass = "";
$basedatos = "dawes";

$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);

mysqli_set_charset($conn, "utf8");

if (!$conn) {
    die("Conexion fallada: ".mysqli_connect_error());
}
```

Consulta preparada para eliminar.

```
require "../conexion.php";

$dato = "Maria";
$sqlBorrar = "DELETE FROM empleado WHERE Nombre = ?";
$stmtBorrar = mysqli_prepare($conn, $sqlBorrar);
mysqli_stmt_bind_param($stmtBorrar, "s", $dato);
mysqli_stmt_execute($stmtBorrar);

//cerramos la conexion
mysqli_close($conn);
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “*SELECT*” CON MySQL ORIENTADA A OBJETOS

Ejecutaremos una sentencia de tipo *SELECT* en la que filtraremos por el campo *Nombre*.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";
$usuario = "root";
$pass = "";
$basedatos = "dawes";

$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);

mysqli_set_charset($conn, "utf8");

if (!$conn) {

    die("Conexion fallada: ".mysqli_connect_error());
}
```

Consulta preparada de seleccion.

```
require "../conexion.php";

$buscar = "Martha";
$sqlBuscar = "SELECT * FROM empleado WHERE Nombre = ?";
$stmtBuscar = $conn->prepare($sqlBuscar);
$stmtBuscar->bind_param('s',$buscar);
$stmtBuscar->execute();
$resultBuscar = $stmtBuscar->get_result();

if ($resultBuscar->num_rows > 0) {
    while ($fila = $resultBuscar->fetch_assoc()) {
        echo "</br> ID: ".$fila['ID_Emple']." - Nombre: ".$fila['Nombre']." - Apellidos
        ".$fila['Apellidos'];
    }
}else{
    echo "0 Resultado...";
}

$conn->close();
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “*SELECT*” CON MySQL ORIENTADO A OBJETOS

Otro ejemplo seria utilizar **LIKE** en la consulta, en cuyo caso sería de esta forma.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";
$usuario = "root";
$pass = "";
$basedatos = "dawes";

$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);

mysqli_set_charset($conn, "utf8");

if (!$conn) {

    die("Conexion fallada: ".mysqli_connect_error());
}
```

Consulta preparada de selección.

```
require "../conexion.php";

$buscar = "%r%";
$sqlBuscar = "SELECT * FROM empleado WHERE Nombre LIKE ?";
$stmtBuscar = $conn->prepare($sqlBuscar);
$stmtBuscar->bind_param('s', $buscar);
$stmtBuscar->execute();
$resultBuscar = $stmtBuscar->get_result();

if ($resultBuscar->num_rows > 0) {
    while ($fila = $resultBuscar->fetch_assoc()) {
        echo "</br> ID: ".$fila['ID_Emple']." - Nombre: ".$fila['Nombre']." - Apellidos  

        ".$fila['Apellidos'];
    }
} else {
    echo "0 Resultado...";
}

$conn->close();
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “INSERT” CON MySQL ORIENTADO A OBJETOS

Realizaremos un INSERT en la tabla empleado.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";  
$usuario = "root";  
$pass = "";  
$basedatos = "dawes";  
  
$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);  
  
mysqli_set_charset($conn, "utf8");  
  
if (!$conn) {  
    die("Conexion fallada: ".mysqli_connect_error());  
}
```

Consulta preparada de inserción.

```
require "../conexion.php";  
  
$sqlInser = "INSERT INTO empleado (Nombre, Apellidos, Telefono, Profesion)  
VALUES(?,?,?,?)";  
$stmtInsert = $conn->prepare($sqlInser);  
$stmtInsert->bind_param('ssss',$nom,$apell,$tlf,$prof);  
$nom = "Hector";  
$apell = "BOLOLO BORINSCALLA";  
$tlf = "222 300 004";  
$prof = "Economista";  
$stmtInsert->execute();
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “UPDATE” CON MySQL ORIENTADO A OBJETOS

Realizaremos un UPDATE sobre un campo de tipo texto.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";  
$usuario = "root";  
$pass = "";  
$basedatos = "dawes";  
  
$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);  
  
mysqli_set_charset($conn, "utf8");  
  
if (!$conn) {  
    die("Conexion fallada: ".mysqli_connect_error());  
}
```

Consulta preparada de actualización.

```
require "../conexion.php";  
  
$antiguoNom = "Martha";  
$nuevoNom = "Juana";  
  
$sqlUpdate = "UPDATE empleado SET Nombre = ? WHERE Nombre = ? ";  
$stmtUpdate= $conn->prepare($sqlUpdate);  
$stmtUpdate->bind_param('ss', $nuevoNom, $antiguoNom);  
$stmtUpdate->execute();
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “UPDATE” CON MySQL ORIENTADO A OBJETOS

Realizaremos un UPDATE sobre un campo de tipo texto.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";  
$usuario = "root";  
$pass = "";  
$basedatos = "dawes";  
  
$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);  
  
mysqli_set_charset($conn, "utf8");  
  
if (!$conn) {  
    die("Conexion fallada: ".mysqli_connect_error());  
}
```

Consulta preparada de actualización.

```
require "../conexion.php";  
  
$antiguoNom = "Martha";  
$nuevoNom = "Juana";  
  
$sqlUpdate = "UPDATE empleado SET Nombre = ? WHERE Nombre = ? ";  
$stmtUpdate= $conn->prepare($sqlUpdate);  
$stmtUpdate->bind_param('ss', $nuevoNom, $antiguoNom);  
$stmtUpdate->execute();
```

SENTENCIAS PREPARADAS PHP Y MySQL

EJEMPLO DE SENTENCIA “*DELETE*” CON MySQL ORIENTADO A OBJETOS

Eliminaremos un registro filtrando por su campo *Nombre*.

Conexión a la base de datos.

```
$servidor = "127.0.0.1";
$usuario = "root";
$pass = "";
$basedatos = "dawes";

$conn = mysqli_connect($servidor, $usuario, $pass, $basedatos);

mysqli_set_charset($conn, "utf8");

if (!$conn) {
    die("Conexion fallada: ".mysqli_connect_error());
}
```

Consulta preparada para eliminar.

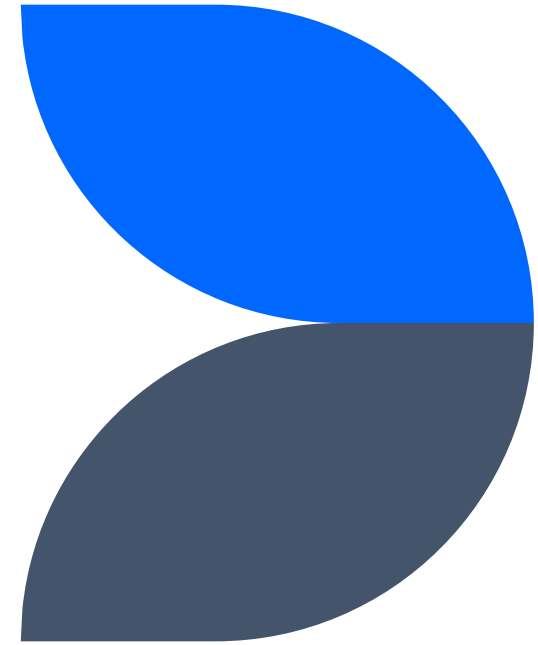
```
require "../conexion.php";

$dato = "Hector";
$sqlBorrar = "DELETE FROM empleado WHERE Nombre = ?";
$stmtBorrar = $conn->prepare($sqlBorrar);
$stmtBorrar->bind_param('s',$dato);
$stmtBorrar->execute();

//cerramos la conexion
mysqli_close($conn);
```


PDO

PHP DATA OBJECTS (OBJETOS DE DATOS EN PHP)



PDO (PHP DATA OBJECTS)

¿Qué es PDO?

PDO significa **PHP Data Objects, Objetos de Datos de PHP**, Es una extensión para acceder a bases de datos. PDO permite acceder a diferentes **sistemas de gestión de bases de datos** con un controlador específico (**MySQL, SQLite, Oracle, SQL Server, PostgreSQL...**) mediante el cual se conecta.

Independientemente del sistema utilizado, **se empleará siempre los mismos métodos**, lo que hace que cambiar de uno a otro resulte más sencillo.

Para ver los **controladores (drivers)** disponibles en tu servidor, puedes emplear el método **getAvailableDrivers()**

```
print_r(PDO::getAvailableDrivers());
```

El **sistema PDO** se fundamenta en 3 clases: **PDO, PDOStatement** y **PDOException**.

- ✓ La clase **PDO** se utiliza para representar la conexión entre php y un servidor de bases de datos.
- ✓ La clase **PDO, PDOStatement**, representa una sentencia preparada y también nos permite acceder al conjunto de resultados asociados, es decir, es la que maneja las sentencias SQL y devuelve los resultados.
- ✓ La clase **PDOException**, se utiliza para manejar los errores.

PDO (PHP DATA OBJECTS): ACCESO A BASES DE DATOS CON PDO

CONEXIÓN A LA BASE DE DATOS

En primer argumento de la clase **PDO** es el **DNS (Data Source Name** o “El Nombre de Origen de Datos”, *que contiene la información necesaria para conectarse a la base de datos*), en el cual se han de especificar el **tipo de base de datos** (*mysql*), el **host**(*localhost*) y el **nombre de la base de datos** (se puede especificar también el **puerto**). Diferentes sistemas de bases de datos tienen **distintos métodos para conectarse**. La mayoría se conectan de forma parecida a como se conecta a **MySQL**.

```
$pdo = new PDO("mysql:host=$servidor;dbname=$db",$usuario, $pass);
```

mysql, es el driver, representa el tipo de base de datos. Podría ser: **mssql**, **PostgreSQL**, **sqlite**, etc.

Ejemplos de conexión a diferentes tipos de base de datos.

```
//conectando a un SGBD MySQL
$pdo = new PDO("mysql:host=$servidor;dbname=$db",$usuario, $pass);
//conectando a MS SQL Server y PostgreSQL
$pdo1 = new PDO("mssql:host=$servidor;dbname=$db", $usuario, $pass);
$pdo2 = new PDO("pgsql:host=$servidor;dbname=$db",$usuario, $pass);
```

PDO (PHP DATA OBJECTS)

EXCEPCIONES Y OPCIONES CON PDO

PDO maneja los errores en forma de excepciones, por lo que la conexión siempre ha de ir encerrada en un bloque **try/catch**. Se puede (y se debe) especificar el modo de error estableciendo el atributo ***error mode***.

PDO puede utilizar las excepciones para gestionar los errores, lo que significa que cualquier cosa que hagamos con **PDO** podríamos encapsularla en un bloque **try/catch**. Para gestionar si produce algún error.

```
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);  
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);  
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

No importa el modo de error, si existe un fallo en la conexión siempre producirá una excepción, por eso siempre se conecta con **try/catch**. Podemos forzar **PDO** para que trabaje en cualquier de los tres modos.

- ✓ **PDO::ERRMODE_SILENT**. Es el **modo de error por defecto, predeterminado**, en el que **PDO** no genera advertencias ni lanza excepciones. Para determinar los errores, debes hacer uso de las siguientes funciones **PDO::errorCode()** y **PDO::errorInfo()** o su versión en **PDOStatement** **PDOStatement::errorCode()** y **PDOStatement::errorInfo()**.

```
$conn = new PDO("mysql:host=localhost; dbname=dawes; charset=utf8", "root", "");  
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);  
$sql = "INSERT INTO empleado VALUES ('1','Domingo','BOLOLO BELOPE','555 192 302','Contable')";  
$result = $conn->exec($sql);  
  
if ($result) {  
    echo "\nDato insertado exitosamente";  
}else{  
    echo "<pre>";  
    var_dump( "\n PDO::errorCode(): ", $conn->errorCode());  
    echo "</pre>";  
}
```

PDO (PHP DATA OBJECTS)

EXCEPCIONES Y OPCIONES CON PDO

- ✓ **PDO::ERRMODE_WARNING.** Establece el código de error y emite un mensaje **E_WARNING**, Genera errores warning php pero permitiría la ejecución normal de la aplicación. Modo empleado para depurar o hacer pruebas para ver errores sin interrumpir el flujo de la aplicación.

```
$conn = new PDO("mysql:host=localhost; dbname=dawes; charset=utf8", "root", "");
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
$sql = "INSERT INTO empleado VALUES ('1','Domingo','BOLOLO BELOPE','555 192 302', 'Contable')";
$result = $conn->exec($sql);

if ($result) {
    echo "\nDato insertado exitosamente";
}else{
    echo "<pre>";
    var_dump( "Información del error: ", $conn->errorInfo());
    echo "</pre>";
}
```

- ✓ **PDO::ERRMODE_EXCEPTION.** Será la forma más utilizada en **PDO**. Dispara una excepción permitiéndonos gestionar el error de forma amigable.

Cualquier error que se lance a través de **PDO**, el sistema lanzará una **PDOException**

Nota: **PDO::ATTR_ERRMODE**, es un reporte de errores y **PDO::ERRMODE_EXCEPTION**, con este atributo obligamos a que lance excepciones, además de ser la opción más humana y legible que hay a la hora de controlar errores.

PDO (PHP DATA OBJECTS)

CONECTAR A UNA BASE DE DATOS CON PDO

```
$host = "localhost";
$user = "root";
$pass = "";
$db = "dawes";

try {
    $dns = "mysql:host=$host;dbname=$db";
    $conn = new PDO($dns,$user,$pass);
    if ($conn) {
        print("conexion exitosa");
    }
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

```
$servidor = "localhost";
$usuario = "root";
$pass = "";
$db = "dawes";
// $pdo = new PDO("mysql:host=$servidor; dbname=$db; charset=utf8", $usuario, $pass);
try {
    //conexion con MySQL
    $pdo = new PDO("mysql:host=$servidor; dbname=$db; charset=utf8", $usuario, $pass);
    //para que genere excepciones a la hora de reportar errores
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    if ($e->getCode()===1049) {
        echo "No se pudo localizar la base de datos $db";
    }else if ($e->getCode() === 1045) {
        echo "Usuario o Contraseña incorrecta";
    }else if ($e->getCode()===2002) {
        echo "Servidor no encontrado.";
    }
}

echo "</br>hola mundo";
```

PDO (PHP DATA OBJECTS)

CERRAR LA CONEXIÓN A LA BASE DE DATOS

Se recomienda cerrar siempre la conexión a la base de datos cuando no se vaya a utilizar más durante nuestro proceso.

Hay que recordar que los **recursos son limitados** y cuando hay pocos usuarios no hay ningún problema, pero si tenemos muchos usuarios simultáneamente entonces es cuando surgen problemas al haber alcanzado el número máximo de conexiones con el servidor, por ejemplo:

Al cerrar la conexión de forma explícita aceleramos la liberación de recursos para que estén disponibles para otros usuarios.

Si quisiéramos cerrar la conexión con la base de datos simplemente podríamos hacer al final del fichero.

```
$pdo = null;
```

PDO (PHP DATA OBJECTS)

INSERT, UPDATE Y DELETE

Insertar nuevos datos, actualizarlos o borrarlos son algunas de las operaciones más comunes en una base de datos. Con **PDO**, se suelen hacer de la siguiente manera.

Para insertar datos con **PDO** podemos utilizar las siguientes funciones **query** o **exec** del objeto **PDO**. Estos métodos son útiles cuando son sentencias SQL que no reciben parámetros.

```
$sql = "INSERT INTO empleado VALUES ('Florencio','NOE BOTAU','+240 222 405 953','Logistico')";  
$pdo->query($sql);
```

El método **query**, es recomendable para **SELECT**. Y hacemos lo mismo para las demás sentencias (actualizar y eliminar).

Y con **exec**, devolverá el número de registros afectados por la consulta (recomendable para **INSERT, UPDATE** o **DELETE**).

```
$sql1 = "DELETE FROM empleado WHERE Nombre = 'Florencia';"  
$result = $pdo->exec($sql1);  
echo $result;
```


PDO (PHP DATA OBJECTS)

CONSULTAS PREPARADAS

Una consulta preparada es una sentencia SQL precompilada que se puede ejecutar múltiples veces simplemente enviando datos al servidor.

El uso de consultas preparadas con **prepare()** nos ayudara a evitar la inyección SQL, con lo que se recomienda utilizarlo.

Para construir una sentencia preparada hay que hacerlo incluyendo unos **marcadores** en nuestra sentencia SQL.

Marcadores anónimos

//Marcadores anónimos

```
$sql = "INSERT INTO empleado (ID_Emple, Nombre, Apellidos, Telefono, Profesion) VALUES (?, ?, ?, ?, ?)";
```

Marcadores conocidos

//Marcadores conocidos

```
$sql1 = "INSERT INTO empleado (Nombre, Apellidos, Telefono, Profesion) VALUES (:nombre, :apellidos, :telefono, :profesion)";
```

Sin marcadores, aquí no lleva marcadores, ideal para una inyección SQL (no se recomienda usar este método por motivos de seguridad, hay que utilizar marcadores).

```
$sql2 = "INSERT INTO empleado (Nombre, Apellidos, Telefono, Profesion) VALUES ($nombre, $apellidos, $telefono, $profesion)";
```

PDO (PHP DATA OBJECTS)

REGISTRAR DATOS CON PDO

La clase **PDOStatement** es la que trata las **sentencias SQL**. Una **instancia de PDOStatement** se crea cuando se llama a **PDO->prepare()**, y con ese objeto creado se llama a métodos como **bindParam()** para pasar valores o **execute()** para ejecutar sentencias. PDO facilita el uso de sentencias preparadas en php, que mejoran el rendimiento y la seguridad de la aplicación. Cuando se **obtienen, insertan o actualizan datos**, el esquema es: **PREPARE->[BIND]->EXECUTE**. Se pueden indicar los parámetros en la sentencia con un interrogante “?” o mediante un **nombre específico**.

➤ *Registrando datos con PDO utilizando interrogantes para los valores*

```
// Prepare
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES (?, ?)");

// Bind
$nombre = "Peter";
$ciudad = "Madrid";
$stmt->bindParam(1, $nombre);
$stmt->bindParam(2, $ciudad);

// Execute
$stmt->execute();
```

PDO (PHP DATA OBJECTS)

REGISTRAR DATOS CON PDO

- *Registrando datos con PDO utilizando variables para los valores*

```
// Prepare
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES (:nombre, :ciudad)");
// Bind
$nombre = "Charles";
$ciudad = "Valladolid";
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':ciudad', $ciudad);
// Execute
$stmt->execute();
```

PDO (PHP DATA OBJECTS)

REGISTRAR DATOS CON PDO

También existe un **método lazy**, que es **pasando los valores mediante un array** (siempre array, aunque sólo haya un valor) al método *execute()*.

```
// Prepare:
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES (:nombre, :ciudad)");
$nombre = "Luis";
$ciudad = "Barcelona";

// Bind y execute:
if($stmt->execute(array(':nombre'=>$nombre, ':ciudad'=>$ciudad))) {
    echo "Se ha creado el nuevo registro!";
}
```

Es el método *execute()* el que realmente **envía los datos a la base de datos**. Si no se llama a *execute* no se obtendrá los resultados sino un error.

PDO (PHP DATA OBJECTS)

DIFERENCIA ENTRE `bindParam()` y `bindValue()`

Existen dos métodos para enlazar valores: **`bindParam()`** y **`bindValue()`**

- con **`bindParam()`**, se vincula la variable al parámetro y en el momento de hacer el **`execute`** es cuando se asigna realmente el valor de la variable a ese parámetro.

```
// Prepare:
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
$nombre = "Morgan";

// Bind

$stmt->bindParam(':nombre', $nombre); // Se enlaza a la variable $nombre

// Si ahora cambiamos el valor de $nombre:

$nombre = "John";

$stmt->execute(); // Se insertará el cliente con el nombre John
```

PDO (PHP DATA OBJECTS)

DIFERENCIA ENTRE `bindParam()` y `bindValue()`

- con **`bindValue()`**, Se asigna el valor de la variable a ese parámetro justo en el momento de ejecutar la instrucción **`bindValue`**, es decir, se enlaza el valor de la variable y **permanece hasta *execute()***.

```
// Prepare:
$stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
$nombre = "Morgan";

// Bind

$stmt->bindValue(':nombre', $nombre); // Se enlaza al valor Morgan

// Si ahora cambiamos el valor de $nombre:

$nombre = "John";

$stmt->execute(); // Se insertará el cliente con el nombre Morgan
```

En la práctica **`bindValue()`** se suele usar cuando se tienen que insertar datos sólo una vez, y **`bindParam()`** cuando se tiene que pasar datos múltiples (desde un array por ejemplo).

Ambas funciones aceptan un **tercer parámetro**, que define el tipo de dato que se espera. Los **data types** más utilizados son: `PDO::PARAM_BOOL` (booleano), `PDO::PARAM_NULL` (null), `PDO::PARAM_INT` (integer) y `PDO::PARAM_STR` (string).

PDO (PHP DATA OBJECTS)

TIPOS DE DATOS UTILIZADOS EN `bindParam()` y `bindValue()`

- Formato de `bindParam()`, Con marcadores conocidos.

```
bindParam(':marcador', $variableVincular, TIPO DATOS PDO);
```

Ejemplo de uso de `bindParam`

```
$stmt->bindParam(':nombre', $nombre, PDO::PARAM_STR);  
$stmt->bindParam(':nombre', $nombre, PDO::PARAM_STR, 20); // 20 CARACTERES COMO MÁXIMO  
$stmt->bindParam(':edad', $edad, PDO::PARAM_INT);
```

El primer argumento de `bindParam` es el nombre del marcador y el segundo la variable que contendrá los datos.

Los marcadores conocidos siempre comienzan con :

```
$stmt->bindParam(':name', $name);
```

```
$name='Pepito';
```

```
$stmt->execute();
```

Otra forma es creando un array asociativo con los marcadores y sus valores:

Los datos a insertar en forma de array asociativo

```
$datos = array('name' => 'Carla', 'addr' => '9 Dark and Twisty', 'city' => 'Cardiff');
```

Fijarse que se pasa el array de datos en `execute()`.

```
$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) value (:name, :addr, :city)");
```

```
$stmt->execute($datos);
```

La última instrucción se podría poner también así:

```
$stmt->execute(array('name' => 'Cathy', 'addr' => '9 Dark and Twisty', 'city' => 'Cardiff'));
```

PDO (PHP DATA OBJECTS)

Ejemplo de INSERT

```
try {  
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $username, $password);  
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
    $stmt = $pdo->prepare('INSERT INTO Tabla VALUES (:name)');  
    $stmt->execute(  
        array(  
            ':name' => 'Justin Bieber'  
        ));  
    # filas afectadas  
    echo $stmt->rowCount(); // 1  
} catch(PDOException $e) {  
    echo 'Error: ' . $e->getMessage();  
}
```


PDO (PHP DATA OBJECTS)

Ejemplo de UPDATE

```
$id = 5;
$name = "Joe the Plumber";
try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->prepare('UPDATE Tabla SET name = :name WHERE id = :id');
    $stmt->execute(array(
        ':id' => $id,
        ':name' => $name
    ));
    echo $stmt->rowCount(); // 1
} catch(PDOException $e) {
    echo 'Error: ' . $e->getMessage();
}
```

PDO (PHP DATA OBJECTS)

Ejemplo de DELETE

```
$id = 5; // de un formulario o algo similar

try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->prepare('DELETE FROM Tabla WHERE id = :id');
    $stmt->bindParam(':id', $id); //Esta vez vamos a utilizar el método bindParam
    $stmt->execute();
    echo $stmt->rowCount(); // 1
} catch(PDOException $e) {
    echo 'Error: ' . $e->getMessage();
}
```

PDO (PHP DATA OBJECTS)

CONSULTAR DATOS CON PDO

Los datos se obtienen a través del método **->fetch** o **->fetchAll** de (PDOStatement).

- **->fetch:** Obtiene la siguiente fila de un recordset (conjunto de resultados).
- **->fetchAll:** Devuelve un array que contiene todas las filas del conjunto de resultados (el tipo de datos a devolver se puede indicar como parámetro).

Antes de llamar al **método ->fetch()** una buena idea es indicarle cómo queremos que nos devuelva los datos de la base de datos. Tendremos las siguientes opciones en el método **->fetch()**.

- **PDO::FETCH_ASSOC:** Devuelve un array indexado por el nombre de campo de la tabla.
- **PDO::FETCH_BOTH (por defecto):** Devuelve un array indexado por nombre de campo de la tabla y por numero de campo.
- **PDO::FETCH_BOUND:** Asigna los valores devueltos a las variables asignadas con el método **->bindColumn()**. **PDOStatement::bindColumn.**
- **PDO::FETCH_CLASS:** Asigna los valores de los campos a las propiedades una clase. Si las propiedades no existen en esa clase, las creará.
- **PDO::FETCH_INT:** Actualiza una instancia existente de una clase.
- **PDO::FETCH_OBJ:** Devuelve un objeto anónimo con nombres de propiedades que corresponden a las columnas.
- **PDO::FETCH_LAZ:** Combina **PDO::FETCH_BOTH** y **PDO::FETCH_OBJ**, creando los nombres de las propiedades del objeto tal como se accedieron.
- **PDO::FETCH_NUM:** Devuelve un array indexado por el numero de campo.

Para ajustar el modo de respuesta:

```
$stmt->setFetchMode(PDO::FETCH_ASSOC);
```

PDO (PHP DATA OBJECTS)

CONSULTAR DATOS CON PDO

Los más utilizados son **FETCH_ASSOC**, **FETCH_OBJ**

➤ **FETCH_ASSOC**, **FETCH_OBJ**

Para ejecutar la consulta **SELECT** si no tenemos parámetros en la consulta podremos usar **->query()** del objeto PDO. Veamos un *ejemplo de consulta SELECT*

```
try {  
    # Para ejecutar la consulta SELECT si no tenemos parámetros en la consulta podremos usar ->query()  
    $stmt = $pdo->query('SELECT name, addr, city from colegas');  
    # Indicamos en qué formato queremos obtener los datos de la tabla en formato de array asociativo.  
    # Si no indicamos nada por defecto se usará FETCH_BOTH lo que nos permitirá acceder como un array asociativo o array numérico.  
    $stmt->setFetchMode(PDO::FETCH_ASSOC);  
    # Leemos los datos del recordset con el método ->fetch()  
    while ($row = $stmt->fetch()) {  
        echo $row['name'] . "<br/>";  
        echo $row['addr'] . "<br/>";  
        echo $row['city'] . "<br/>";  
    }  
    # Para liberar los recursos utilizados en la consulta SELECT  
    $stmt = null;  
} catch (PDOException $err) {  
    // Mostramos un mensaje genérico de error.  
    echo "Error: ejecutando consulta SQL.";  
}
```

PDO (PHP DATA OBJECTS)

CONSULTAR DATOS CON PDO

Para ejecutar la consulta **SELECT** si tenemos parámetros lo haríamos así.

Para ejecutar la consulta SELECT si tenemos parámetros lo haríamos así:

Preparamos la consulta como siempre

```
$stmt = $pdo->prepare('SELECT name, addr, city FROM colegas WHERE city =:ciudad');
```

```
try{
```

Indicamos en qué formato queremos obtener los datos de la tabla en formato de array asociativo.

Si no indicamos nada por defecto se usará FETCH_BOTH lo que nos permitirá acceder como un array asociativo o array numérico.

```
$stmt->execute(array('ciudad'=>'Santiago de Compostela');
```

Leemos los datos del recordset con el método ->fetch()

Por defecto ya los devuelve en forma de array asociativo si no indicamos nada con setFetchMode()

```
while ($row = $stmt->fetch()) {  
    echo $row['name'] . "<br/>";  
    echo $row['addr'] . "<br/>";  
    echo $row['city'] . "<br/>";  
}
```

Para liberar los recursos de la consulta SELECT

```
$stmt=null;
```

```
} catch(PDOException $err) {
```

// Mostramos un mensaje genérico de error.

```
    echo "Error: ejecutando consulta SQL.";
```

```
}
```

PDO (PHP DATA OBJECTS)

CONSULTAR DATOS CON PDO

➤ FETCH_OBJ

En este tipo de modo de consulta se creará un objeto standard por cada fila que leemos del recordset

```
try{  
    # Creamos la consulta  
    $stmt = $pdo->query('SELECT name, addr, city FROM colegas');  
    # Ajustamos el modo de obtención de datos  
    $stmt->setFetchMode(PDO::FETCH_OBJ);  
    # Mostramos los resultados.  
    # Fijaros que se devuelve un objeto cada vez que se lee una fila del recordset.  
    while($row = $stmt->fetch()) {  
        echo $row->name . "<br/>";  
        echo $row->addr . "<br/>";  
        echo $row->city . "<br/>";  
    }  
    # Liberamos los recursos utilizados por $stmt  
    $stmt=null;  
} catch(PDOException $err) {  
    // Mostramos un mensaje genérico de error.  
    echo "Error: ejecutando consulta SQL.";  
}
```

PDO (PHP DATA OBJECTS)

OTROS MÉTODOS INTERESANTES.

- **lastInsertId():** Devuelve el id del ultimo registro insertado en la tabla. Es un método de PDO.

```
$pdo->lastInsertId();
```

- **rowCount():** Devuelve un entero indicando el numero de filas afectadas por la ultima operación.

```
$rows_affected = $stmt->rowCount();
```

PDO (PHP DATA OBJECTS)

Consultar datos con PDO

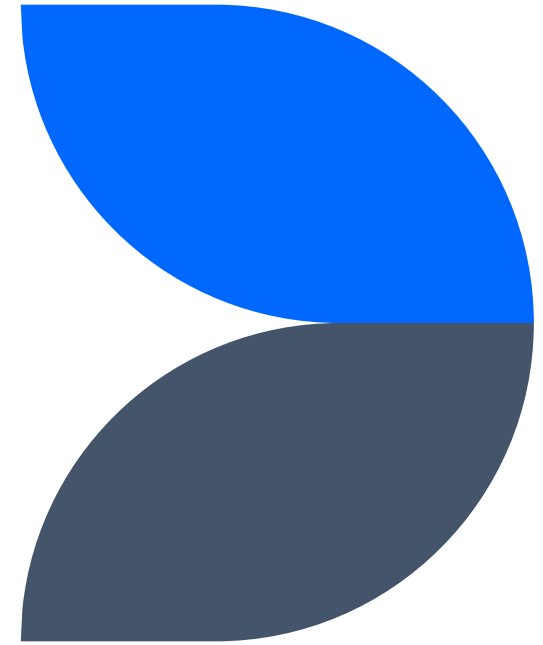
Finalmente, para la consulta de datos también se puede emplear directamente **PDOStatement::fetchAll()**, que devuelve un **array con todas las filas devueltas por la base de datos** con las que poder iterar. También acepta estilos de devolución.

```
// fetchAll() con PDO::FETCH_ASSOC
$stmt = $dbh->prepare("SELECT * FROM Clientes");
$stmt->execute();
$clientes = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach($clientes as $cliente){
    echo $cliente['nombre'] . "<br>";
}
```

```
// fetchAll() con PDO::FETCH_OBJ
$stmt = $dbh->prepare("SELECT * FROM Clientes");
$stmt->execute();
$clientes = $stmt->fetchAll(PDO::FETCH_OBJ);
foreach ($clientes as $cliente){
    echo $cliente->nombre . "<br>";
}
```


AJAX

ENVIÓ Y RECEPCIÓN DE DATOS AL
SERVIDOR CON AJAX Y PHP



ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

¿Qué es AJAX?

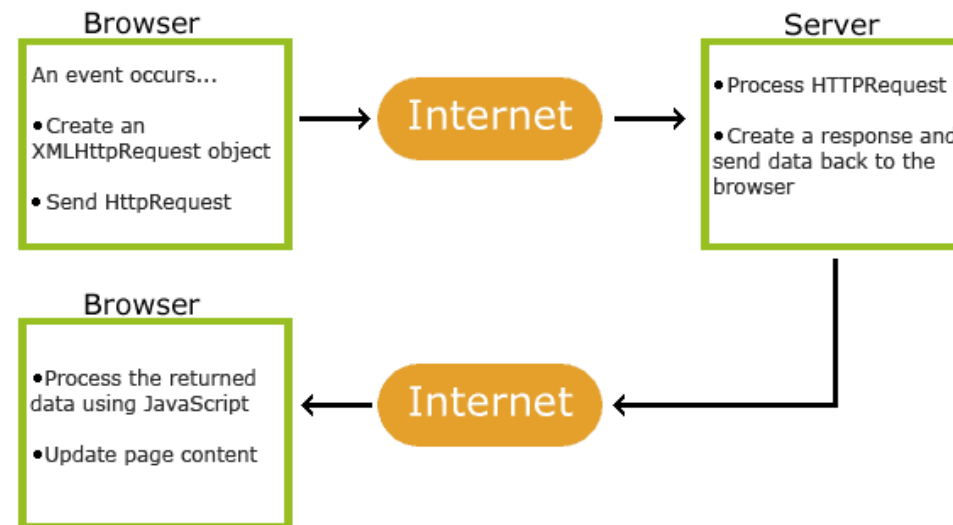
AJAX, es el acrónimo de **Asynchronous JavaScript and XML** (JavaScript y XML asincrónico).

AJAX, consiste en actualizar partes de una pagina web, sin tener que volver a cargar toda la página.

AJAX es una técnica para crear páginas web rápidas y dinámicas, AJAX permite que las páginas web se actualice de forma asincrónica mediante el intercambio de pequeñas cantidades de datos con el servidor detrás de escena. Esto significa que es posible actualizar partes de una página web, sin tener que volver a cargar toda la página.

Las páginas web clásicas (que no utilizan AJAX) deben volver a cargar toda la página si el contenido debería cambiar.

¿Cómo funciona AJAX?



ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

El objeto XMLHttpRequest

XMLHttpRequest, es un objeto que existe en el navegador desde hace mucho tiempo. De hecho, de entre todos los modelos de trabajo para realizar conexiones por **AJAX**, **XMLHttpRequest** *es el más antiguo que existe* y por supuesto está disponible en todos los navegadores desde la época de **IE6**.

XMLHttpRequest es una clase existente en el navegador que está preparada para realizar las **conexiones asíncronas para obtener datos de nuestro servidor web o de cualquier servicio web** de internet que trabaje por HTTP.

Cuando queremos hacer conexión AJAX podemos instanciar un objeto de la clase **XMLHttpRequest** con JavaScript. Posteriormente podremos realizar llamadas a métodos para realizar la conexión y escuchar diversos eventos disponibles dentro del objeto para saber cuándo se ha recibido respuesta.

EL OBJETO XMLHttpRequest

MÉTODOS Y PROPIEDADES DEL OBJETO XMLHttpRequest

Los métodos disponibles para el objeto XMLHttpRequest son los siguientes:

Método	DESCRIPCIÓN
new XMLHttpRequest()	Crea un nuevo objeto XMLHttpRequest
Open(“método”, “url”)	Establece los parámetros de la petición que se realiza al servidor. Los parámetros necesarios son el método HTTP empleado y la URL destino.
Send()	Envía la solicitud al servidor. Utilizado para solicitudes GET
Send(contenido)	Envía la solicitud al servidor. Utilizado para solicitudes POST

El método **open()** requiere dos parámetros (método HTTP y URL) y acepta de forma opcional otros tres parámetros. Definición formal del método open().

open(string método, string URL, [boolean asíncrono, string usuario, string password]).

EL OBJETO XMLHttpRequest

MÉTODOS Y PROPIEDADES DEL OBJETO XMLHttpRequest

Las propiedades definidas para el objeto XMLHttpRequest son:

PROPIEDADES	DESCRIPCIÓN
readyState	Valor numérico (entero) que almacena el estado de la petición.
responseText	El contenido de la respuesta del servidor en forma de cadena de texto.
status	El código de estado HTTP devuelto por el servidor (200 para una respuesta correcta, 404 para “No encontrado”, 500 para un error de servidor, etc.)

Los valores definidos para la propiedad *readyState* son los siguientes:

Valor	DESCRIPCIÓN
0	No inicializado (objeto creado, pero no se ha invocado el método open).
1	Cargando (objeto creado, pero no se ha invocado el método send).
2	Cargando (se ha invocado el método send , pero el servidor aún no ha respondido).
3	Interactivo (se han recibido algunos datos, aunque no se pueden emplear la propiedad <i>responseText</i>).
4	Completado (se han recibido todos los datos de la respuesta del servidor)

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest EL OBJETO XMLHttpRequest

Para instanciar el objeto **XMLHttpRequest** lanzamos la siguiente sentencia.

```
let xhr = new XMLHttpRequest();
```

CONFIGURAR LA SOLICITUD A REALIZAR POR XMLHttpRequest

Ahora podemos realizar métodos para gestionar las comunicaciones por AJAX, el más usado seria **open()**, al que tenemos que suministrar el método de HTTP que vamos a usar y la URL que queremos consultar.

```
xhr.open("GET", url-de-consulta.html);
```

La URL de consulta puede ser relativa a la pagina desde donde hacemos la llamada o bien absoluta, comenzado por <http://> o <https://>

ENVIANDO LA SOLICITUD AJAX

Una vez configurada la consulta con el método **open()** lo que nos debe quedar claro es que no se realiza todavía la solicitud contra el servidor propiamente dicha, sino que la tendremos que invocar nosotros cada vez que sea necesario con el método **send()**.

```
xhr.send();
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

EVENTOS DE XMLHttpRequest PARA GESTIONAR LA RESPUESTA DEL SERVIDOR

En método **send()** no devuelve nada, ¿Cómo obtenemos la respuesta del servidor una vez la solicitud AJAX ha recibido los datos de vuelta? Para ello usamos los eventos que levanta el objeto **XMLHttpRequest**.

Estos eventos son varios y nos pueden servir para ejecutar acciones ante diversos supuestos.

- **Evento onload:** Este es el evento más básico, que se ejecuta cada vez que el objeto **XMLHttpRequest** ha recibido la respuesta completamente.
- **Evento onreadystatechange:** Este es el evento que nos permite acompañar el estado de la solicitud AJAX, desde la creación del objeto, la realización de la solicitud, en adelante.

ENVIO DE DATOS POR POST CON XMLHttpRequest

Vamos a comenzar viendo rápidamente el formulario HTML que vamos a usar. No tiene nada de especial. En realidad, solo lo colocamos para que se vea que es un formulario normal y corriente, pues toda la funcionalidad de envió mediante AJAX la tenemos que realizar con JavaScript.

- ❖ Si el envió es por el método **GET**, la URL de la petición debe enviar los datos, como cualquier URL normal.
Ejemplo: `archivo.php?dato1=valor1&dato2=valor2`.
- ❖ Si el envió es por el método **POST**, los datos se pueden enviar a través del argumento del **send()**.

```
<div class="container mt-4">
  <form action="/php/procesar.php" method="POST" id="MyForm">
    <div class="row">
      <div class="col-sm-12 col-md-6 col-xl-6 mt-2">
        <label for="" class="form-label">Usuario:</label>
        <input autocomplete="off" placeholder="Ingrese su nombre de usuario aqui" type="text"
          class="form-control" name="usuario" id="" />
      </div>
      <div class="col-sm-12 col-md-6 col-xl-6 mt-2">
        <label for="" class="form-label">Contraseña:</label>
        <input autocomplete="off" placeholder="Ingrese su contraseña aqui" type="password"
          class="form-control" name="pass" id="" />
      </div>
    </div>
    <div class="row mt-2">
      <div class="col-12">
        <input type="submit" value="Iniciar sesion" class="btn btn-success form-control" />
      </div>
    </div>
  </form>
</div>
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

CONFIGURAR EL EVENTO DE ENVÍO DEL FORMULARIO CON JAVASCRIPT

Como segundo paso vamos a realizar el **evento de envío de formulario**, que tendremos que detectar para **evitar que se mande de manera corriente**.

Para ello vamos a acceder al objeto del formulario y crear un evento **“submit”**, parando el envío del formulario predeterminado.

```
let miformulario = document.querySelector("#MyForm");
miformulario.addEventListener("submit", function(e){
    e.preventDefault();//El formulario se le bloquea para que no se envíe de manera normal
});
```

CÓDIGO PARA ENVÍO POR AJAX ENVIANDO DATOS POR GET

Antes de continuar también es necesario que veamos como podemos enviar los datos por **GET**, si es que lo deseamos.

```
let form = document.getElementById('MyForm');
form.addEventListener("submit", function (e) {
    let nom = document.getElementById('nom').value;
    let pass = document.getElementById('pass').value;
    let xhr = new XMLHttpRequest();
    e.preventDefault();
    xhr.open("GET", './php/sesion.php?nombre='+nom+'&password='+pass,true);
    xhr.onload = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {
            console.log(xhr.responseText);
            document.getElementById('respuesta').innerHTML = xhr.responseText;
        }
    }
    xhr.send();
});
```


ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

CONFIGURAR EL EVENTO DE ENVÍO DEL FORMULARIO CON JAVASCRIPT

CÓDIGO PARA ENVÍO POR AJAX ENVIANDO DATOS POR POST

Ahora vamos con el código que más nos interesa, que es justamente la parte de la conexión por AJAX y el envío de los datos por **POST**. Se pondrá todo el código a continuación y lo iremos comentando después.

```
let miformulario = document.querySelector("#MyForm");
miformulario.addEventListener("submit", function(e){
    //El formulario se le bloquea para que no se envíe de manera normal

    e.preventDefault();
    let xhr = new XMLHttpRequest();
    xhr.open("POST", miformulario.action);
    xhr.setRequestHeader('content-type', 'application/x-www-form-urlencoded');
    xhr.send(datosFormulario());
    xhr.onload = function(e){
        document.getElementById('respuesta').innerHTML = xhr.responseText;
    }
});
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

CÓDIGO PARA ENVÍO POR AJAX ENVIANDO DATOS POR POST

Comenzamos viendo cómo **configurar el objeto AJAX** para que se envíe **usando el método POST** del HTTP.

```
xhr.open("POST", miformulario.action);
```

Recuerda que con el método **open()** solamente se prepara la solicitud. Le estamos pasando **“POST”** como método del HTTP y con **miformulario.action** accedemos al valor del atributo action de nuestro formulario HTML.

Luego vemos algo muy importante, que es la **cabecera del HTTP** que se debe enviar para que se sepa que se están mandando datos por POST.

```
xhr.setRequestHeader('content-type', 'application/x-www-form-urlencoded');
```

Por último, los **datos enviados por el formulario** los tenemos que componer nosotros mismos y enviarlos en el método **send()**.

```
xhr.send(datosFormulario());
```

Como se puede apreciar para componer los datos estamos usando una función llamada **datosFormulario()**, que se encargará de acceder a los campos del formulario que tengamos y colocarlos todos en una cadena. Esta función podría ser algo como esto.

```
function datosFormulario(){  
    let datos = '';  
    datos += 'usuario='+document.getElementById('user').value;  
    datos += '&password='+document.getElementById('pass').value;  
    return datos;  
}
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

USANDO FormData PARA COMPONER LOS DATOS DEL FORMULARIO

Para utilizar **FormData**, vamos a crear un objeto de la clase FormData indicando el formulario sobre el que queremos obtener los datos. Esto lo conseguimos de manera tan sencilla como puedes ver.

```
let objFormData = new FormData(document.getElementById('MyForm'));
```

Usando **FormData** además sería innecesaria la cabecera que habíamos entregado antes **application/x-www-form-urlencoded**. Con todo, el código del envío por POST de un formulario con AJAX podría resumirse así.

```
let miformulario = document.querySelector("#MyForm");
miformulario.addEventListener("submit", function(e){
    e.preventDefault();//El formulario se le bloquea para que no se envíe de manera normal
    let xhr = new XMLHttpRequest();
    xhr.open("POST", miformulario.action);
    xhr.send(new FormData(miformulario));
    xhr.onload = function(e){
        document.getElementById('respuesta').innerHTML = xhr.responseText;
    }
});
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

ENVIAR ARCHIVO CON AJAX USANDO FormData

Para enviar un archivo, solo debemos agregar a nuestro formulario un input de tipo **file**, utilizando **FormData**, nos será más sencillo.

```
<form action="/.php/procesar.php" method="POST" id="MyForm">
  <div class="row">
    <div class="col-sm-12 col-md-6 col-xl-6 mt-2">
      <label for="" class="form-label">Nombre:</label>
      <input placeholder="Ingrese su nombre aqui" type="text" class="form-control" name="nom" />
    </div>
    <div class="col-sm-12 col-md-6 col-xl-6 mt-2">
      <label for="" class="form-label">Apellidos:</label>
      <input placeholder="Ingrese sus Apellidos aqui" type="text" class="form-control" name="apellidos" />
    </div>
  </div>
  <div class="row">
    <div class="col-sm-12 col-xl-12">
      <input type="file" class="form-control mt-2" name="foto">
    </div>
  </div>

  <div class="row mt-2">
    <div class="col-12">
      <input type="submit" value="Iniciar sesion" class="btn btn-success form-control" />
    </div>
  </div>
</form>
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

ENVIAR ARCHIVO CON AJAX USANDO FormData

CÓDIGO JAVASCRIPT

```
let miformulario = document.querySelector("#MyForm");
miformulario.addEventListener("submit", function(e) {
    e.preventDefault();
    let xhr = new XMLHttpRequest();
    xhr.open("POST", miformulario.action);
    xhr.send(new FormData(miformulario));
    xhr.onload = function(e) {
        document.getElementById('respuesta').innerHTML =
        xhr.responseText;
        miformulario.reset();
    }
});
```

CÓDIGO PHP

```
$des = "../img/";
$dirTem = $_FILES["foto"]["tmp_name"];
$foto = $_FILES["foto"]["name"];
$nombre = $_POST['nom'];
$apellidos = $_POST['apellidos'];
$tlf = $_POST['tlf'];
$profesion = $_POST['Profesion'];

if (move_uploaded_file($dirTem, $des.$foto)) {
    echo "</br>se subio el archivo";
    $sql = "INSERT INTO empleado VALUES
    (',$nombre','$apellidos','$tlf','$profesion','$foto)";
    $result = $conn->query($sql);
    if ($result) {
        echo "</br>Datos ingresados Exitosamente";
    }
} else {
    echo "</br>No se pudo subir el archivo ".$foto;
}
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

SELECCIONAR DATOS CON AJAX

CÓDIGO JAVASCRIPT

```
function seleccion() {  
    let xhr = new XMLHttpRequest();  
    xhr.open("GET", "./php/seleccion.php", true);  
    xhr.onreadystatechange = function(){  
        if (xhr.readyState == 4 && xhr.status == 200) {  
            let datosJSON = JSON.parse(xhr.responseText);  
            let tableBody = document.getElementById('tableBody');  
            datosJSON.forEach(empleado => {  
                tableBody.innerHTML += `  
                    <tr>  
                        <td>  
                              
                        </td>  
                        <td>${empleado.Nombre}</td>  
                        <td>${empleado.Apellidos}</td>  
                        <td>${empleado.Telefono}</td>  
                        <td>${empleado.Profesion}</td>  
                        <td><button onclick=actualizar(${empleado.ID_Emple}) class="btn btn-primary"><i  
                            class="fa fa-edit"></i></button></td>  
                        <td><button onclick=confirmar('${empleado.ID_Emple}','${empleado.Nombre}') class="btn btn-danger"><i  
                            class="fa fa-trash"></i></button></td>  
                    </tr>  
                `;  
            });  
        }  
    }  
    xhr.send();  
}
```

ENVIAR LOS DATOS DE UN FORMULARIO POR AJAX USANDO XMLHttpRequest

SELECCIONAR DATOS CON AJAX

CÓDIGO PHP

```
require "./conex.php";  
$json = array();  
$sql = "SELECT * FROM empleado";  
$resul = $conn->query($sql);  
while ($datos = $resul->fetch_assoc()) {  
    $json[] = $datos;  
}  
echo json_encode($json);
```