-

TEMA 05. Conexión a base de datos en PHP

Desarrollo web entorno servidor
CFGS DAW

Autor: Francisco Aldarias Raya -
paco.aldarias@ceedcv.es
Revisado por: Vanessa Tarí –
vanessa.tari@ceedcv.es

Versión:031220.1608

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Revisiones.

- Añadida línea de $query a null cuando se cierra la conexión. Vanessa Tarí – 03/12/20
- Cambio del paso de parámetros con bindParam y parámetros posicionales. Vanessa Tarí – 03/12/20

Contenido

UD05. Conexión a base de datos con PHP

1. ESTRUCTURA DE UNA APLICACIÓN WEB CON BASES DE DATOS

Hasta el momento hemos estudiado la sintaxis y funcionamiento de PHP, abarcando el envío, recepción, y procesamiento de datos. Sin embargo, aún nos queda pendiente una parte fundamental de la mayoría de las aplicaciones que encontraremos: el almacenamiento de datos.

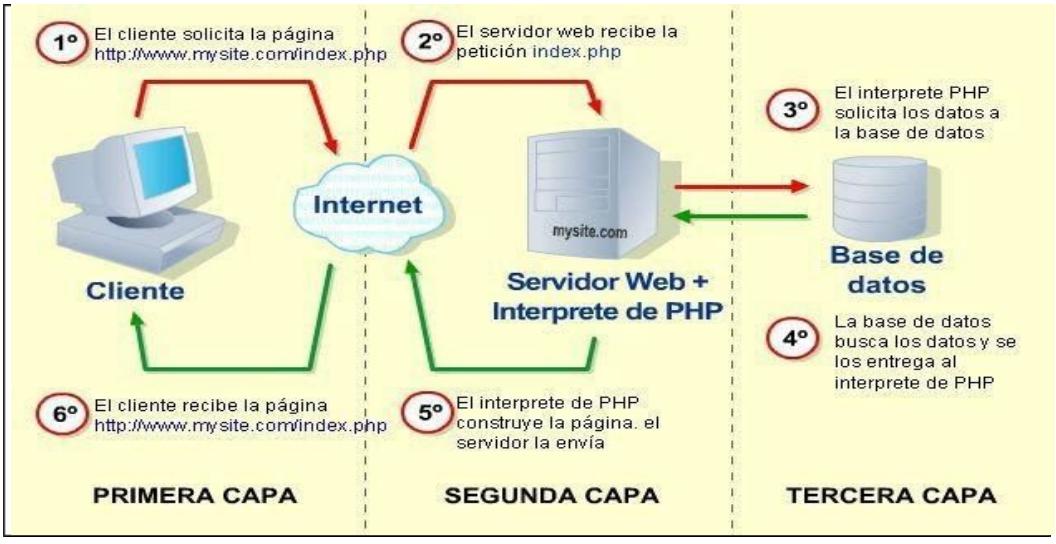
Si bien hemos aprendido a acceder a un fichero de texto para leer y guardar información, este mecanismo afectará gravemente al rendimiento de nuestra aplicación cuando tengamos que tratar con grandes cantidades de información.

Con este propósito, disponemos de software especializado llamado **Sistema de Gestión de Bases de Datos, SGBD**. Se trata de un software que, como su nombre indica, gestiona el acceso y uso de las bases de datos (en adelante DB).

El uso de las DB resulta fundamental para la mayoría de aplicaciones de un tamaño considerable. Entre otras aplicaciones podríamos destacar las siguientes: tienda virtual, catálogo en *Internet*, oficina/secretaría virtual, gestor de contenidos.

Al añadir bases de datos a una aplicación web se amplía el esquema que interviene en la generación de una página web con un nuevo elemento, y por tanto una capa nueva.

El funcionamiento es el siguiente:



A esta disposición se la denomina arquitectura web de tres capas:

1. Capa primera: cliente web (navegador).
2. Capa segunda: servidor web con intérprete de *PHP*.

3. Capa tercera: sistema gestor de bases de datos.

El sistema gestor de bases de datos puede estar instalado en el mismo equipo que el servidor web o en uno diferente. En nuestro entorno, asumiendo que todos usamos LAMP/WAMP/XAMPP, tenemos el servidor *Apache* y la base de datos *MySQL* ubicados en el mismo ordenador.

2. CONTROLADORES DE MYSQL PARA PHP

2.1 Versiones

Dependiendo de la versión de PHP, existen dos o tres API de PHP para acceder a las bases de datos de MySQL. Los usuarios de PHP 5 pueden elegir entre la extensión mysql obsoleta, mysqli, o PDO_MySQL. PHP 7 elimina la extensión mysql, dejando solamente las últimas dos opciones.

2.2 Elegir una API

PHP ofrece tres API diferentes para conectarse a MySQL. Abajo se muestran las API proporcionadas por las extensiones mysql, mysqli, y PDO (PHP Data Object). Cada trozo de código crea una conexión al servidor de MySQL que se está ejecutando en "ejemplo.com" con el nombre de usuario "usuario" y la contraseña "contraseña". También se ejecuta una consulta para saludar al usuario.

Ejemplo: Comparación de las tres API de MySQL

```php
<?php
// mysql.Obsoleta
$c = mysql_connect("ejemplo.com", "usuario", "contraseña");
mysql_select_db("basedatos");
$resultado = mysql_query("SELECT '¡Hola, querido usuario de MySQL!' AS _message FROM
DUAL");
$fila = mysql_fetch_assoc($resultado);
echo htmlentities($fila['_message']);
// mysqli. No OO.
$mysqli = new mysqli("ejemplo.com", "usuario", "contraseña", "basedatos");
$resultado = $mysqli->query("SELECT '¡Hola, querido usuario de MySQL!' AS _message FROM
DUAL");
$fila = $resultado->fetch_assoc();
echo htmlentities($fila['_message']);
// PDO. PHP OO.
$pdo = new PDO('mysql:host=ejemplo.com;dbname=basedatos', 'usuario', 'contraseña');
$sentencia = $pdo->query("SELECT '¡Hola, querido usuario de MySQL!' AS _message FROM
DUAL");
$fila = $sentencia->fetch(PDO::FETCH_ASSOC);
echo htmlentities($fila['_message']);
```

2.3 API recomendada

Se recomienda usar las extensiones mysqli o PDO_MySQL. No se recomienda usar la extensión mysql antigua para nuevos desarrollos, ya que ha sido declarada obsoleta en PHP 5.5.0 y eliminada en PHP 7.

Las principales ventajas de PDO es que está soportada por muchas bases de datos (MySQL, MSSQL, Informix, PostgreSQL), es orientado a objetos y permite usar consultas preparadas para poder mejorar la seguridad y evitar ataques de inyección de SQL.

> Atención
>
> Utilizaremos PDO que es compatible con PHP 5, PHP7

3. EXTENSIÓN PDO (PHP DATA OBJECT)

La extensión PDO (PHP Data Objects) permite acceder a distintas bases de datos utilizando las mismas funciones, lo que facilita la portabilidad. En PHP 5 existen drivers para acceder a las bases de datos más populares (MySQL, Oracle, MS SQL Server, PostgreSQL, SQLite, Firebird, DB2, Informix, etc).

La velocidad, PDO esta compilado en C/C++, por los que ofrece mayor rapidez de ejecución que sus competidoras (de código interpretado). Por otra parte: viene incorporado en PHP5 (es la solución “oficial” y en continua evolución) y desde PHP 5.1 viene “enabled” por defecto. De esta manera no tienes que importar ni cargar código de terceros cuando quieres usarlo.

PDO está completamente orientado a objetos, lo que facilita su uso frente a la necesidad de usar diferentes funciones de las API de cada sistema de bases de datos.

PDO actualmente soporta: Microsoft SQL Server, Firebird, IBM Informix, MySQL, Oracle, ODBC, PostgreSQL, SQLite.

Tenéis la definición de la clase PDO en esta dirección. http://php.net/manual/es/class.pdo.php

3.1 Conexiones y su administración

Las conexiones se establecen creando instancias de la clase base PDO. No importa el controlador que se utilice; siempre se usará el nombre de la clase PDO. El constructor acepta parámetros para especificar el origen de la base de datos (conocido como DSN) y, opcionalmente, el nombre de usuario y la contraseña (si la hubiera).

Ejemplo: Conectarse a MySQL

```php
<?php
$mbd = new PDO('mysql:host=localhost;dbname=prueba', $usuario, $contraseña);
?>
```

Si hubiera errores de conexión, se lanzará un objeto *PDOException*. Se puede capturar la excepción si fuera necesario manejar la condición del error, o se podría optar por dejarla en manos de un manejador de excepciones global de aplicación configurado mediante set_exception_handler().

Ejemplo: Manejo de errores de conexión

```php
<?php
try {
    $mbd = new PDO('mysql:host=localhost;dbname=prueba', $usuario, $contraseña);
    foreach ($mbd->query('SELECT * from FOO') as $fila) {
        print_r($fila);
    }
    $mbd = null;
} catch (PDOException $e) {
    print "¡Error!: " . $e->getMessage() . "<br/>";
    die();
}
```

En el caso de MySQL, la creación de la clase PDO incluye el nombre del servidor, el nombre de usuario y la contraseña.

Para poder acceder a MySQL mediante PDO, debe estar activada la extensión php_pdo_mysql en el archivo de configuración php.ini, si utilizáis XAMMP ya está activada.

Ejemplo: Función de conexión con la BD MYSQL

```php
<?php
function conectar_db()
{
    //uso de las excepciones en php try y catch
    try {
        $db = new PDO("mysql:host=localhost", "root", "");
        $db->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true);
        return($db);
    } catch (PDOException $e) {
        cabecera("Error grave");
        print "<p>Error: No puede conectarse con la base de datos.</p>\n";
        print "<p>Error: " . $e->getMessage() . "</p>\n";
        exit();
    }
```

```
}
```

Si la aplicación no captura la excepción lanzada por el constructor de PDO, la acción predeterminada es la de finalizar el script y mostrar información de rastreo. Esta información probablemente revelará todos los detalles de la conexión a la base de datos, incluyendo el nombre de usuario y la contraseña. Es su responsabilidad capturar esta excepción, ya sea explícitamente (con una sentencia catch) o implícitamente por medio de set_exception_handler().

Más detalles en: http://php.net/manual/es/pdo.connections.php

La forma de conectarse a otro sistema de bases de datos es muy similar.

Ejemplo: Conectarse a PostgreSQL

```
<?php
$mbd = new PDO('pgsql:host=localhost;dbname=prueba', $usuario, $contraseña);
?>
```

3.2 Desconexión con la base de datos

Para desconectar con la base de datos hay que destruir el objeto PDO. Si no se destruye el objeto PDO, PHP lo destruye al terminar la página.

```
$db = NULL;
```

4. CONSULTAS

Una vez realizada la conexión a la base de datos, las operaciones se realizan a través de consultas.

El método para efectuar consultas es PDO->query($consulta), que devuelve el resultado de la consulta. Dependiendo del tipo de consulta, el dato devuelto debe tratarse de formas distintas.

4.1 Consulta que no devuelve registros

La función query devuelve TRUE si tiene éxito, o FALSE en caso contrario. Si una consulta no devuelve registros devolverá FALSE. Si un registro no ha podido insertarse devolverá FALSE.

Ejemplo: Consulta de inserción

```
<?php
$db = conectar_db();
$consulta = "INSERT INTO $dbTabla (nombre, apellidos) VALUES ('$nombre',
'$apellidos')";
```

```
 if ($db->query($consulta)) {
     print "<p>Registro creado correctamente.</p>\n";
 } else {
     print "<p>Error al crear el registro.<p>\n";
 }
$db = null;
```

4.2 Consulta que devuelve registros

Si la consulta devuelve registros, el método devuelve los registros correspondientes. En ese caso sí que es conveniente guardar lo que devuelve el método en una variable para procesarla posteriormente. Si contiene registros, la variable es de un tipo especial llamado recurso que no se puede acceder directamente, pero que se puede recorrer con un bucle foreach().

Ejemplo: Consulta de selección

```
<?php
$db = conectar_db();
$consulta = "SELECT * FROM $dbTabla";
$result = $db->query($consulta);
if (!$result) {
    print "<p>Error en la consulta.</p>\n";
} else {
    foreach ($result as $valor) {
        print "<p>$valor[nombre] $valor[apellidos]</p>\n";
    }
}
```

En los ejemplos, la consulta se realiza en dos líneas, pero podría estar en una sola:

```
<?php
// En dos líneas
$consulta = "SELECT * FROM $dbTabla";
$result = $db->query($consulta);
// En una sola línea
$result = $db->query("SELECT * FROM $dbTabla");
```

No hay motivo para preferir la primera versión, salvo que se quiera imprimir la consulta mientras se está programando para comprobar que no tiene errores:

```php
<?php
$consulta = "SELECT * FROM $dbTabla";
print "<p>Consulta: $consulta</p>\n";
$result = $db->query($consulta);
```

4.3 Consulta SELECT

Para obtener registros que cumplan determinados criterios se utiliza la consulta SELECT.

Ejemplo: Select

```php
<?php
$db = conectar_db();
$consulta = "SELECT * FROM $dbTabla";
$result = $db->query($consulta);
if (!$result) {
    print "<p>Error en la consulta.</p>\n";
} else {
    print "<p>Consulta ejecutada.</p>\n";
}
$db = null;
```

Para acceder a los registros devueltos por la consulta, se puede utilizar un bucle foreach.

Ejemplo: Select con Foreach

```php
<?php
$db = conectaDb();
$consulta = "SELECT COUNT(*) FROM $dbTabla";
$result = $db->query($consulta);
if (!$result) {
    print "<p>Error en la consulta.</p>\n";
} elseif ($result->fetchColumn() == 0) {
    print "<p>No se ha creado todavía ningún registro en la tabla.</p>\n";
} else {
    $consulta = "SELECT * FROM $dbTabla";
    $result = $db->query($consulta);
    if (!$result) {
        print "<p>Error en la consulta.</p>\n";
    } else {
        foreach ($result as $valor) {
            print "<p>Nombre: $valor[nombre] - Apellidos:
```

```
    $valor[apellidos]</p>\n";
        }
    }
}
```

O también se puede utilizar la función PDOStatement->fetch()

Ejemplo: Select con fetch

```
<?php
$db = conectaDb();
$consulta = "SELECT * FROM $dbTabla";
$result = $db->query($consulta);
if (!$result) {
    print "<p>Error en la consulta.</p>\n";
} else {
    while ($fila = $result->fetch()) {
        print "<pre>\n";
        print_r($fila);
        print "</pre>\n";
    }
}
$result = null;
$db = null;
```

El problema es que, si la consulta no devuelve ningún registro, estos dos métodos no escribirían nada. Por ello se recomienda hacer primero una consulta que cuente el número de resultados de la consulta y, si es mayor que cero, hacer la consulta.

El ejemplo siguiente utiliza la función **PDOStatement->fetchColumn(),** que devuelve la primera columna del primer resultado (que en este caso contiene el número de registros de la consulta).

Ejemplo: Select con fetchColumn

```
$<?php
$db = conectaDb();
$consulta = "SELECT COUNT(*) FROM $dbTabla";
$result = $db->query($consulta);
if (!$result) {
    print "<p>Error en la consulta.</p>\n";
} elseif ($result->fetchColumn() == 0) {
    print "<p>No se ha creado todavia ningun registro en la tabla.</p>\n";
```

```
} else {
    $result = $db->query($consulta);
    if (!$result) {
        print "<p>Error en la consulta.</p>\n";
    } else {
        foreach ($result as $valor) {
            print "<p>Nombre: $valor[nombre] - Apellidos:
    $valor[apellidos]</p>\n";
        }
    }
}
$db = null;
```

Una opción más óptima sería haciendo uso de fecthAll()

```
<?php
$result=$db->query("Select * from $dbTabla ");
$resultSet=$result->fetchAll();
if (count($resultSet)==0) {
    print "<p>Error en la consulta.</p>\n";
} else {
    foreach ($result as $valor) {
        print "<p>Nombre: $valor[nombre] - Apellidos:
    $valor[apellidos]</p>\n";
    }
}
```

A continuación, vemos las consultas preparadas:

Ejemplo: Select de dos tablas

```
$db = conectar_db();
$consulta = "SELECT $dbPrestamos.id AS id, $dbUsuarios.nombre as nombre,
$dbUsuarios.apellidos as apellidos, $dbObras.titulo as titulo,
$dbPrestamos.prestado as prestado, $dbPrestamos.devuelto as devuelto FROM $dbPrestamos,
$dbUsuarios, $dbObras
WHERE $dbPrestamos.id_usuario=$dbUsuarios.id AND
$dbPrestamos.id_obra=$dbObras.id and $dbPrestamos.devuelto='0000-00-00' ORDER BY $campo
$orden";
$result = $db->query($consulta);
if (!$result) {
print "<p>Error en la consulta.</p>\n";
} else {
...
}
$result = NULL;
$db = NULL;
```

4.4 Comparación de fetch

Seguidamente se van a comparar tipos de fetch:

- **Fetch():** Recupera una fila con sus columnas en un vector cuyas campos son sus columnas.
- **fetch(PDO::FETCH_OBJ)** recupera un objeto con atributos del nombre de la columna de la tabla. Sólo trae una fila.
- **fetchAll(PDO::FETCH_OBJ);** igual que el anterior pero trae un array de objetos genéricos.

Ejemplo: Función que devuelve un vector de objetos.

```
<?php
public function getProfesores() {
    $db = conectar_db();
    $consulta = "SELECT * FROM profesor";
    $result = $db->query($consult);
    $profesores = array();
    $cont = 0;
    /* Opcion1
    while ($fila = $result->fetch()) {
    $profesor = new Profesor($fila["id"], $fila["nombre"]);
    $profesores[$cont] = $profesor;
    $cont++;
    }
    */
    /* Opcion2
    $cont = 0;
    $fila = $result→fetch(PDO::FETCH_OBJ); while ($fila) {
    $profesores[$cont] = new Profesor($fila->id, $fila->nombre);
    $fila = $result->fetch(PDO::FETCH_OBJ)
    $cont++;
    }
    */
    // Opción 3.
    $filas = $result->fetchAll(PDO::FETCH_OBJ);
    //print_r($filas);
    foreach ($filas as $fila) {
        //echo $fila->id . "<br>";
        $profesores [$cont] = new Profesor($fila->id, $fila->nombre);
```

```
        $cont++;
    }

    return $profesores;
}
```

5. SEGURIDAD EN LAS CONSULTAS: CONSULTAS PREPARADAS

Para evitar **ataques de inyección SQL**, son un tipo de ataque a una aplicación que aprovecha fallos de seguridad en la base de datos (lo veremos con más detalle en la unidad de seguridad), se recomienda el uso de sentencias preparadas, en las que PHP se encarga de "desinfectar" los datos en caso necesario.

5.1 Consultas preparadas

El método para efectuar consultas es primero preparar la consulta con **PDO->prepare($consulta)** y después ejecutarla con **PDO→execute(array(parámetros))**, que devuelve el resultado de la consulta. Dependiendo del tipo de consulta, el dato devuelto debe tratarse de formas distintas, como se ha explicado en el apartado anterior.

El siguiente ejemplo muestra cómo se realizaría una consulta:

```
<?php
$result = $db->prepare("SELECT * FROM $dbTabla");
$result->execute();
```

Si la consulta incluye datos introducidos por el usuario, los datos pueden incluirse directamente en la consulta, pero en ese caso, PHP no realiza ninguna "desinfección" de los datos, por lo que estaríamos corriendo riesgos de ataques:

```
<?php
$nombre= $_REQUEST["nombre"];
$consulta = "SELECT COUNT(*) FROM $dbTabla WHERE nombre=$nombre AND apellidos=$apellidos";
// DESACONSEJADO: PHP NO DESINFECTA LOS DATOS
$result = $db->prepare($consulta);
$result->execute();
if (!$result) {
    print "<p>Error en la consulta.</p>\n";
}
```

Para que PHP desinfecte los datos, estos deben enviarse al ejecutar la consulta, no al prepararla. Para ello es necesario indicar en la consulta la posición de los datos. Esto se puede hacer mediante interrogantes o mediante parámetros.

5.2 Envío de datos mediante bindParam

Ejemplo #1 Inserciones repetidas utilizando sentencias preparadas

Este ejemplo realiza dos consultas INSERT sustituyendo name y value por los marcadores correspondientes.

```
<?php
$sentencia = $mbd->prepare("INSERT INTO REGISTRY (name, value) VALUES
(:name, :value)");
$sentencia->bindParam(':name', $nombre);
$sentencia->bindParam(':value', $valor);

// insertar una fila
$nombre = 'uno';
$valor = 1;
$sentencia->execute();

// insertar otra fila con diferentes valores
$nombre = 'dos';
$valor = 2;
$sentencia->execute();
?>
```

5.3 Envio de datos mediante marcadores posicionales ‘?’

Ejemplo #2 Inserciones repetidas utilizando sentencias preparadas

Este ejemplo realiza dos consultas INSERT sustituyendo name y value por los marcadores posicionales '?'.

```
<?php
$sentencia = $mbd->prepare("INSERT INTO REGISTRY (name, value) VALUES (?,
?)");
$sentencia->bindParam(1, $nombre);
$sentencia->bindParam(2, $valor);

// insertar una fila
$nombre = 'uno';
$valor = 1;
$sentencia->execute();

// insertar otra fila con diferentes valores
$nombre = 'dos';
$valor = 2;
```

```
$sentencia->execute();
?>
```

6. CONSULTAS DE CREACIÓN, INSERCIÓN Y BORRADO

6.1. Creación y borrado de una base de datos

Para crear una base de datos, se utiliza la consulta **CREATE DATABASE.**

```
<?php
// EJEMPLO DE CONSULTA DE CREACION DE BASE DE DATOS
$db = conectaDb();
$consulta = "CREATE DATABASE $dbDb";
if ($db->query($consulta)) {
    print "<p>Base de datos creada correctamente.</p>\n";
} else {
    print "<p>Error al crear la base de datos.</p>\n";
}
$db = null;
```

Para borrar una base de datos, se utiliza la consulta **DROP DATABASE**.

```
<?php
// EJEMPLO DE CONSULTA DE BORRADO DE BASE DE DATOS
$db = conectaDb();
$consulta = "DROP DATABASE $dbDb";
if ($db->query($consulta)) {
    print "<p>Base de datos borrada correctamente.</p>\n";
} else {
    print "<p>Error al borrar la base de datos.</p>\n";
}
$db = null;
```

6.2. Creación y borrado de una tabla

Para crear una tabla, se utiliza la consulta **CREATE TABLE.**

```
<?php
// EJEMPLO DE CONSULTA DE CREACIÓN DE TABLA EN MYSQL
$db = conectaDb();
$consulta = "CREATE TABLE $dbTabla (id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
nombre VARCHAR($tamNombre),
apellidos VARCHAR($tamApellidos),
PRIMARY KEY(id)
```

```
)";
if ($db->query($consulta)) {
    print "<p>Tabla creada correctamente.</p>\n";
} else {
    print "<p>Error al crear la tabla.</p>\n";
}
$db = null;
```

Para borrar una tabla, se utiliza la consulta **DROP TABLE.**

```
<?php
// EJEMPLO DE CONSULTA DE BORRADO DE TABLA
$db = conectaDb();
$consulta = "DROP TABLE $dbTabla";
if ($db->query($consulta)) {
    print "<p>Tabla borrada correctamente.</p>\n";
} else {
    print "<p>Error al borrar la tabla.</p>\n";
}
$db = null;
```

6.3. Inserción, actualización y eliminación de datos en una tabla

Para añadir un registro a una tabla, se utiliza la consulta **INSERT INTO.**

```
// EJEMPLO DE CONSULTA DE INSERCION DE REGISTRO
$db = conectaDb();
$nombre = recoge("nombre");
$apellidos = recoge("apellidos");
$consulta = "INSERT INTO $dbTabla (nombre, apellidos) VALUES (:nombre,
:apellidos)";
$result = $db->prepare($consulta);
if ($result->execute(array(":nombre" => $nombre, ":apellidos" => $apellidos))) {
  print "<p>Registro creado correctamente.</p>\n";
} else {
  print "<p>Error al crear el registro.</p>\n";
}
$db = NULL;
```

Para modificar un registro a una tabla, se utiliza la consulta **UPDATE**.

```
// EJEMPLO DE CONSULTA DE MODIFICACION DE REGISTRO
$db = conectaDb();
```

```
$nombre = recoge("nombre");
$apellidos = recoge("apellidos");
$id= recoge("id");
$consulta = "UPDATE $dbTabla SET nombre=:nombre, apellidos=:apellidos WHERE
id=:id";
$result = $db->prepare($consulta);
if ($result->execute(array(":nombre" => $nombre, ":apellidos" =>$apellidos,":id"
=> $id))) {
  print "<p>Registro modificado correctamente.</p>\n";
else {
  print "<p>Error al modificar el registro.</p>\n";
}
$db = NULL;
```

Para borrar un registro de una tabla, se utiliza la consulta **DELETE FROM**.

En el ejemplo, los registros a borrar se reciben en forma de matriz y se recorre la matriz borrando un elemento en cada iteración.

```
// EJEMPLO DE CONSULTA DE BORRADO DE REGISTRO
$db = conectaDb();
$id = recogeMatriz("id");
foreach ($id as $indice => $valor) {
$consulta = "DELETE FROM $dbTabla WHERE id=:indice";
$result = $db->prepare($consulta);
if ($result->execute(array(":indice" => $indice))) {
  print "<p>Registro borrado correctamente.</p>\n";
} else {
  print "<p>Error al borrar el registro.</p>\n";
}}
$db = NULL;
```

7. BIBLIOGRAFÍA

1. Bonilla, H. (2013): *Desarrollo web en entorno servidor*, CEEDCV.
2. Cristina Alvarez (2014): *Desarrollo web en entorno servidor*, CEEDCV.
3. López, M.; Vara, JM; Verde, J.; Sánchez, D.M.; Jiménez, J.J.; Castro, V. (2012): *Desarrollo web en entorno servidor*, RA-MA, Madrid
4. Martínez, V.; Guarinos, J.J. (): *Curso de programación en PHP y MySQL*, "Bloque V: Acceso a MySQL desde PHP", CEFIRE de Castelló.
5. PDO vs. MySQLi: Which Should You Use? https://code.tutsplus.com/tutorials/pdo-vs-mysqli-which-should-you-use--net-24059
6. 7 More Mistakes PHP Developers Often Make. https://www.sitepoint.com/7-mistakes-

commonly-made-php-developers/