

Cómo crear apps con PHP y MariaDB

1 Aplicaciones de servidor y una base de datos	1
2 Elementos comunes	2
2.1 Tablas para las consultas select	2
2.2 Creación de formularios en HTML	2
2.2.1 Campos de texto y numéricos.	2
2.2.2 Importancia del atributo name e id en los campos de texto y numéricos	2
2.2.3 La etiqueta form y sus atributos method y action	3
2.3 Código PHP para leer lo que se envía desde un formulario PHP	3
2.4 Código PHP para hacer consultas SQL	4
2.4.1 La clase DatabaseService	4
3 Elementos particulares	5
3.1 Una consulta (select)	6
3.2 Una consulta donde buscamos o filtramos información (select).	7
3.3 Una inserción (insert)	8
3.3 Un borrado (delete).	10
Paso 1. Mostrar el contenido a borrar	10
Paso 2. Creamos el fichero que realmente borra: action-delete.php	11
3.3 Una modificación (update)	14
Paso 1. Mostrar el contenido a modificar con un botón para modificar	14
Paso 2. Formulario para modificar	15
Paso 3. Creamos el fichero que realmente modifica: action-update-cliente.php	17
4 Esquema general de cualquier aplicación	18
5 Diagrama de flujo para cualquier aplicación	22

1 Aplicaciones de servidor y una base de datos

En general, las aplicaciones de servidor son las que utilizan un lenguaje de programación como PHP, que se ejecuta en el servidor web. Y suelen conectarse a una base de datos, en nuestro caso, MariaDB.

En este tipo de aplicaciones con una gran probabilidad tenemos una página web con un formulario HTML donde viene información que se va procesar. Cualquier gestión que se haga suele caer en una de estas categorías.

1. Una consulta (select)
2. Una consulta donde *buscamos* o *filtramos* información (select).
3. Una inserción (insert)
4. Un borrado (delete).
5. Una modificación (update).

Si hacemos un análisis vemos que las operaciones típicas de SQL: select, insert, delete y update cubren todas las necesidades.

Vamos a estructurar este documento de la siguiente manera:

1. Vamos a ver una serie de elementos comunes que nos sirven para todas las categorías anteriores como son: crear formularios, leer el contenido de esos formularios desde PHP y cómo llamar a la base de datos.
2. Después atacaremos cada uno de los problemas anteriores de uno en uno.

2 Elementos comunes

2.1 Tablas para las consultas select

Hay que repasar cómo se crea una tabla en HTML ya que las consultas select requieren la creación de tablas con mucha probabilidad.

https://www.w3schools.com/html/html_tables.asp

2.2 Creación de formularios en HTML

Cuando la consulta SQL no es un select, tenemos que construir un formulario HTML.

Por lo tanto, es importante saber cómo crear formularios en PHP. Tienes un capítulo entero dedicado a esto aquí: https://www.w3schools.com/html/html_forms.asp

Principalmente en IAW nosotros vamos necesitar únicamente los siguientes detalles, aunque lógicamente se puede ampliar.

2.2.1 Campos de texto y numéricos.

En los ejercicios que vamos a hacer principalmente tenemos texto y números. Por lo tanto, los únicos campos que necesitamos para un formulario son esos dos:

- `<input type="text">`
- `<input type="number">`

2.2.2 Importancia del atributo name e id en los campos de texto y numéricos

Cada vez que pongas un campo en un formulario **pon siempre** los atributos *name* e *id* con el mismo valor.

Por ejemplo, voy a poner un campo en un formulario para recoger el nombre de una persona:

```
<input type="text" name="nombre" id="nombre">
```

Otro ejemplo, voy a poner un campo en un formulario para recoger el stock que queda de un determinado producto.

```
<input type="number" name="stock" id="stock">
```

2.2.3 La etiqueta form y sus atributos method y action

Aquí tienes un ejemplo de un formulario. Nosotros vamos a usar dos propiedades de los formularios HTML:

- `method="post"`. Aunque esto no es obligatorio, **nosotros vamos a usar esto siempre para que los datos viajen sin cambiar la URL de nuestro sitio web.**
- `action=" [fichero.php]"`. Este atributo tampoco es obligatorio.
 - Si no se pone, cuando se pulse en el botón “submit” del formulario, se recarga la misma página enviando los datos del formulario.
 - Si sí se pone, los datos del formulario se envían al fichero que esté en el atributo.

En el siguiente ejemplo, tenemos que cuando se haga clic en el botón de enviar (el input de tipo *submit*) los datos del formulario (un nombre y un apellido) se van a enviar mediante *post* al fichero que se llama *action_page.php*.

```
<form action="/action_page.php" method="post">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

En este otro ejemplo

```
<form action="/action_page.php" method="post">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

2.3 Código PHP para leer lo que se envía desde un formulario PHP

Nosotros siempre vamos a utilizar *method="post"* en nuestros formularios y una serie de campos. Tenemos que comprobar en nuestros ficheros PHP que se está utilizando *post* y que además los campos que estamos esperando están llegando.

Por tanto, en aquellos ficheros PHP que procesen formularios, tenemos que poner una condición que compruebe que aquellos datos están llegando. Esto lo hacemos de la siguiente manera:

```
<?php
if ($_SERVER["REQUEST_METHOD"] === "POST"
    && isset($_POST["id"]))
```

```

    && isset($_POST["nombre"])
    && isset($_POST["correo"])
    && isset($_POST["fechaRegistro"])) {
...
    // Código de nuestra aplicación
}
else {
    echo "No has enviado la información suficiente.";
}

```

Fíjate que tenemos

- una gran condición: el “if”.
- una comparación que dice esto: `$_SERVER["REQUEST_METHOD"] === "POST"` la cual siempre tiene que estar porque es la que nos asegura que la información llega por el método post.
- Un conjunto de funciones `isset()` unidas por el operador `&&`. Esto hace que se compruebe que están llegando todos los campos que necesitamos.
 - Tenemos que poner una función `isset()` por cada parámetro de formulario que esperemos.
 - En el ejemplo anterior estamos esperando del formulario HTML los siguientes parámetros: id, nombre, correo y fechaRegistro.
 - Como todos los campos anteriores tienen que llegar por el método post, tenemos que leerlas del array asociativo `$_POST[]` y de ahí, la forma que tiene el código anterior.

2.4 Código PHP para hacer consultas SQL

2.4.1 La clase DatabaseService

2.4.1.1 Resumen rápido

Lo único que tienes que hacer es tener un fichero como este en tu carpeta donde estés trabajando y con la configuración correcta para la base de datos:

- **host: ip donde encontramos la base de datos**
- **db_name: nombre de la base de datos**
- **db_user: nombre del usuario que tiene permisos sobre la base de datos**
- **db_password: contraseña del usuario.**

2.4.1.2 Explicación larga

En todo nuestro código PHP vamos a tener un fichero que se llama `DatabaseService.php`. Este fichero contiene la información de la conexión con la base de datos de forma que la escribamos una única vez y no tengamos que repetirla más veces.

```

1  <?php
2  /**
3   * DatabaseService.php
4   * Servicio para la conexión a la base de datos
5   */
6  class DatabaseService {

```

```

7   // Aquí ponemos la configuración de la base de datos
8   private $db_host = "localhost";
9   private $db_name = "tienda";
10  private $db_user = "usuario";
11  private $db_password = "usuario";
12
13  // Variable para guardar la conexión
14  private $connection = null;
15
16  /**
17   * Obtiene la conexión a la base de datos
18   */
19  public function getConnection() {
20
21      if ($this->connection == null) {
22          $this->connection = new
PDO("mysql:host=$this->db_host;dbname=$this->db_name", $this->db_user,
$this->db_password);
23      }
24
25      return $this->connection;
26  }
27 }
```

Puntos clave:

- Línea 6: Definición de la clase *DatabaseService*. Esta clase nos va a servir para guardar la información de la conexión con la base de datos (líneas 8, 9, 10 y 11) y no tener que repetirla más veces que esta.
 - El uso de clases y objetos (conceptos de la programación orientada a objetos implica utilizar el operador *new* que lo verás algunas veces).
- Línea 19. Función *getConnection()*. En cualquier lenguaje de programación tenemos que realizar una conexión con la base de datos con la que trabajamos. En este caso, tenemos esta función que nos lo permite.
 - La función *getConnection()* utiliza un atributo llamado *\$connection* definido en la línea 14.
 - La primera vez que llamamos a la función *getConnection()* el atributo *\$connection* es *null*. Por eso en la línea 21, si esa conexión *\$connection* es nula (no se ha realizado nunca), la creamos con el código que vemos en la línea 22.
- Línea 22. Cadena de conexión. En PHP se utiliza la clase PDO (PHP Data Object) para realizar conexiones a diferentes bases de datos. En concreto, la cadena de conexión para *MariaDB* comienza con “*mysql:…*” y continúa indicando dónde está la base de datos (*\$db_host*), la base de datos a la que nos conectamos (*\$dbname*), el usuario (*\$db_user*) y la contraseña (*\$db_password*).

3 Elementos particulares

Para cerrar este tema, tenemos los 5 tipos de aplicaciones que hemos definido antes.

1. Una consulta (*select*)
2. Una consulta donde *buscamos* o *filtramos* información (*select*).
3. Una inserción (*insert*)

4. Un borrado (delete).
5. Una modificación (update).

Vamos a analizarlos uno a uno.

3.1 Una consulta (select)

No tenemos más que crear un fichero que termine en .php e insertamos el código que definimos a continuación.

En primer lugar, ponemos el código que crea la conexión con la base de datos.

```
<?php

// Incluimos el servicio de base de datos, que nos permite conectarnos.
include_once 'DatabaseService.php';

// Creamos una instancia del servicio de base de datos.
$databaseService = new DatabaseService();

// Nos conectamos a la base de datos.
$connection = $databaseService->getConnection();
```

Después creamos la consulta select en una variable y la ejecutamos.

```
$query = "select * from una_tabla";
$stmt = $connection->query($query);
```

Finalmente, utilizando el objeto \$stmt comprobamos si ha habido error. Si no, mostramos el resultado de todas las filas consultadas.

Fíjate que la variable \$unaFila es una fila y que para acceder a las diferentes columnas tenemos que conocer la estructura de la tabla que estemos consultando. Para acceder al contenido de una columna concreta hay que escribirlo un poco raro, ya que se ponen unas llaves y dentro de ellas el nombre de la variable y entre corchetes y usando comillas el nombre de la columna

```
if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error";
}
else {
    $filas = $stmt->fetchAll();

    foreach ($filas as $unaFila) {
        echo "{$unaFila['col1']}";
        echo "{$unaFila['col2']}";
        ...
    }
}
```

3.2 Una consulta donde buscamos o filtramos información (select).

Una consulta donde buscamos o filtramos tiene una pinta, y principalmente es utilizar la palabra `where`.

```
select * from <una_tabla> where id = <un_id>
```

En este caso, necesitamos un formulario donde se indica el criterio de búsqueda o de filtro. Por ejemplo podemos tener lo siguiente, donde procuramos hacer un select filtro por un nombre, por lo tanto, tenemos que pedir ese nombre a la persona que utilice nuestro programa.

Esto podría estar contenido en un fichero que se llame `consultar-clientes-por-nombre.html`.

```
<form method="post" action="filtrar-cliente-por-nombre.php">
    <label for="nombre">El nombre comienza por:</label><br>
    <input type="text" id="nombre" name="nombre">
</form>
```

Continuamos creando el fichero PHP que procesa el anterior formulario, que se llama `filtrar-cliente-por-nombre.php`.

Como este último es un fichero que procesa un formulario hemos de comprobar que llega por el método `post` la información que estamos esperando.

Fíjate que hemos creado también una variable `$nombre` que lee ese contenido que llega por `post` al código PHP.

```
<?php

if ($_SERVER["REQUEST_METHOD"] === "POST"
    && isset($_POST["nombre"]))
{
    // Creamos las variables que nos permiten leer la información que
    // viene de $_POST
    $nombre = $_POST["nombre"];

    // Aquí se inserta el resto del código que vamos a ir comentando
}

else {
    echo "No has enviado la información suficiente.";
}
```

Dentro de la parte del `if` que es la que se ejecuta cuando todo está correcto ponemos el código que nos crea la conexión con la base de datos.

```
// ¡¡Estamos dentro del if!!

// Incluimos el servicio de base de datos, que nos permite conectarnos.
include_once 'DatabaseService.php';
```

```
// Creamos una instancia del servicio de base de datos.
$databaseService = new DatabaseService();

// Nos conectamos a la base de datos.
$connection = $databaseService->getConnection();
```

Después creamos la consulta select en una variable y la ejecutamos. Date cuenta que para evitar el SQL injection, hemos utilizado en la variable \$query la referencia “:nombre”, que tenemos que unir con la función *bindParam()* a la variable real \$nombre. Finalmente ejecutamos la consulta con el método *execute()* y obtenemos en la variable \$resultadoEjecucion qué ha sucedido tras la consulta de esta consulta.

```
// Continuamos con el código anterior.

$query = "select * from clientes where nombre = :nombre";
$stmt = $connection->prepare($query);
$stmt->bindParam(':nombre', $nombre);
$resultadoEjecucion = $stmt->execute();
```

Para terminar, comprobamos si ha habido error o no para mostrar un mensaje.

```
// Continuamos con el código anterior.

if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error";
}
else {
    $filas = $stmt->fetchAll();

    foreach ($filas as $unaFila) {
        echo "{$unaFila['col1']}";
        echo "{$unaFila['col2']}";
        ...
    }
}
```

3.3 Una inserción (insert)

Cuando queremos hacer una inserción en una base de datos mediante una página web tenemos que crear primero un formulario con los campos que se han de insertar. Por tanto, tenemos que analizar la estructura de la tabla con la que estamos trabajando.

En segundo lugar, creamos el formulario correspondiente. En este ejemplo, tenemos la tabla *Cliente* que solo tiene dos campos a insertar: email y nombre.

Este código podría estar en un fichero que se llame *insertar-cliente.html*.

```
<form method="post" action="insertar-cliente.php">
    <input type="text" id="nombre" name="nombre">
    <input type="text" id="email" name="email">
</form>
```

Después, creamos el fichero *insertar-cliente.php*, el cual tiene que leer los datos del formulario anterior, por lo tanto tenemos que comenzar el fichero de la siguiente manera, comprobando que llegan los datos que esperamos y además los leemos en variables.

```
<?php

if ($_SERVER["REQUEST_METHOD"] === "POST"
    && isset($_POST["nombre"])
    && isset($_POST["email"]))
) {
    // Creamos las variables que nos permiten leer la información que
    // viene de $_POST
    $nombre = $_POST["nombre"];
    $email = $_POST["email"];

    // Aquí se inserta el resto del código que vamos a ir comentando

}
else {
    echo "No has enviado la información suficiente.";
}
```

Continuamos escribiendo código dentro de la condición *if* debajo de haber leído las variables que llegan del formulario HTML.

```
// !!Estamos dentro del if!!

// Incluimos el servicio de base de datos, que nos permite conectarnos.
include_once 'DatabaseService.php';

// Creamos una instancia del servicio de base de datos.
$databaseService = new DatabaseService();

// Nos conectamos a la base de datos.
$connection = $databaseService->getConnection();
```

Ahora realizamos la consulta SQL que este caso es un *insert*. Fíjate que:

- cuando creamos la consulta *\$query* en los valores del nombre y email estamos poniendo *:nombre* y *:email* y no directamente las variables de PHP. Esta sustitución es así para evitar el problema de SQL injection.
- Después ejecutamos la función *prepare()* utilizando la sentencia SQL anterior.
- Utilizamos la función *bindParam()* para ahora sí, sustituir las variables *:nombre* y *:email* por las variables que tenemos *\$nombre* y *\$email* de PHP.
- Finalmente, ejecutamos el método *execute()* que realiza la operacion SQL.

```
$query = "insert into clientes (nombre, email) values (:nombre, :email);"
```

```
$stmt = $connection->prepare($query);
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':email', $email);

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

Finalmente comprobamos si ha habido un error y no y mostramos un mensaje.

```
if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error en la inserción";
}
else {
    /* No se ha producido un error */
    echo "Se ha ejecutado bien la inserción.";
}
```

3.3 Un borrado (delete).

Cuando queremos borrar algo de la base de datos tenemos que hacerlo en dos pasitos:

1. Mostrar el contenido del cual seleccionamos algo para borrar.
2. Elegir qué se va a borrar.

Paso 1. Mostrar el contenido a borrar

En este caso estamos en un *select*. Esto ya lo hemos explicado previamente. Solo hay una cosa que añadir que es un botón *Borrar*.

Por lo tanto seguimos todos los pasos que se correspondan con un *select* hasta llegar al punto donde tenemos este código:

```
$filas = $stmt->fetchAll();

foreach ($filas as $unaFila) {
    echo "{$unaFila['col1']}";
    echo "{$unaFila['col2']}";
    ...
}
```

Y lo sustituimos por este código.

```
<?php
$filas = $stmt->fetchAll();

foreach ($filas as $unaFila) {
    $clavePrimaria = $unaFila["id"];
?>
```

```

<form method="post"
      action="action-delete.php"
      <input type="hidden" name="id" value="<?= $clavePrimaria ?>">
      <button type="submit">Borrar</button>
    </form>

<?php
  }
?>
```

Fíjate que simplemente hemos añadido un formulario que tiene dos partes:

1. La clave primaria del elemento a borrar que se pasa como un elemento *hidden* al código PHP que haremos después que realmente borra.
2. Un botón de *submit* que pone *Borrar*.

Este formulario es como todos los que hemos hablado anteriormente: tiene en su atributo *method* el valor *post* y en el atributo *action* ponemos el nombre del fichero PHP que realmente va a borrar, en este caso, sería *action-delete.php*.

Nota.

Si estamos borrando podemos meter en el formulario este otro atributo que nos confirme si queremos borrar:

```
onsubmit="return confirm('¿Estás seguro de que deseas borrar este cliente?');">
```

Paso 2. Creamos el fichero que realmente borra: *action-delete.php*

Ahora creamos el fichero *action-delete.php* que es el que realmente borra. Este fichero es llamado desde un formulario, por lo tanto tenemos que hacer (como en los casos anteriores).

Primero, creamos el fichero *action-delete.php* con el contenido que comprueba si están realmente llegando los campos que queremos mediante *post*. En este caso, vamos a pasar únicamente la clave primaria de la tabla de la que queramos borrar una fila suponiendo que esa columna se llama *id*.

```

<?php

if ($_SERVER["REQUEST_METHOD"] === "POST"
  && isset($_POST["id"]))
{
  // Creamos las variables que nos permiten leer la información que
  // viene de $_POST
  $id = $_POST["id"];

  // Aquí se inserta el resto del código que vamos a ir comentando
```

```

    }
else {
    echo "No has enviado la información suficiente.";
}

```

Continuamos añadiendo dentro del *if* las líneas que nos permite conectarnos con la base de datos.

```

// ¡¡Estamos dentro del if!!

// Incluimos el servicio de base de datos, que nos permite conectarnos.
include_once 'DatabaseService.php';

// Creamos una instancia del servicio de base de datos.
$databaseService = new DatabaseService();

// Nos conectamos a la base de datos.
$connection = $databaseService->getConnection();

```

Ahora realizamos la consulta SQL que este caso es un *delete*. Fíjate que:

- cuando creamos la consulta *\$query* en el valor de la columna *id* estaremos poniendo *:id* y no directamente la variables PHP. Esta sustitución es así para evitar el problema de SQL injection.
- Después ejecutamos la función *prepare()* utilizando la sentencia SQL anterior.
- Utilizamos la función *bindParam()* para ahora sí, sustituir la variable *:id* por las variable que tenemos *\$id* de PHP.
- Finalmente, ejecutamos el método *execute()* que realiza la operacion SQL.

```

$query = "delete from clientes where id = :id";

$stmt = $connection->prepare($query);

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

$resultadoEjecucion = $stmt->execute();

```

Finalmente comprobamos si ha habido un error y no y mostramos un mensaje.

```

if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error en la inserción";
}
else {
    /* No se ha producido un error */
    echo "Se ha ejecutado bien la inserción.";
}

```

formulario con los campos que se han de insertar. Por tanto, tenemos que analizar la estructura de la tabla con la que estamos trabajando.

En segundo lugar, creamos el formulario correspondiente. En este ejemplo, tenemos la tabla *Cliente* que solo tiene dos campos a insertar: email y nombre.

Este código podría estar en un fichero que se llame *insertar-cliente.html*.

```
<form method="post" action="insertar-cliente.php">
    <input type="text" id="nombre" name="nombre">
    <input type="text" id="email" name="email">
</form>
```

Después, creamos el fichero *insertar-cliente.php*, el cual tiene que leer los datos del formulario anterior, por lo tanto tenemos que comenzar el fichero de la siguiente manera, comprobando que llegan los datos que esperamos y además los leemos en variables.

```
<?php

if ($_SERVER["REQUEST_METHOD"] === "POST"
    && isset($_POST["nombre"])
    && isset($_POST["email"]))
) {
    // Creamos las variables que nos permiten leer la información que
    // viene de $_POST
    $nombre = $_POST["nombre"];
    $email = $_POST["email"];

    // Aquí se inserta el resto del código que vamos a ir comentando

}
else {
    echo "No has enviado la información suficiente.";
}
```

Continuamos escribiendo código dentro de la condición *if* debajo de haber leído las variables que llegan del formulario HTML.

```
// !!Estamos dentro del if!!

// Incluimos el servicio de base de datos, que nos permite conectarnos.
include_once 'DatabaseService.php';

// Creamos una instancia del servicio de base de datos.
$databaseService = new DatabaseService();

// Nos conectamos a la base de datos.
$connection = $databaseService->getConnection();
```

Ahora realizamos la consulta SQL que este caso es un *insert*. Fíjate que:

- cuando creamos la consulta \$query en los valores del nombre y email estamos poniendo :nombre y :email y no directamente las variables de PHP. Esta sustitución es así para evitar el problema de SQL injection.
- Después ejecutamos la función *prepare()* utilizando la sentencia SQL anterior.
- Utilizamos la función *bindParam()* para ahora sí, sustituir las variables :nombre y :email por las variables que tenemos \$nombre y \$email de PHP.
- Finalmente, ejecutamos el método *execute()* que realiza la operacion SQL.

```
$query = "insert into clientes (nombre, email) values (:nombre, :email)";

$stmt = $connection->prepare($query);

$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':email', $email);

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

Finalmente comprobamos si ha habido un error y no y mostramos un mensaje.

```
if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error en la inserción";
}
else {
    /* No se ha producido un error */
    echo "Se ha ejecutado bien la inserción.";
```

3.3 Una modificación (update)

Cuando queremos modificar algo de la base de datos tenemos que hacerlo en cuatro pasitos:

1. Mostrar el contenido del cual seleccionamos algo para modificar añadiendo un algo que indique cómo modificar esa fila.
2. Mostrar un formulario con la información actual para que la persona usuaria lo modifique.
3. Realizar realmente la modificación.

Paso 1. Mostrar el contenido a modificar con un botón para modificar

Este paso de mostrar la información a modificar es igual que ya hicimos en el *delete*. Y es simplemente un *select*. Esto ya lo hemos explicado previamente. Solo hay una cosa que añadir que es un botón *Borrar*.

Por lo tanto seguimos todos los pasos que se correspondan con un *select* hasta llegar al punto donde tenemos este código:

```
$filas = $stmt->fetchAll();

foreach ($filas as $unaFila) {
    echo "{$unaFila['col1']}";
```

```

echo "{$unaFila['col2']}";
...
}

```

Y lo sustituimos por este código. A diferencia del eliminar, en este caso, estamos pasando toda la información de la fila mediante post y campos ocultos. Así, la página *form-update-cliente.php* donde la persona usuaria modificará esta fila, puede tener esta información antigua como referencia.

Este fichero podría llamar *lista-clientes-para-modificar.php*.

```

<?php

// Aquí solo hay una parte, tienes que hacer todo el select y añadir
// la parte que representa la diferencia con el select.

$filas = $stmt->fetchAll();

foreach ($filas as $unaFila) {
    $id = $unaFila["id"];
    $nombre = $unaFila["nombre"];
    $email = $unaFila["email"];
    $fechaRegistro = $unaFila["fechaRegistro"];
?>

    <form method="post"
        action="form-update-cliente.php"
        <input type="hidden" name="id" value="<?= $id ?>">

        <input type="hidden" name="nombre" value="<?= $nombre ?>">
        <input type="hidden" name="correo" value="<?= $email ?>">
        <input type="hidden" name="fechaRegistro"
               value="<?= $fechaRegistro ?>">

        <button type="submit">Modificar</button>
    </form>

<?php
    }
?>

```

Paso 2. Formulario para modificar

Siguiendo el ejemplo, el formulario para modificar debe estar en un fichero que se llama *form-update-cliente.php*.

Este fichero php se llama después de hacer clic en el botón *Modificar* de una fila concreta. Por lo tanto este fichero tiene que comprobar que se han enviado los datos por post de forma correcta.

```
<?php

if ($_SERVER["REQUEST_METHOD"] === "POST"
    && isset($_POST["id"])
    && isset($_POST["nombre"])
    && isset($_POST["email"])
    && isset($_POST["fechaRegistro"]))
) {
    // Creamos las variables que nos permiten leer la información que
    // viene de $_POST
    $id = $_POST["id"];
    $nombre = $_POST["nombre"];
    $email = $_POST["email"];
    $fechaRegistro = $_POST["fechaRegistro"];

    // Aquí se inserta el resto del código que vamos a ir comentando

}
else {
    echo "No has enviado la información suficiente.";
}
```

Y ahora, continuando dentro del *if* anterior tenemos que crear un formulario utilizando esos valores.

```
<form method="post" action="action-update-cliente.php">
    <input type="text" id="nombre" name="nombre" value="<?= $nombre?>">
    <input type="text" id="email" name="email" value="<?= $email?>">
    <input type="text" id="fechaRegistro" name="fechaRegistro" value="<?= $fechaRegistro?>">
</form>
```

Hemos de tener cuidado a la hora de hacer esto al mezclar código PHP y código HTML. El resultado final podría ser algo así, mezclando los dos últimos trozos de código.

Fíjate que dentro del código HTML se insertan variables PHP, las cuales, están entre los operadores **<?=** y **?>** que es otra forma de introducir código PHP en un fichero HTML cuando solo se quiere introducir el valor de una variable.

```
<?php

if ($_SERVER["REQUEST_METHOD"] === "POST"
    && isset($_POST["id"])
    && isset($_POST["nombre"])
    && isset($_POST["email"])
    && isset($_POST["fechaRegistro"]))
) {
    // Creamos las variables que nos permiten leer la información que
    // viene de $_POST
    $id = $_POST["id"];
    $nombre = $_POST["nombre"];
    $email = $_POST["email"];
    $fechaRegistro = $_POST["fechaRegistro"];
```

```
// Aquí se inserta el resto del código que vamos a ir comentando

?>

<form method="post" action="action-update-cliente.php">
    <input type="text" id="nombre" name="nombre" value="<?= $nombre?>">
    <input type="text" id="email" name="email" value="<?= $email?>">
    <input type="text" id="fechaRegistro" name="fechaRegistro" value="<?= $fechaRegistro?>">
    <input type="submit" value="Modificar">
</form>

<?php

}

else {
    echo "No has enviado la información suficiente.";
}
```

Paso 3. Creamos el fichero que realmente modifica:
action-update-cliente.php

Ahora creamos el fichero *action-update-cliente.php* que es el que realmente modifica. Este fichero es llamado desde un formulario, por lo tanto tenemos que hacer (como en los casos anteriores).

Primero, creamos el fichero *action-delete.php* con el contenido que comprueba si están realmente llegando los campos que queremos mediante *post*. En este caso, vamos a pasar únicamente la clave primaria de la tabla de la que queramos borrar una fila suponiendo que esa columna se llama *id*.

```
<?php

if ($_SERVER["REQUEST_METHOD"] === "POST"
    && isset($_POST["id"])
    && isset($_POST["nombre"])
    && isset($_POST["email"])
    && isset($_POST["fechaRegistro"]))
) {
    // Creamos las variables que nos permiten leer la información que
    // viene de $_POST
    $id = $_POST["id"];
    $nombre = $_POST["nombre"];
    $email = $_POST["email"];
    $fechaRegistro = $_POST["fechaRegistro"];

    // Aquí se inserta el resto del código que vamos a ir comentando

}
else {
    echo "No has enviado la información suficiente.";
}
```

Continuamos añadiendo dentro del *if* las líneas que nos permite conectarnos con la base de datos.

```
// !!Estamos dentro del if!!

// Incluimos el servicio de base de datos, que nos permite conectarnos.
include_once 'DatabaseService.php';

// Creamos una instancia del servicio de base de datos.
$databaseService = new DatabaseService();

// Nos conectamos a la base de datos.
$connection = $databaseService->getConnection();
```

Ahora realizamos la consulta SQL que este caso es un *delete*. Fíjate que:

- cuando creamos la consulta *\$query* en el valor de la columna *id* estamos poniendo *:id* y no directamente la variables PHP. Esta sustitución es así para evitar el problema de SQL injection.
- Después ejecutamos la función *prepare()* utilizando la sentencia SQL anterior.
- Utilizamos la función *bindParam()* para ahora sí, sustituir la variable *:id* por las variable que tenemos *\$id* de PHP.
- Finalmente, ejecutamos el método *execute()* que realiza la operacion SQL.

```
$query      = "update clientes set nombre=:nombre, email=:email,
fechaRegistro=:fechaRegistro where id=:id";

$stmt = $connection->prepare($query);

$stmt->bindParam(':id', $id);
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':email', $email);
$stmt->bindParam(':fechaRegistro', $fechaRegistro);

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

Finalmente comprobamos si ha habido un error y no y mostramos un mensaje.

```
if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error en la inserción";
}
else {
    /* No se ha producido un error */
    echo "Se ha ejecutado bien la inserción.";
}
```

4 Esquema general de cualquier aplicación

En cualquier fichero PHP donde necesitemos ejecutar una consulta SQL necesitamos tener el fichero *DatabaseService.php* en la misma carpeta.

En nuestros proyectos, que van a ser pequeños, es recomendable tener todos los ficheros en la misma carpeta con nombres bien elegidos para saber qué estamos haciendo.

Utilizar la clase *DatabaseService* respeta siempre la siguiente estructura:

1. Siempre ponemos las siguientes tres líneas

```
<?php

// Incluimos el servicio de base de datos, que nos permite conectarnos.
include_once 'DatabaseService.php';

// Creamos una instancia del servicio de base de datos.
$databaseService = new DatabaseService();

// Nos conectamos a la base de datos.
$connection = $databaseService->getConnection();
```

2. A continuación, tenemos que crear una variable que se llame `$query` con nuestra consulta SQL. Tenemos dos alternativas:
 - a. Que la sentencia SQL sea totalmente estática (por ejemplo `select * from cliente;`), es decir, no tiene valores que provengan de un formulario HTML o de cualquier otro cálculo.
En este caso continuamos por el paso 3.
 - b. Que la sentencia SQL sí tiene valores que provienen de fuera, normalmente variables que provienen de un formulario HTML o de cualquier otro cálculo
En este caso continuamos por el paso 5.
3. Como la sentencia SQL (la que sea) no tiene variables (no dependes de un formulario) podemos escribir tal cual tal sentencia en una variable `$query` y escribimos el siguiente código para continuar la ejecución.

En la variable `$stmt` tienes el resultado de ejecutar la sentencia. Es *false* si ha habido algún problema.

```
$query = "select * from una_tabla";
$stmt = $connection->query($query);
```

Este ejemplo con “insert” también sería válido (pero es raro).

```
$query = "insert into una_tabla(col1, col2) values ('1', 'hola')";
$stmt = $connection->query($query);
```

Continuamos por el paso 4.

4. Ahora tenemos que diferenciar entre dos casos:
 - a. Si hemos ejecutado un `select`. Continuamos por el paso 7.
 - b. No hemos ejecutado un `select`.Continuamos por el paso 8.

5. Como la sentencia SQL (la que sea) tiene variables, necesitamos introducirlos de forma segura en la consulta evitando problemas de SQL injection. Por tanto hacemos lo siguiente:
- Escribimos la consulta en la variable \$query, y donde tenga que haber una variable, ponemos el nombre de la variable comenzando por el símbolo dos puntos (:).
 - Ponemos una línea que ejecuta la función *prepare*.
 - Por cada variable que empiece por dos puntos, ponemos la función *bindParam* asociando esa variable que comenzaba por dos puntos (:) con la variable o cálculo de PHP que sea necesario.
 - Finalmente, tenemos la variable \$resultadoEjecucion que devuelve *false* si hubo un error en la ejecución de la función *execute()*. Aquí es realmente cuando se realiza la operación sobre la base de datos.

```
$query = "update clientes set nombre = :nombre, correo = :correo,
fecha_registro = :fechaRegistro where id = :id";

$stmt = $connection->prepare($query);

$stmt->bindParam(':id', $idCliente);
$stmt->bindParam(':nombre', $nombreCliente);
$stmt->bindParam(':correo', $correoCliente);
$stmt->bindParam(':fechaRegistro', $fechaRegistroCliente);

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

Continuamos en el paso 6.

- Ahora tenemos que diferenciar entre dos casos:
 - Si hemos ejecutado un *select*. Continuamos por el paso 9.
 - No hemos ejecutado un *select*.Continuamos por el paso 10.
- Estamos en el caso en que hemos ejecutado un *select* y solo nos queda extraer las filas del resultado e ir procesando fila a fila.

Por tanto, tenemos que ejecutar la función *fetchAll()* sobre el objeto \$stmt y obtenemos otro objeto llamado \$filas que al recorrerlo mediante la construcción *foreach* podemos acceder a cada una de las filas usando la variable \$unaFila.

Esta variable \$unaFila es un array asociativo, que permite mediante los corchetes y poniendo dentro una cadena de texto que representa el nombre de la columna acceder a esa columna. Fíjate que hay que poner entre llaves toda la construcción anterior para que funcione.

```
if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error";
}
else {
```

```
$filas = $stmt->fetchAll();

foreach ($filas as $unaFila) {
    echo "{$unaFila['col1']}";
    echo "{$unaFila['col2']}";
    ...
}
```

Y aquí terminamos.

8. Tenemos una consulta que no es un select y que hemos ejecutado sin variables. Por tanto, solo queremos saber si se ha ejecutado correctamente.

```
if (!$stmt) {
    /* Se ha producido un error */
    echo "Se ha producido un error";
}
else {
    /* No se ha producido un error */
    echo "Se ha ejecutado bien";
}
```

9. En este caso estamos ejecutando un select con variables en la consulta. Nos queda comprobar que la consulta se ha hecho correctamente y recorrer las filas del resultado.

Finalmente ejecutamos la función `execute()` de forma que el resultado de la ejecución quede en una variable `$resultado`. Esta variable será `false` si ha habido algún problema.

Si no ha habido ningún error podemos recorrer cada una de las filas de la consulta.

```
if (!$resultadoEjecucion) {
    /* Se ha producido un error */
    echo "Se ha producido un error";
}
else {
    $filas = $stmt->fetchAll();

    foreach ($filas as $unaFila) {
        echo "{$unaFila['col1']}";
        echo "{$unaFila['col2']}";
        ...
    }
}
```

10. En este caso no estamos ejecutando un select y tenemos variables en la consulta. Nos queda comprobar que la consulta se ha hecho correctamente y listo.

```
if (!$resultadoEjecucion) {
```

```

/* Se ha producido un error */
echo "Se ha producido un error";
}
else {
    /* No se ha producido un error */
    echo "Se ha ejecutado bien";
}

```

5 Diagrama de flujo para cualquier aplicación

Este diagrama de flujo representa lo mismo que se ha explicado en el punto anterior pero mediante un dibujo para que sea, tal vez, más claro.

