estudios abiertos
SEAS
GRUPO SANVALERO

5

UNIDAD DIDACTICA 5

PHP

5. PHP MySQL

ÍNDICE

OBJETIVOS

- Aprender a conectarnos a un servidor de bases de datos MySQL desde PHP.
- Conocer los distintos tipos de conexiones que podremos realizar a un servidor de bases de datos.
- Ejecutar consultas SQL desde un script PHP.
- Realizar las operaciones típicas de creación de tablas, y añadir, eliminar, modificar y visualizar registros.
- Consultar las principales recomendaciones para realizar un código bien escrito y estructurado.
- Poner en práctica todo lo aprendido con un ejemplo completo.

INTRODUCCIÓN

En capítulos anteriores hemos aprendido a trabajar con bases de datos MySQL, comprendiendo el concepto de tablas, registros y campos, y cómo realizar las operaciones frecuentes a través de la consola de comandos.

Por otra parte, hemos comenzado a descubrir los conceptos importantes relativos al lenguaje de programación PHP.

En esta unidad, el objetivo es aprovechar el conocimiento que ya poseemos tanto en PHP como en MySQL, para poder interactuar con bases de datos a través de este lenguaje de programación. Para ello, explicaremos y practicaremos con las principales instrucciones que nos posibilitarán estas operaciones.

No obstante, antes de comenzar con la explicación de estos comandos, vamos a repasar brevemente las operaciones que se suceden cuando los usuarios visitan nuestras páginas web, realizadas con programación en PHP y apoyándose en bases de datos MySQL:

1. En primer lugar, nuestro usuario realizará una petición HTTP cuando realice una solicitud a una de nuestras páginas.
2. El servidor web que tengamos configurado en nuestro servidor, ya sea Apache o cualquier otro, pasará esta solicitud al motor PHP para que la procese adecuadamente.
3. El motor de PHP comienza a realizar su trabajo y comienza a interpretar los comandos que incluya dicha página. Seguramente, se encontrará en algún momento con alguna instrucción que requiere una conexión a la base de datos para realizar algún tipo de consulta de datos o alguna operación de mantenimiento como hemos visto en unidades anteriores. En ese momento, el motor PHP realizará una conexión al servidor MySQL y le enviará la operación o consulta en concreto.
4. En este punto, el servidor MySQL recibirá dicha consulta y la ejecutará, devolviendo los resultados al motor de PHP.
5. El motor de PHP continuará ejecutando los comandos de su página PHP, convirtiendo todos los resultados a código HTML, que devolverá al servidor web.
6. Por último, el servidor web enviará dicho código HTML al navegador del usuario, que por fin obtendrá los resultados esperados acordes a su solicitud inicial.

En la siguiente imagen, podemos observar de forma gráfica el funcionamiento que acabamos de describir:

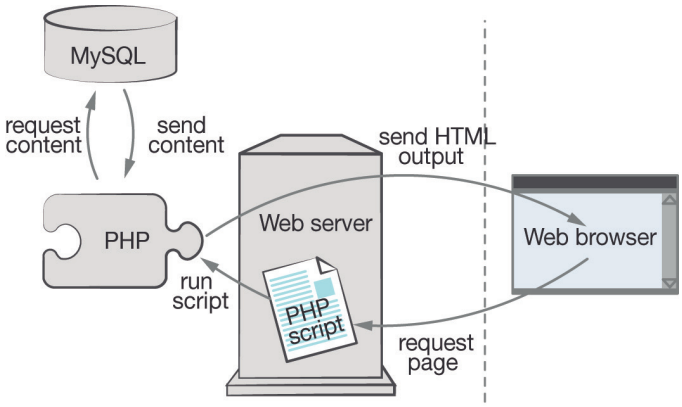

Figura 5.71. Arquitectura web con bases de datos.

Ésta será la forma de operar conjunta del trinomio Servidor Web-PHP-MySQL, y que es importante comprender antes de seguir avanzando.

A partir de este momento, vamos a conocer la forma en la que PHP nos permite trabajar con bases de datos, en este caso, bases de datos MySQL.

5.1. Conexión a MySQL

Antes de comenzar a realizar cualquier tipo de operaciones con nuestras bases de datos, en primer lugar deberemos de conectarnos al servidor MySQL, para lo cual, deberemos de usar los valores de conexión específicos que ya vimos en la Unidad 2. Serían los siguientes:

- Nombre de Host. En nuestro caso, si tenemos instalado un entorno de servidor web local, este valor sería "localhost", ya que tendremos todo instalado de forma local. Necesitaremos cambiar este valor si pretendemos conectarnos contra un servidor MySQL que se encuentre funcionando en un servidor de hosting en Internet.

- Usuario y contraseña. Cuando realizamos la instalación y configuración de nuestro servidor MySQL, por defecto se creó el usuario "root" sin contraseña. Estos valores deberíamos utilizar para realizar dicha conexión. No obstante, volvemos a repetir que esto nos puede servir para trabajar en modo local, pero cuando estemos trabajando en un entorno real, es obvio deducir que nos interesará tener configurados un usuario y contraseñas lo más complicados posibles para ofrecer un mínimo de seguridad.

En PHP, la función que utilizaremos para realizar la conexión con bases de datos MySQL será "mysqli_connect()". En el interior de los paréntesis, introduciremos los parámetros necesarios para realizar dicha conexión. Su sintaxis es la siguiente:

EJEMPLO

```
int mysqli_connect ([string hostname [:puerto][, string
usuario [, string password[, string database]]]])
```

La función devolverá un valor "1" si ha tenido éxito en la conexión, y un "0" si no ha logrado conectar con la base de datos.

Todos los argumentos de la función serán opcionales, y si no se introdujera ninguno, se asumirían los valores por defecto, que serían "localhost" para el servidor, el usuario propietario del proceso y la contraseña vacía.

En el argumento "hostname" podemos introducir también un puerto de conexión.

Si realizáramos una nueva llamada a la función "mysqli_connect()" con los mismos argumentos, no sería establecido un nuevo enlace, sino que nos devolvería el enlace que ya estuviera abierto.

El enlace al servidor será cerrado tan pronto como la ejecución del script finalice, a menos que lo cerremos antes explícitamente mediante la función mysqli_close().

Veamos un ejemplo de conexión a la base de datos:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db ='';
if ($conexion = mysqli_connect($host, $user, $pass,$db))
{
echo "Conexión realizada con éxito!";
echo "<br>";
mysqli_close($conexion);
}
else
{
echo "No se pudo realizar la conexión!";
echo "<br>";
}
?>
```

Como podemos observar en el anterior ejemplo, el enlace a la conexión de la base de datos es almacenado en la variable "$conexion", que utilizaremos posteriormente para realizar las operaciones con la base de datos.

Para cerrar la conexión con la base de datos mediante la función "mysqli_close()", utilizaremos siempre como argumento la variable donde hemos almacenado el enlace creado en la conexión satisfactoria.

En el ejemplo, controlamos que hemos conectado de forma satisfactoria al servidor de bases de datos mediante una estructura de tipo IF. Pero también podríamos haber realizado un control más corto y práctico de la siguiente forma:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db='';
$conexion = mysqli_connect($host, $user, $pass,$db)
        or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
mysqli_close($conexion);
?>
```

Este ejemplo de conexión suele ser muy utilizado, y lo habitual suele ser la creación de un fichero PHP donde se encuentre localizado este código de conexión al servidor MySQL, de tal forma que posteriormente podamos incorporarlo a cualquier script mediante la instrucción "include". Si, por ejemplo, hubiéramos almacenado este código en un fichero denominado conexion.php, podríamos incorporarlo a cualquier script de la siguiente forma:

EJEMPLO

```
<?php
include("conexion.php");
?>
```

5.2. Ejecución de consultas SQL

En la unidad 2 aprendimos a crear diferentes tipos de consultas en lenguaje SQL para realizar todo tipo de operaciones de consulta y/o mantenimiento de nuestras bases de datos.

PHP ofrece la posibilidad de ejecutar consultas SQL contra nuestra base de datos MySQL mediante la función **mysqli_query()**, que tiene la siguiente sintaxis:

DEFINICIÓN

```
resource mysqli_query ( int id_de_enlace, string $consulta)
```

Los argumentos utilizados en esta función son los siguientes:

- **$id_de_enlace.** Identificador de la conexión sobre la que el comando de SQL será enviado al servidor de base de datos.
- **$consulta.** Sentencia SQL que será enviada al servidor para su ejecución.

El argumento **$id_de_enlace** es un argumento obligatorio que indicará el enlace de conexión a la base de datos sobre la que queramos realizar la consulta.

Imaginemos una consulta donde queremos obtener los campos nombre, apellidos y email, de una tabla denominada DATOS, donde utilizaremos el identificador de conexión **$connect,** que previamente habremos definido para conectarnos a la base de datos. Para ejecutar esta consulta, tendríamos que utilizar la función **mysqli_query()** de la siguiente forma:

EJEMPLO

```
<?PHP
$consulta="SELECT NOMBRE, APELLIDO, EMAIL FROM DATOS";
$query=mysqli_query( $consulta,$connect);
>?
```

En este ejemplo, la consulta ha sido guardada en la variable **$consulta** previamente, y utilizada como argumento en la función **mysqli_query()** para ser ejecutada.

NOTA

El string de la consulta no debería terminar con un punto y coma. Los datos insertados en la consulta deberían estar correctamente escapados.

5.3. Creación de una base de datos desde PHP

Una vez que hayamos creado una conexión con nuestro servidor de bases de datos MySQL, podremos crear una nueva base de datos.

El método recomendado para crear una base de datos, sería a través de la ejecución de una consulta mediante la función **mysqli_query()** tal y como hemos explicado en el capítulo anterior.

Podríamos hacerlo de la siguiente forma:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db='';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";
$consulta="CREATE DATABASE IF NOT EXISTS nuevaDB";
if (mysqli_query($conexion,$consulta))
{
echo "Base de datos creada de forma satisfactoria!!";
}
else
{
echo "No pudo crearse la base de datos";
}
mysqli_close($conexion);
?>
```

En este ejemplo, hemos utilizado como parámetros en la función mysqli_query(), en primer lugar, el identificador de la conexión del servidor de la base de datos ($conexion) y, en segundo lugar la consulta SQL que queremos ejecutar contra el servidor.

5.4. Selección de la base de datos

En un servidor de bases de datos MySQL, podremos tener creadas varias bases de datos dispuestas a ser utilizadas.

Antes de empezar a trabajar con cualquiera de ellas, en primer lugar, tendremos que seleccionar la base de datos con la que queramos trabajar en cada momento.

En las últimas versiones de PHP, está selección de la base de datos la vamos a implementar por medio de la propia instrucción mysqli_connect(). El cuarto parámetro que recibe la instrucción hace referencia al nombre de la base de datos a la que nos queremos conectar.

Veamos un ejemplo:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db='test';
$conexion = mysqli_connect($host, $user, $pass,$db)
        or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
mysqli_close($conexion);
?>
```

En el ejemplo anterior se realizará una conexión a la base de datos test del servidor localhost.

5.5. Creación de una nueva tabla

Ahora que ya conocemos la forma de cómo ejecutar consultas mediante la función mysqli_query(), de la misma forma que la hemos utilizado para crear una base de datos, también podemos utilizarla para crear tablas en una base de datos en concreto.

Veamos un ejemplo de creación de tablas:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db='nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";
$consulta="CREATE TABLE IF NOT EXISTS ARTICULOS(
ID_ART INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
ARTICULO       VARCHAR(20)NOT NULL,
COD_FABRICANTE INT(3) NOT NULL,
PESO           INT(3) NOT NULL ,
CATEGORIA      VARCHAR(10) NOT NULL,
PRECIO_VENTA   DECIMAL(6,2),
PRECIO_COSTO   DECIMAL(6,2),
EXISTENCIAS    INT(5),
FECHA_COMPRA DATE )";
if (mysqli_query($conexion,$consulta))
{
echo "Tabla creada de forma satisfactoria!!";
}
else
{
echo "No pudo crearse la tabla";
}
mysqli_close($conexion);
?>
```

En el ejemplo, una vez superadas las fases de conexión a la base de datos donde se indica la base de datos sobre la que queremos ejecutar la consulta de creación de tablas, guardamos en la variable $consulta una consulta para crear una tabla, tal y como la explicamos en la unidad 2, cuando aprendimos a crear tablas mediante la consola de MySQL.

A continuación, ejecutamos la consulta controlando si hemos tenido éxito o no en dicha operación.

5.6. Inserción de un nuevo registro en una tabla

Mediante la utilización de la función **mysqli_query()** ya hemos aprendido a crear bases de datos y tablas. Ahora vamos a aprender a generar una consulta para poder insertar registros en dichas tablas.

La sintaxis básica SQL para la inserción de registros sería de la siguiente forma:

EJEMPLO

```
INSERT INTO NOMBRE_TABLA (CAMPO1, CAMPO2, CAMPO3, …) VALUES
('VALOR1','VALOR2','VALOR3',…)
```

En el anterior capítulo, hemos creado una nueva tabla denominada ARTICULOS, que sería la misma que hemos estado utilizando como ejemplo en la unidad 2. Para continuar con dicho ejemplo, vamos a insertar los mismos datos que teníamos en dicha tabla, de la siguiente forma:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db='nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="INSERT INTO ARTICULOS (ID_ART, ARTICULO, COD_
FABRICANTE, PESO, CATEGORIA, PRECIO_VENTA, PRECIO_COSTO,
EXISTENCIAS, FECHA_COMPRA)
VALUES
(1, 'MACARRONES', 100, 2, 'PRIMERA', 10.00, 5.00, 400, '2011-
03-04'),
(2, 'MACARRONES', 101, 3, '', 8.00, 4.00, 500, '2010-08-15'),
(3, 'JUDIAS', 100, 4, 'PRIMERA', 3.00, 2.00, 600, '2010-12-
12'),
(4, 'LENTEJAS', 102, 5, 'PRIMERA', 2.00, 1.00, 200, '2010-
11-11'),
(5, 'GARBANZOS', 103, 3, 'SEGUNDA', 6.00, 4.00, 300, '2011-
01-01')";
if (mysqli_query($conexion,$consulta))
{
echo "Datos insertados de forma satisfactoria!!";
}
else
{
echo "No pudo introducirse los datos";
}
mysqli_close($conexion);
?>
```

De esta forma, hemos guardado en la variable **$consulta**, la instrucción básica SQL para la inserción de varios registros en una sola vez, y posteriormente la ejecutamos, comprobando el éxito de la operación.

5.7. Eliminación de un registro de una tabla

Para eliminar un registro de una tabla, procedemos de la misma forma que en los anteriores capítulos, es decir, creando una consulta SQL y guardándola en una variable para su posterior inclusión en la función **mysqli_query()** como argumento principal.

No obstante, en la eliminación de registros, lo habitual suele ser que eliminemos una serie de registros en función de una o varias condiciones que tienen que ser cumplidas, para eliminar dichos registros.

Siguiendo con la tabla con la que estamos realizando estos ejemplos, vamos a suponer que queremos eliminar aquellos registros cuyo ARTICULO="MACARRONES".

En este ejemplo, lo único que varía en relación a anteriores capítulos es la instrucción SQL que vamos a ejecutar, en este caso DELETE, para eliminar los registros. Su sintaxis sería la siguiente:

DEFINICIÓN

```
DELETE FROM nombre_tabla WHERE condición;
```

Veamos cómo se realizaría en el siguiente ejemplo:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="DELETE FROM ARTICULOS WHERE ARTICULO='MACARRONES'";
if (mysqli_query($conexion,$consulta))
{
echo "Datos eliminados de forma satisfactoria!!";
}
else
{
echo "No pudieron eliminarse los datos";
}
mysqli_close($conexion);
?>
```

Hay que destacar que la consulta que hemos preparado para que posteriormente sea ejecutada, como siempre, la hemos encerrado con comillas dobles, pero para especificar el nombre del artículo que queremos eliminar, lo hemos encerrado con comillas simples.

5.8. Modificación de un registro de una tabla

Para realizar modificaciones o actualizaciones de registros en una tabla de la base de datos, también se procederá como en los anteriores ejemplos. De nuevo, lo único que varía es la instrucción SQL a utilizar, que en este caso sería la instrucción UPDATE. Su sintaxis sería la siguiente:

DEFINICIÓN

```
UPDATE nombre_tabla
SET lista_asignaciones
[ WHERE expresión_condicional ]
```

Las asignaciones se especifican del modo: nombre_columna = nuevo_valor.

De esta forma, la instrucción UPDATE actualizará las columnas de la tabla que especifiquemos en la cláusula SET, con los nuevos valores que introduzcamos a continuación.

Si utilizamos la cláusula opcional WHERE, solo serán actualizadas las columnas que cumplan con la expresión condicional.

En el ejemplo que escribimos a continuación, nos proponemos modificar los valores de todos los registros cuyos artículos estén catalogados como de PRIMERA categoría, modificando el valor del campo PRECIO_VENTA por un nuevo valor que será igual a la expresión matemática PRECIO_COSTO*1.6. Es decir, que deseamos obtener un beneficio del 60% sobre el costo inicial de cada artículo.

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="UPDATE ARTICULOS SET PRECIO_VENTA=PRECIO_COSTO*1.6
WHERE CATEGORIA='PRIMERA'";
if (mysqli_query($conexion,$consulta))
{
echo "Datos actualizados de forma satisfactoria!!";
}
else
{
echo "No pudieron actualizarse los datos";
}
mysqli_close($conexion);
?>
```

5.9. Visualización de registros de una tabla

Una vez hemos explicado la manera en la que podemos realizar operaciones de mantenimiento de una base de datos, vamos a explicar las diferentes soluciones que podemos aplicar en la visualización de resultados de consultas SQL a nuestras bases de datos, a través de las diferentes funciones que nos ofrece PHP.

Todas las funciones que vamos a explicar a continuación, utilizarán como argumento una variable que va a almacenar el resultado de una consulta que habremos ejecutado mediante la función **mysqli_query()**.

5.9.1. Función mysqli_fetch_row()

La función **mysqli_fetch_row()** recupera una fila de datos del resultado asociado al identificador de resultado especificado. La fila es devuelta como un array. Cada campo del resultado es almacenado en un índice del array, empezando desde 0. En último lugar devolverá FALSE si ya no quedan más filas que mostrar.

Su sintaxis es la siguiente:

DEFINICIÓN

```
array mysqli_fetch_row ( resource $result )
```

El argumento pasado a la función **$result**, va a contener el resultado de la función **mysqli_query()** que habremos ejecutado previamente.

Para observar cómo trabaja, vamos a realizar una consulta que nos devuelva todos los registros de una tabla, utilizando también instrucciones HTML para la creación de una tabla HTML para mostrar los datos de una forma más ordenada.

Como decimos, vamos a recorrer toda la tabla, pero por cada registro, vamos a simular que solo queremos mostrar sus 3 primeros campos: ID_ART, ARTICULO y COD_FABRICANTE. Para mostrar solo estos campos, tendremos que hacer referencia a las posiciones 0, 1 y 2 del array respectivamente.

Veamos como lo solucionaríamos:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT * FROM ARTICULOS";
$datos=mysqli_query($conexion,$consulta);
echo '<table border="1">';
while ($reg = mysqli_fetch_row($datos))
{
echo '<tr>';
echo '<td>',$reg[0],'</td>';
echo '<td>',$reg[1],'</td>';
echo '<td>',$reg[2],'</td>';
echo '</tr>';
}
echo '</table >';

mysqli_close($conexion);
?>
```

En este ejemplo, utilizamos un bucle **while**, que va a recorrer toda la tabla mientras el resultado de la función **mysqli_fetch_row($datos)** no devuelva FALSE, cosa que solo ocurrirá cuando llegue al final de la tabla.

Por cada uno de estos pasos por el bucle, guardará cada fila de la tabla en la variable **$reg**. En su interior, estarán todos los campos de este registro, los cuales podremos consultar utilizando un índice numérico (reg[0], reg[1] y reg[2]).

Supongamos que lo que queremos es directamente visualizar todos los campos de todos los registros. Para ello, podemos utilizar el bucle foreach() que ya explicamos en la unidad 3, y que nos ayudará a recorrer automáticamente todos los campos de la tabla.

Veamos cuál sería el código a utilizar:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT * FROM ARTICULOS";
$datos=mysqli_query($conexion,$consulta);
echo '<table border="1">';
while ($reg = mysqli_fetch_row($datos))
{
echo '<tr>';
foreach($reg as $cambia)
{
echo '<td>',$cambia,'</td>';
}
echo '</tr>';
}
echo '</table >';

mysqli_close($conexion);
?>
```

De esta forma, utilizamos el bucle **while** para explorar todos los registros resultantes de la consulta, y con el bucle **foreach** vamos a extraer todos los campos del registro que estamos evaluando en cada paso por el bucle **while**.

5.9.2. Función mysqli_fetch_array()

La función **mysqli_fetch_row()** tiene un inconveniente, y es que el array que devuelve, solo admite referencias numéricas a los campos obtenidos de la consulta. El primer campo referenciado es el 0, el segundo el 1 y así sucesivamente.

Mediante la función **mysqli_fetch_array()** podremos referenciar a los campos por su nombre, como si fuera un array asociativo, o si lo estimamos oportuno, con un índice numérico. Su sintaxis es la siguiente:

DEFINICIÓN

```
array mysqli_fetch_array ( resource $result [, int $result_
type = MYSQL_BOTH ] )
```

Mediante el argumento **$result_type** podemos especificar el tipo de array que va a ser devuelto. Es una constante y puede asignarle los siguientes valores:

- MYSQLI_ASSOC. Se obtienen solo los índices asociativos (tal como funciona mysql_fetch_assoc() que veremos posteriormente).
- MYSQLI_NUM. Se obtienen solo los índices numéricos (tal como funciona mysql_fetch_row()).
- MYSQLI_BOTH. Es la opción predeterminada, y se obtendrá un array con ambos índices: asociativos y numéricos.

En primer lugar, veamos cómo sería el código para utilizar índices numéricos. Como podremos comprobar, el código es prácticamente idéntico al utilizado con la función mysqli_fetch_row().

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT * FROM ARTICULOS";
$datos=mysqli_query($conexion,$consulta);
echo '<table border="1">';
while ($reg = mysqli_fetch_array($datos,MYSQLI_NUM))
{
echo '<tr>';
echo '<td>',$reg[0],'</td>';

echo '<td>',$reg[1],'</td>';
echo '<td>',$reg[2],'</td>';
echo '</tr>';
}
echo '</table >';

mysqli_close($conexion);
?>
```

A continuación, veamos cómo sería el código para utilizar índices asociativos.

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT * FROM ARTICULOS";
$datos=mysqli_query($conexion,$consulta);
echo '<table border="1">';
while ($reg = mysqli_fetch_array($datos,MYSQLI_ASSOC))
{
echo '<tr>';
echo '<td>',$reg['ID_ART'],'</td>';
echo '<td>',$reg['ARTICULO'],'</td>';
echo '<td>',$reg['COD_FABRICANTE'],'</td>';
echo '</tr>';
}
echo '</table >';

mysqli_close($conexion);
?>
```

Por último, veamos cómo sería la opción predeterminada, utilizando la constante MYSQLI_BOTH o directamente sin poner ningún parámetro.

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT * FROM ARTICULOS";
$datos=mysqli_query($conexion,$consulta);
echo '<table border="1">';
while ($reg = mysqli_fetch_array($datos,MYSQLI_BOTH))
{
echo '<tr>';
echo '<td>',$reg[0],'</td>';
echo '<td>',$reg['ARTICULO'],'</td>';
echo '<td>',$reg[2],'</td>';
echo '</tr>';
}
echo '</table >';

mysqli_close($conexion);
?>
```

Una cosa importante a tener en cuenta es que mysqli_fetch_array() no es significativamente más lenta que mysqli_fetch_row(), sin embargo provee un valor añadido importante.

NOTA

Los nombres de los campos devueltos por esta función son sensibles a mayúsculas y minúsculas.

5.9.3. Función mysqli_fetch_assoc()

Mediante la función **mysqli_fetch_assoc()** es equivalente a llamar ejecutar la función mysqli_fetch_array() utilizando como segundo parámetro la constante MYSQLI_ASSOC. Por lo tanto, devolverá un array asociativo que corresponderá con la fila recuperada, moviendo el apuntador de datos interno hacia delante.

Como en las anteriores funciones, cuando ya no tenga más filas disponibles devolverá el valor FALSE.

Su sintaxis es la siguiente:

DEFINICIÓN

```
array mysqli_fetch_assoc ( resource $result )
```

Por lo tanto, el ejemplo sería muy parecido al visto anteriormente:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT * FROM ARTICULOS";
$datos=mysqli_query($conexion,$consulta);
echo '<table border="1">';
while ($reg = mysqli_fetch_assoc($datos))
{
echo '<tr>';
echo '<td>',$reg['ID_ART'],'</td>';
echo '<td>',$reg['ARTICULO'],'</td>';
echo '<td>',$reg['COD_FABRICANTE'],'</td>';
echo '</tr>';
}
echo '</table >';

mysqli_close($conexion);
?>
```

5.9.4. Función mysqli_fetch_object()

La principal diferencia entre la función **mysqli_fetch_objetc()** y las vistas anteriormente, es que el resultado que devuelve, en lugar de ser un array, es un objeto con propiedades, que corresponderán a la fila recuperada, moviendo también el apuntador interno hacia delante.

Su sintaxis es la siguiente:

DEFINICIÓN

```
object mysqli_fetch_object ( resource $result [, string
$class_name [, array $params ]] )
```

Los parámetros utilizados son los siguientes:

- **$result**. Será la variable que contiene el resultado de haber realizado anteriormente la ejecución de la función **mysqli_query()**.
- **$class_name**. El nombre de la clase para instanciar y configurar sus propiedades.
- **$params**. Un array opcional de parámetros para pasar al constructor de los objetos **class_name**.

Veamos cómo sería el mismo ejemplo, pero utilizando objetos:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT * FROM ARTICULOS";
$datos=mysqli_query($conexion,$consulta);
echo '<table border="1">';
while ($reg = mysqli_fetch_object($datos))
{
echo '<tr>';
echo '<td>',$reg->ID_ART,'</td>';
echo '<td>',$reg->ARTICULO,'</td>';
echo '<td>',$reg->COD_FABRICANTE,'</td>';
echo '</tr>';
}
echo '</table >';

mysqli_close($conexion);
?>
```

5.9.5. Paginación de resultados

En numerosas ocasiones, los resultados que nos devuelva una consulta, seguramente será necesario mostrarlos en varios pantallazos, para una mejor lectura de ellos.

Para resolver este problema, se pueden implementar diferentes soluciones para paginar resultados, de tal forma que podamos decidir cuántos registros queremos mostrar por cada página.

La mejor forma de entender la solución propuesta, es adelantar el script que hemos preparado para el ejercicio final de esta unidad, donde realizaremos un pequeño proyecto web para realizar altas, bajas, modificaciones y listado de una sencilla tabla de una base de datos.

En este proyecto, se ofrece al usuario la posibilidad de poder paginar sus resultados.

Antes de mostrar el script, vamos a explicar el mismo para que pueda ser mejor comprendido.

Para comenzar, en este ejercicio utilizamos dos funciones que aún no han sido comentadas.

int mysqli_num_rows (resource $result)

Recupera el número de filas de una consulta. Este comando es únicamente válido para sentencias como SELECT o SHOW. El parámetro $result es el resultado de la ejecución previa de la función mysqli_query().

Esta función devolverá, el número de filas de una consulta en caso de éxito y FALSE en caso de error.

bool mysqli_data_seek (resource $result , int $row_number)

Esta función se encargará de mover el apuntador interno utilizado para trabajar con la fila actual. Esta función se utiliza junto con funciones como mysqli_fetch_row(), mysqli_fetch_array() o mysqli_fetch_assoc(), ya que devolverá la siguiente fila tras realizar una llamada a estas funciones.

Utiliza los siguientes parámetros:

- $result. Será el identificador que proviene de una llamada a mysqli_query().
- $row_number. Número de la fila deseada del resultado nuevo del apuntador. El primer valor que tiene será 0 y su rango debería ser un valor entre 0 a mysql_num_rows() -1. Sin embargo, si el conjunto de resultados está vacío (mysql_num_rows() == 0), una búsqueda a 0 fallará con un E_WARNING y mysql_data_seek() devolverá FALSE.

Veamos cómo podríamos utilizar esta función, por ejemplo, para realizar un recorrido inverso en los resultados de una consulta:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

$consulta="SELECT ARTICULO, CATEGORIA FROM ARTICULOS;";
$datos=mysqli_query($conexion,$consulta);
/* búsqueda de filas en orden inverso*/
for ($i = mysqli_num_rows($datos) - 1; $i >= 0; $i--) {
    if (!mysqli_data_seek($datos, $i)) {
        echo "No se encuentra la fila $i: " . mysql_error() . "\n";
        continue;
    }
    if (!($row = mysqli_fetch_assoc($datos))) {
        continue;
    }

    echo $row['ARTICULO'] . ' ' . $row['CATEGORIA'] . "<br />\n";
}

mysqli_close($conexion);
?>
```

A continuación utilizaremos el uso de una función mysqli_result definida por nosotros que reemplaza a la depreciada mysql_result:

```
function mysqli_result($res,$row=0,$col=0){

    $numrows = mysqli_num_rows($res);

    if ($numrows && $row <= ($numrows-1) && $row >=0){

        mysqli_data_seek($res,$row);

        $resrow = (is_numeric($col)) ? mysqli_fetch_row($res) : mysqli_fetch_
assoc($res);

        if (isset($resrow[$col])){
```

```
            return $resrow[$col];
        }
    }
    return false;
}
```

Está función recibe tres parámetros:

- $res. Identificador de la consulta devuelta por mysqli_query().
- $row. Número de fila a la que acceder para leer el dato
- $col. Campo de la fila que se quiere obtener

Esta función devuelve los resultados de una celda de una fila en caso de éxito, o FALSE en caso de fallo.

Las variables que utilizaremos para el ejercicio de paginación de resultados son las siguientes:

- $per_page. Variable en la que especificamos cuántos registros queremos mostrar por página.
- $result. Resultado de la consulta a la base de datos.
- $total_results. Número de registros que devuelve la consulta.
- $total_pages. Número de páginas totales que podrán ser mostradas.
- $start y $end. Intervalo de registros que serán mostrados en una página en concreto.

Una vez explicadas las nuevas funciones que van a ser utilizadas, y las variables que vamos a emplear en las operaciones referentes a la paginación de resultados, veamos cómo se implementará este script de paginación:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$conexion = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");
echo "Conexión realizada con éxito!";
echo "<br>";

//Función para ejecutar mysql_result de una manera compatible a mysqli
function mysqli_result($res,$row=0,$col=0){
    $numrows = mysqli_num_rows($res);
    if ($numrows && $row <= ($numrows-1) && $row >=0){
        mysqli_data_seek($res,$row);
         $resrow = (is_numeric($col)) ? mysqli_fetch_row($res) : mysqli_fetch_
assoc($res);
        if (isset($resrow[$col])){
            return $resrow[$col];
        }
    }
    return false;
}


// número de resultados para mostrar por página
$per_page = 3;
// calcular el total de páginas en la base de datos
$result = mysqli_query($conexion,"SELECT ARTICULO, CATEGORIA FROM
ARTICULOS;");
$total_results = mysqli_num_rows($result);
// la función ceil() redondea el resultado hacia arriba
$total_pages = ceil($total_results / $per_page);
// comprueba si la varible "page" está configurada en la URL
// EJEMPLO: view-paginated.php?page=1
if (isset($_GET['page']) && is_numeric($_GET['page']))
{
    $show_page = $_GET['page'];
```

EJEMPLO

```
    // Nos aseguramos que el valor de $show_page es válido
    if ($show_page > 0 && $show_page <=$total_pages)
    {
        $start = ($show_page -1) * $per_page;
        $end = $start + $per_page;
    }
    else
    {
    // error - muestra el primer conjunto de resultados
        $start = 0;
        $end = $per_page;
    }
}
else
{
    /* Si la página no está configurada, muestra el
    primer conjunto de resultados*/
    $start = 0;
    $end = $per_page;
}
// Muestra la Paginación en la parte superior de la pantalla
echo “<p><b>Ver P&aacute;gina:</b> “;
for ($i = 1; $i <= $total_pages; $i++)
{
    echo “<a href=’paginacion.php?page=$i’>$i</a> “;
}
echo “</p>”;
// Visualiza los datos en la tabla
echo “<table border=’1’ cellpadding=’10’>”;
echo “<tr> <th>ARTICULO</th> <th>CATEGORIA</th></tr>”;
// bucle a través de los resultados de la consulta a la base de datos,
// visualizándolos en la tabla
for ($i = $start; $i < $end; $i++)
{
     // nos aseguramos que PHP no intenta mostrar los resultados que no
existan
    if ($i == $total_results) { break; }
    // visualiza los contenidos en una tabla
```

EJEMPLO

```
for ($i = $start; $i < $end; $i++)
{
    // nos aseguramos que PHP no intenta mostrar los resultados que no existan
    if ($i == $total_results) { break; }
    // visualiza los contenidos en una tabla
    // para cada fila, extrae sus campos
    // y al final muestra los enlaces para editar o eliminar correspondientes al ID

    echo “<tr>”;
    echo ‘<td>’ . mysqli_result($result, $i, ‘ARTICULO’) . ‘</td>’;
    echo ‘<td>’ . mysqli_result($result, $i, ‘CATEGORIA’) . ‘</td>’;
    echo “</tr>”;

}
// cerramos la tabla>
echo “</table>”;
// paginación
?>
```

5.10. Otras funciones interesantes

En este capítulo vamos a comentar otras funciones que en algún momento también nos serán de utilidad.

int mysqli_errno ([resource $link_identifier])

Con esta función podremos recuperar el error en formato de texto de la última función de MySQL que hayamos ejecutado.

Utiliza los siguientes parámetros:

- **$link_identifier.** La conexión MySQL. Si el identificador de enlace no se especifica, el último enlace abierto por mysqli_connect() es asumido. Si no se encuentra dicho enlace, la función intentará establecer un nuevo enlace como si mysqli_connect() fuese invocado sin parámetros. Si no se encuentra o establece una conexión, un error de nivel E_WARNING es generado.

int mysqli_error ([resource $link_identifier])

Con esta función podremos recuperar el error en formato numérico de la última función de MySQL que hayamos ejecutado. Los errores que vienen del proceso final de la base de datos ya no emiten peligros. A su vez, utilizan mysqli_errno() para recuperar el código con error.

Utiliza los siguientes parámetros:

- **$link_identifier.** La conexión MySQL. Si el identificador de enlace no se especifica, el último enlace abierto por mysqli_connect() es asumido. Si no se encuentra dicho enlace, la función intentará establecer un nuevo enlace como si mysqli_connect() fuese invocado sin parámetros. Si no se encuentra o establece una conexión, un error de nivel E_WARNING es generado.

Veamos cómo podríamos utilizar estas funciones, por ejemplo, para comprobar el error que se produce al intentar conectarse a una base de datos que no existe:

EJEMPLO

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'nuevaDB';
$link = mysqli_connect($host, $user, $pass,$db)
or die ("No se pudo realizar la conexión!");

echo "Conexión realizada con éxito!";
echo "<br>";

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

if (!mysqli_query($link, "SET a=1")) {
    printf("Errorcode: %d\n", mysqli_errno($link));
}

/* close connection */
mysqli_close($link);
?>
```

Esta función debería de devolver un error como el siguiente:

EJEMPLO

```
Errorcode: 1193
```

5.11. Estructura de la aplicación

Antes de comenzar a escribir cualquier código de cualquier proyecto, siempre será necesario emplear el tiempo oportuno en la planificación de objetivos a conseguir con el proyecto y marcarse unos hitos en su evolución.

Es importante realizar la tarea de una forma ordenada, ya que si no nos podremos encontrar con problemas de aplicaciones defectuosas o simplemente que el código sea ilegible hasta para el propio creador, una vez pasado un tiempo.

A continuación, vamos a formular una serie de recomendaciones que pueden ser útiles para conseguir estos propósitos:

- Antes de comenzar con cualquier proyecto, detenerse a pensar qué es lo que se quiere crear, pensando en el objetivo final. Para ello, una ayuda es pensar en quiénes van a ser los usuarios finales del proyecto web, ya que no hay que olvidar que sobre todo, la aplicación tiene que ser útil para ellos principalmente.
- Dividir la aplicación en diferentes módulos o scripts y probarlos cada uno por separado y de forma conjunta. Tendremos que realizarnos preguntas como las siguientes:
  - ¿Cuáles son las partes en las que se divide mi aplicación?
  - ¿Qué tareas tiene que realizar cada una de ellas?
  - ¿Cómo pueden interactuar unas con otras?
- Una vez que tengamos claro cuáles son los diferentes módulos que van a componer nuestra aplicación, tendremos que revisar si alguno de ellos ya lo tenemos diseñado para otra aplicación que hayamos realizado anteriormente, o tenga una finalidad muy parecida. PHP, al ser de código abierto, también nos ofrece la posibilidad de que alguna solución ya esté aportada por alguna comunidad de desarrolladores. En definitiva, determinar en qué partes tendremos que crear el código desde cero, y en cuáles podremos aprovechar algo ya creado.
- Realizar optimizaciones del código de manera constante, probando la aplicación de todas las formas posibles.
- Especificar convenciones de nomenclatura. Para una mejor comprensión del código, siempre será oportuno realizar este tipo de aclaraciones, facilitando el otorgar un sentido rápido al objetivo de una variable o de una función. Siempre ayudará definir estas variables o funciones con unos nombres que sean fáciles de recordar.
- Utilizar el sangrado para dar una coherencia y sentido al código. El sangrado siempre ayudará a facilitar la lectura del código, con mayor rapidez si nos encontramos IF anidados, o bucles dentro de bucles, por ejemplo.
- Implementar un control de versiones. Si tenemos entre manos un proyecto lo suficientemente grande, puede ser interesante ayudarnos de un control de versiones. De esta forma, siempre podremos devolver un fragmento de código dividido a una versión que ya sabíamos que previamente funcionaba. Utilizando versiones, también establecemos la oportunidad de poder trabajar de forma conjunta con otros programadores.

5.12. Todas las operaciones en una aplicación. (Ejemplo completo)

Para poner en práctica todo lo aprendido en esta unidad, vamos a explicar y escribir el código completo de un pequeño proyecto web, a través del cual podremos realizar las operaciones básicas con una base de datos:

Crear la estructura de la base de datos que va a ser utilizada.

- Altas de registros.
- Bajas de registros.
- Modificaciones de registros.
- Listado de registros con y sin paginación de resultados.

Para ello, vamos a descomponer nuestra aplicación en diferentes módulos y scripts, donde cada uno de los cuales, cumplirá alguno de estos propósitos. Los archivos serían los siguientes:

- CONNECT-DB.PHP. Script donde vamos a definir las variables que utilizaremos para la conexión a la base de datos (servidor, usuario, password y base de datos), para posteriormente realizar la conexión al servidor de base de datos y seleccionar la base de datos que queremos utilizar.
- VIEW.PHP. Script principal de nuestra aplicación, donde se mostrará un listado total de registros con opción de paginación. Para cada uno de los registros se ofrecerá la posibilidad de editarlo o de eliminarlo. Además, también mostrará un enlace para añadir un nuevo registro.
- VIEW.PAGINATED.PHP. Es un script comentado ya anteriormente, donde se muestran los resultados paginados.
- EDIT.PHP. Script que mostrará un formulario para poder editar el registro que hayamos elegido. Para realizar la consulta SQL, utilizará el campo “ID” para indicar cuál es el registro que queremos editar.
- DELETE.PHP. Script que eliminará el registro elegido. Para realizar la consulta SQL, utilizará el campo “ID” para indicar cuál es el registro que queremos eliminar.
- NEW.PHP. Script que mostrará un formulario para añadir un nuevo registro. El único control que se realiza para validar los datos, es comprobar que no estén vacíos.
- TABLA.SQL. Script escrito en lenguaje SQL, y que podremos importar, por ejemplo, con PHPMYADMIN para crear la tabla que vamos a utilizar en esta aplicación, junto con unos primeros valores para que tenga algo de contenido para probar la aplicación.

A continuación, veamos el contenido de cada uno de estos módulos.

TABLA.SQL.

La tabla que vamos a utilizar en nuestra aplicación es muy sencilla. Simplemente va a tener los siguientes campos:

- ID.
- NOMBRE.
- APELLIDO.

EJEMPLO

```
CREATE TABLE players (
id int(11) NOT NULL auto_increment,
nombre varchar(32) NOT NULL,
apellido varchar(32) NOT NULL,
PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO players VALUES(1, 'German', 'Romeo');
INSERT INTO players VALUES(2, 'Pedro', 'Rodriguez');
INSERT INTO players VALUES(3, 'Raquel', 'Roberta');
INSERT INTO players VALUES(4, 'Samuel', 'Clavero');
```

CONNECT-DB.PHP

En este script podemos personalizar el servidor de bases de datos, el nombre de la base de datos, usuario y contraseña de conexión, así como los mensajes de control.

EJEMPLO

```
<?php
/*
 CONNECT-DB.PHP
Permite conectarnos a la base de datos
*/

 // Variables que usamos en la base de datos para la conexión
 $server = 'localhost';
 $user = 'root';
 $pass = '';
 $db = 'test';

 // Conexión a la base de datos
 $connection = mysqli_connect($server, $user, $pass,$db) or
die ("No pudo conectarse a la base de datos ... \n" .mysqli_
error ());
?>
```

VIEW.PHP

EJEMPLO

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Listado de Registros</title>
</head>
<body>
<?php
/*
VIEW.PHP
Muestra en pantalla todos los datos de la tabla "players"
*/
// conectamos a la base de datos
include('connect-db.php');
// obtenemos los resultados de la base de datos mediante la
consulta
$result = mysqli_query($connection,"SELECT * FROM players")
or die(mysqli_error());
/* Visualizamos los datos en una tabla
Primero mostramos los links necesarios para ver sin paginar o
paginados. El parámetro ?page, nos indicará al tener valor 1
que es
primera página de resultados posibles.
*/
echo "<p><b>Ver todos</b> | <a href='viewpaginated.
php?page=1'>Ver paginados</a></p>";
echo "<table border='1' cellpadding='10'>";
echo "<tr> <th>ID</th> <th>Nombre</th> <th>Apellido</th>
<th></th><th></th></tr>";
//bucle para extraer los resultados de la base de datos
//los visualiza en el interior de la tabla
while($row = mysqli_fetch_array( $result ))
{
// salida de contenidos de cada columna en una fila de la tabla
echo "<tr>";
echo '<td>' . $row['id'] . '</td>';
echo '<td>' . $row['nombre'] . '</td>';
echo '<td>' . $row['apellido'] . '</td>';
echo '<td><a href="edit.php?id=' . $row['id'] . '">Editar</
a></td>';
echo '<td><a href="delete.php?id=' . $row['id'] . '">Eliminar</
a></td>';
echo "</tr>";
}
// terminamos la tabla
echo "</table>";
/* En la parte final de la página, mostramos un link para
añadir un
nuevo registro*/
?>
<p><a href="new.php">A&ntilde;adir un nuevo registro</a></p>
</body>
</html>
```

VIEW-PAGINATED.PHP

Script ya comentado previamente, donde podremos ofrecer los resultados en varias páginas. Mediante la variable $per_page, podremos especificar cuántos registros queremos por pantalla.

EJEMPLO

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Listado de Registros</title>
</head>
<body>
<?php
/*
VIEW-PAGINATED.PHP
Visualiza todos los datos de la tabla "player"
Es una versión modificada del fichero view.php, que
incluye paginación
*/

//Función para ejecutar mysql_result de una manera
compatible a mysql
function mysqli_result($res,$row=0,$col=0){
    $numrows = mysqli_num_rows($res);
    if ($numrows && $row <= ($numrows-1) && $row >=0){
        mysqli_data_seek($res,$row);
         $resrow = (is_numeric($col)) ? mysqli_fetch_
row($res) : mysqli_fetch_assoc($res);
        if (isset($resrow[$col])){
            return $resrow[$col];
        }
    }
    return false;
}


// conectamos a la base de datos
include('connect-db.php');
// número de resultados para mostrar por página
$per_page = 3;
```

EJEMPLO

```
// calcular el total de páginas en la base de datos
$result = mysqli_query($connection,"SELECT * FROM
players");
$total_results = mysqli_num_rows($result);
// la función ceil() redondea el resultado hacia arriba
$total_pages = ceil($total_results / $per_page);
// comprueba si la varible "page" está configurada en
la URL
// EJEMPLO: view-paginated.php?page=1
if (isset($_GET['page']) && is_numeric($_GET['page']))
{
    $show_page = $_GET['page'];

     // Nos aseguramos que el valor de $show_page es
válido
    if ($show_page > 0 && $show_page <=$total_pages)
    {
        $start = ($show_page -1) * $per_page;
        $end = $start + $per_page;
    }
    else
    {
    // error - muestra el primer conjunto de resultados
        $start = 0;
        $end = $per_page;
    }
}
else
{
    /* Si la página no está configurada, muestra el
    primer conjunto de resultados*/
    $start = 0;
    $end = $per_page;
}
// Muestra la Paginación en la parte superior de la
pantalla
echo "<p><a href='view.php'>Ver todas</a> | <b>Ver
P&aacute;gina:</b> ";
for ($i = 1; $i <= $total_pages; $i++)
{
   echo "<a href='viewpaginated.php?page=$i'>$i</a> ";
}
```

EJEMPLO

```
// calcular el total de páginas en la base de datos
$result = mysqli_query($connection,"SELECT * FROM
players");
$total_results = mysqli_num_rows($result);
// la función ceil() redondea el resultado hacia arriba
$total_pages = ceil($total_results / $per_page);
// comprueba si la varible "page" está configurada en
la URL
// EJEMPLO: view-paginated.php?page=1
if (isset($_GET['page']) && is_numeric($_GET['page']))
{
    $show_page = $_GET['page'];

     // Nos aseguramos que el valor de $show_page es
válido
    if ($show_page > 0 && $show_page <=$total_pages)
    {
        $start = ($show_page -1) * $per_page;
        $end = $start + $per_page;
    }
    else
    {
    // error - muestra el primer conjunto de resultados
        $start = 0;
        $end = $per_page;
    }
}
else
{
    /* Si la página no está configurada, muestra el
    primer conjunto de resultados*/
    $start = 0;
    $end = $per_page;
}
// Muestra la Paginación en la parte superior de la
pantalla
echo "<p><a href='view.php'>Ver todas</a> | <b>Ver
P&aacute;gina:</b> ";
for ($i = 1; $i <= $total_pages; $i++)
{
   echo "<a href='viewpaginated.php?page=$i'>$i</a> ";
}
```

EDIT.PHP.

EJEMPLO

```
<?php
/*
EDIT.PHP
Permite a los usuarios editar una entrada específica de
la base
de datos
*/
/* Crea el formulario para editar el registro desde
este
formulario, que es usado en múltiples ocasiones en este
fichero,
se convierte en una función que podemos volver a usar
fácilmente
*/
function renderForm($id, $nombre, $apellido, $error)
{
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Editar Registro</title>
</head>
<body>
<?php
// si hay errores, nos los muestra en pantalla
if ($error != '')
{
echo '<div style="padding:4px; border:1px solid red;
color:red;">'.$error.'</div>';
}
?>
<form action="" method="post">
<input type="hidden" name="id" value="<?php echo $id;
?>"/>
<div>
<p><strong>ID:</strong> <?php echo $id; ?></p>
<strong>Nombre:    *</strong>    <input    type="text"
name="nombre" value="<?php echo $nombre; ?>"/><br/>
<strong>Apellido:    *</strong>    <input    type="text"
name="apellido" value="<?php echo $apellido; ?>"/><br/>
```

EJEMPLO

```
<p>* Requerido</p>
<input type=”submit” name=”submit” value=”Submit”>
</div>
</form>
</body>
</html>
<?php
}
// conectamos a la base de datos
 include(‘connect-db.php’);
//comprueba si el formulario ha sido enviado. Si es correcto, procesa el formulario y guarda los datos en la BD.
if (isset($_POST[‘submit’]))
{
//confirma que el “ID” recibido es un valor válida antes de obtener los datos del formulario
if (is_numeric($_POST[‘id’]))
{
// obtiene los datos del formulario, asegurándonos que son válidos
$id = $_POST[‘id’];
$nombre=mysqli_real_escape_string($connection,htmlspecialchars($_POST[‘nombre’]));
$apellido=mysqli_real_escape_string($connection,htmlspecialchars($_POST[‘apellido’]));
// comprobamos que el nombre y apellidos tienen algún valor
/*}
else
{*/
// guardamos los datos en la base de datos
mysqli_query($connection,”UPDATE players SET nombre =’$nombre’,apellido=’$apellido’ WHERE id=’$id’”) or die(mysqli_error());
//una vez guardados, redirigimos a la página principal
header(“Location: view.php”);

//}
}
else
{
```

EJEMPLO

```
/* Si el formulario no ha sido enviado, obtenemos los
datos del
formulario desde la base de datos y visualizamos el
formulario*/
{
/* Obtenemos el "id" desde la URL (si existe),
asegurándonos
que es válido comprobando que sea numérico o mayor que
0*/
if (isset($_GET['id']) && is_numeric($_GET['id']) &&
$_GET['id']> 0)
{
// consulta a la base de datos
$id = $_GET['id'];
$result = mysqli_query($connection,"SELECT * FROM
players WHERE id=$id")or die(mysqli_error());
$row = mysqli_fetch_array($result);
// comprueba que el "id" coincide con una fila en la
base de datos
if($row)
{
// obtenemos datos desde la bd
$nombre = $row['nombre'];
$apellido = $row['apellido'];
// mostramos el formulario
renderForm($id, $nombre, $apellido, '');
}
else
// si no hay resultados, lo indicamos

   {
echo "No hay resultados!";
}
}
else
/* Si el "ID" de la URL no es válido, o si no hay un
valor de "id",
mostramos un error*/
{
echo 'Error!';
}
}
?>
```

NEW.PHP.

EJEMPLO

```
<?php
/*
 NEW.PHP
 Permite a los usuarios crear una nueva entrada en la
base de datos
*/

 // crear el nuevo formulario de nuevo registro

 function renderForm($nombre, $apellido, $error)
{
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Nuevo Registro</title>
</head>
<body>
<?php
// Si hay errores, los muestra en pantalla
if ($error != '')
{
echo '<div style="padding:4px; border:1px solid
red;color:red;">'.$error.'</div>';
}
?>
<form action="" method="post">
<div>
<strong>Nombre: *</strong> <input
type="text"name="nombre" value="<?php echo $nombre;
?>"
/><br/>
<strong>Apellido: *</strong> <input
type="text"name="apellido" value="<?php echo $apellido;
?>"
/><br/>
<p>* Requerido</p>
<input type="submit" name="submit" value="Submit">
</div>
</form>
</body>
```

EJEMPLO

```
</html>
<?php
}
// conectar a la base de datos
 include('connect-db.php');
// Comprueba si el formulario ha sido enviado.
// Si se ha enviado, comienza el proceso el formulario
y guarda los datos en la DB
if (isset($_POST['submit']))
{
// Obtenemos los datos del formulario, asegurándonos
que son válidos.
$nombre=mysqli_real_escape_string($connection,htmlspe
cialchars($_POST['nombre']));
$apellido =mysqli_real_escape_string($connection,htmls
pecialchars($_POST['apellido']));
// Comprueba que ambos campos han sido introducidos
if ($nombre == '' || $apellido == '')
{
// Genera el mensaje de error
$error = 'ERROR: Por favor, introduce todos los campos
requeridos.!';
// Si ningún campo está en blanco, muestra el formulario
otra vez
renderForm($nombre, $apellido, $error);
}
else
{
// guardamos los datos en la base de datos
mysqli_query($connection,"INSERT players SET
nombre='$nombre', apellido='$apellido'") or die(mysqli_
error());
/* Una vez que han sido guardados, redirigimos a la
página de vista principal*/
header("Location: view.php");
}
}
else
// Si el formulario no han sido enviado, muestra el
formulario
{
renderForm('','','');
}
?>
```

DELETE.PHP

EJEMPLO

```
<?php
/*
DELETE.PHP
Elimina una entrada específica de la tabla "players"
*/
// conectar a la base de datos
include('connect-db.php');
/*comprobamos si la variable "id" está configurada en la
URL, y comprobamos que es válida */
if (isset($_GET['id']) && is_numeric($_GET['id']))
{
// obtener el valor de ID mediante el método GET
$id = $_GET['id'];
// eliminamos la entrada
$result = mysqli_query($connection,"DELETE FROM players
WHERE id=$id") or die(mysqli_error());
// redirigimos de vuelta a la página de vista principal
header("Location: view.php");
}
else
/* Si el ID no está configurado, o no es válido, volvemos
a la página principal*/
{
header("Location: view.php");
}
?>
```

RESUMEN

- Hemos aprendido funciones que nos permiten interactuar directamente con bases de datos MySQL.
- Estas funciones nos han permitido incluir en nuestras aplicaciones el código necesario para poder conectarnos a un servidor de bases de datos MySQL, mediante conexiones persistentes o no, para posteriormente determinar cuál es la base de datos con la que queremos trabajar.
- Hemos puesto en práctica también otras funciones para realizar las operaciones típicas de mantenimiento y visualización de registros de una base de datos.
- Se ha explicado un ejemplo amplio para poner en práctica todo lo aprendido anteriormente.