

ÍNDICE**UD 5. PHP Y LAS BASES DE DATOS**

5.1 Estructura de una aplicación web con bases de datos.....	2
5.2 ¿Qué es MySQL?.....	2
5.2.1 Bases de datos relacionales.....	2
5.2.2 MySQL.....	2
5.3 El lenguaje SQL.....	3
5.3.1 ¿Qué es SQL?.....	3
5.3.2 Trabajar con MySQL.....	3
5.3.3 Sintaxis de SQL.	3
5.3.4 Crear la estructura de la base de datos.....	4
5.3.5 Operaciones con datos.....	7
5.4 Funciones de PHP con MySQL.....	12
5.4.1 Conectar con la base de datos.....	12
5.4.2 Leer datos.....	13
5.4.3 Añadir datos.....	15
5.4.4 Modificar datos.	15
5.4.5 Eliminar datos.....	16

5. PHP Y LAS BASES DE DATOS

5.1 Estructura de una aplicación web con bases de datos.

Hasta el momento hemos estudiado la sintaxis y funcionamiento de PHP, abarcando el envío, recepción, y procesamiento de datos entre scripts. **Sin embargo, aún nos queda pendiente una parte fundamental de la mayoría de aplicaciones que encontraremos: el almacenamiento de datos.** Si bien hemos aprendido como acceder a un fichero de texto para leer y guardar información, este mecanismo afectaría gravemente al rendimiento de nuestra aplicación cuando tengamos que tratar con grandes cantidades de información. Con este propósito, disponemos de software especializado llamado *Sistema de Gestión de Bases de Datos*, **SGBD**, o sencillamente bases de datos. El uso de las bases de datos resulta fundamental para la mayoría de aplicaciones de un tamaño considerable. Entre otras aplicaciones podríamos destacar las siguientes: tienda virtual, catálogo en Internet, oficina/secretaría virtual, gestor de contenidos. Al añadir bases de datos a una aplicación web se amplía el esquema que interviene en la generación de una página web con un nuevo elemento.

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

- Capa primera: cliente web (navegador).
- Capa segunda: servidor web con intérprete de PHP.
- Capa tercera: sistema gestor de bases de datos.

El sistema gestor de bases de datos puede estar instalado en el mismo equipo que el servidor web o en uno diferente. En nuestro entorno, asumiendo que todos usamos Xampp, tenemos el servidor Apache y la base de datos MySQL (o MariaDB en las últimas versiones. A partir de ahora, todo lo que hablemos de Mysql se aplica a MaríaDB también) ubicados en el mismo ordenador. PHP es el encargado de la comunicación entre la segunda y la tercera capa. Esto puede hacerlo de dos formas:

- Mediante el API específico de la base de datos.
- Mediante ODBC, el estándar de comunicación con bases de datos.

Nosotros usaremos la primera forma, ya que es más sencilla y eficiente.

5.2 ¿Qué es MySQL?

5.2.1 Bases de datos relacionales.

Aunque existen diversos tipos de bases de datos, el más generalizado en la actualidad es la base de datos relacional.

Éste es el modelo más adecuado para modelar problemas reales, y resulta sencillo de usar y administrar. El usuario se puede desentender de donde y como se almacenan los datos. La información puede ser recuperada o almacenada mediante consultas que ofrecen una amplia flexibilidad y poder para administrar la información. Las bases de

datos relacionales guardan la información en tablas formadas por registros, que corresponden a las filas de la tabla y que, a su vez, están formados por campos, que se corresponderían a las columnas de la tabla.

5.2.2 MySQL.

MySQL es el sistema gestor de bases de datos más utilizado en las aplicaciones web de Internet, desarrollado por la compañía MySQL AB. En sus comienzos carecía de muchas funcionalidades habituales en este tipo de bases de datos como integridad referencial o soporte para transacciones, pero precisamente por estas carencias resultaba una base de datos muy rápida y eficiente, lo que la hizo destacar en popular para el desarrollo web. La versión actual ha evolucionado notablemente y es mucho más completa.

MySQL es una base de datos relacional y destaca por las siguientes características.

- Es sencilla de usar y administrar.
- Es segura.
- Es eficiente.
- Es software libre, y por lo tanto gratuito (hoy en día tenemos otros como MariaDB, ya que MySql puede no ser libre dependiendo el uso que se le de)
- Soporta el lenguaje de consultas SQL.
- Está disponible para casi todos los sistemas operativos.
- Ofrece posibilidades avanzadas de administración
- Su sencillez no es un inconveniente, la versión actual incluye muchas de las características avanzadas que ofrecen otras bases de datos de pago.

MySQL es la pareja perfecta para las aplicaciones programadas en PHP.

5.3 El lenguaje SQL.

5.3.1 ¿Qué es SQL?

SQL es el acrónimo del término inglés Structured Query Language o lenguaje de consultas estructurado. Es un lenguaje que se usa para comunicarnos con el sistema gestor de bases de datos. Es a la vez potente y sencillo a la hora de recuperar información de forma estructurada.

5.3.2 Trabajar con MySQL.

El lenguaje de consultas SQL se va a explicar (como recordatorio, ya que la mayoría lo estudiasteis en el módulo de primero de Bases de Datos) a través de ejemplos sobre MySQL. Uno de los inconvenientes que tiene MySQL es que, el producto básico no dispone de un entorno gráfico amigable para administrar la base de datos, por lo que se ha de hacer a través de una consola en modo texto. No obstante, con un poco de práctica encontraremos que la consola es un medio rápido para conseguir nuestros objetivos. Para poder acceder a la consola de MySQL es necesario seguir los siguientes pasos, suponiendo que hayamos instalado el paquete Xampp en Windows:

1. Xampp debe estar iniciado. De no ser así iremos a la carpeta C:\xampp (o el

directorio donde hemos instalado Xampp) y ejecutaremos “xampp_start.exe”.

2. Una vez arrancado Xampp ejecutaremos el comando “mysql -u root”. Si hemos instalado Xampp en la ruta indicada arriba, tendremos que hacerlo desde la ruta C:\xampp\mysql\bin abriendo un símbolo del sistema. Esto se puede hacer en Windows desde “Inicio -- > Ejecutar” o desde MS-DOS (símbolo de sistema).

La consola es igual tanto en Windows como en Linux, por lo que una vez dentro no se apreciarán diferencias.

5.3.3 Sintaxis de SQL.

A diferencia de PHP, MySQL no distingue entre mayúsculas y minúsculas y las instrucciones se pueden escribir tanto en una como en la otra forma. Sin embargo, para hacerlo más legible, es habitual escribir las palabras clave del lenguaje en mayúsculas y el resto del código en minúsculas. Esta convención se seguirá a lo largo de todo el texto. Otra característica de MySQL es que las sentencias se terminan siempre en punto y coma, como en PHP.

5.3.4 Crear la estructura de la base de datos.

Antes de poder almacenar datos hemos de crear una estructura que los pueda contener.

5.3.4.1 Creación de una base de datos.

MySQL es un sistema gestor de bases de datos capaz de gestionar más de una base de datos independiente. Para trabajar con una base de datos, el primer paso siempre será crearla. Para ello, emplearemos la sentencia CREATE DATABASE de SQL antes de continuar con nuestras pruebas.

```
CREATE DATABASE nombre_base_datos;
```

Ejemplo 5.3.1. Creamos nuestra base de datos que llamaremos “curso_php”.

5.3.4.2 Selección de una base de datos.

Antes de poder trabajar con la base de datos hemos de seleccionarla. Para ello usaremos el comando USE.

```
USE nombre_base_datos;
```

Una vez ejecutado el comando USE, cualquier operación que hagamos con MySQL se aplicará a la base de datos seleccionada para su uso.

Ejemplo 5.3.2. Procedemos a usar la base de datos “curso_php”.

5.3.4.3 Creación de tablas.

Las bases de datos almacenan la información en tablas, que son el pilar fundamental de las bases de datos relacionales. La instrucción SQL que se utiliza para crear una tabla es CREATE TABLE.

```
CREATE TABLE nombre_tabla (definicion_tabla);
```

La definición de la tabla está compuesta por una lista de sus campos (columnas).

```
CREATE TABLE nombre_tabla (  
nombre_campo1 tipo_campo1 atributos_campo1,  
nombre_campo2 tipo_campo2 atributos_campo2,  
...  
);
```

La definición de cada campo, salvo el último, debe terminar en coma. Cada campo sólo puede almacenar un tipo de datos. De momento nos bastará con conocer los siguientes:

- INT: almacena números enteros.
- CHAR(longitud): almacena cadenas de texto de un máximo de longitud caracteres.
- DATE: almacena fechas.
- DATETIME: almacena fechas con la hora.

Los atributos de un campo estarán vacíos en la mayoría de los casos. Aunque hay una excepción: en casi todas las tablas se suele poner un campo id que contenga un identificador único de cada registro. A este campo se le dan los atributos:

- PRIMARY KEY: indica que el campo identifica de forma única el registro (fila).
- AUTO_INCREMENT: el campo se rellena automáticamente incrementándose su valor en 1 cada vez que se añade un registro. Aunque no siempre se debe optar por usar un identificador autoincremental, en muchos casos suele ser recomendable su uso.

Ejemplo 5.3.3. En este ejemplo se ve la creación de una tabla con datos sobre los empleados de una empresa.

```
CREATE TABLE empleados (  
id INT PRIMARY KEY AUTO_INCREMENT,  
nombre CHAR(50),  
puesto CHAR(20),  
fecha_nacimiento DATE,  
salario INT  
);
```

Los campos que se han definido en la tabla tienen las siguientes características:

- id: campo de tipo entero que identifica de forma única un registro. Cada vez que se añade un registro se incrementa automáticamente.
- nombre: campo que puede almacenar una cadena de hasta 50 caracteres, destinado a guardar el nombre. No tiene atributos.

- puesto: campo, que puede almacenar una cadena de hasta 20 caracteres, destinado a guardar el puesto que desempeña. No tiene atributos.
- fecha_nacimiento: almacena la fecha de nacimiento en formato YYYY-MM-DD (4 cifras para el año, guión, 2 cifras para el mes, guión, 2 cifras para el día). No tiene atributos.
- salario: campo de tipo entero, que guarda el sueldo en euros del empleado. No tiene atributos.

Ejecutando sentencia SQL, que también podríamos hacerlo en la consola de MySQL, obtendremos el siguiente resultado.

```
mysql> CREATE TABLE empleados (  
-> id INT PRIMARY KEY AUTO_INCREMENT,  
-> nombre CHAR(50),  
-> puesto CHAR(50),  
-> fecha_nacimiento DATE,  
-> salario INT  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

5.3.4.4 Listar tablas.

Para ver todas las tablas que hay disponibles en una base de datos se usa la instrucción SHOW TABLES.

```
SHOW TABLES;
```

Ejemplo 5.3.4. Al listar las tablas de la base de datos comprobamos que solo tenemos la tabla que

```
mysql> SHOW TABLES;  
+-----+  
| Tables_in_curso_php |  
+-----+  
| empleados           |  
+-----+  
1 row in set (0.03 sec)
```

acabamos de crear.

5.3.4.5 Mostrar la definición de una tabla.

Esto se hace mediante la sentencia DESCRIBE de SQL.

```
DESCRIBE nombre_tabla;
```

Ejemplo 5.3.5. Al mostrar la descripción de la tabla empleados se puede comprobar que coincide con la definición que usamos al crearla.

```
mysql> DESCRIBE empleados;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)   | NO   | PRI | NULL    | auto_increment |
| nombre     | char(50)  | YES  |     | NULL    |              |
| puesto     | char(50)  | YES  |     | NULL    |              |
| fecha_nacimiento | date      | YES  |     | NULL    |              |
| salario    | int(11)   | YES  |     | NULL    |              |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.02 sec)
```

5.3.4.6 Borrar tablas.

Para borrar una tabla se usa la sentencia DROP DATABASE de SQL. Hay que tener mucho cuidado ya que al borrar una tabla también eliminaremos de forma irrecuperable todos los datos que hubiéramos introducido.

```
DROP TABLE nombre_tabla;
```

Ejemplo 5.3.6. Vamos a eliminar la tabla empleados. Todos los datos que contenga se borrarán.

```
mysql> DROP TABLE empleados;
Query OK, 0 rows affected (0.00 sec)
```

Sin embargo, tal y como nos muestra el sistema, no se ha visto afectada ninguna tupla (o fila) de la tabla, por lo que podríamos deducir que estaba vacía en el momento de su eliminación.

5.3.4.7 Creación de tablas mediante un script.

La definición de una base de datos es costosa de escribir y de introducir manualmente, como hemos visto en las anteriores consultas. Además, un pequeño error nos obligará a volver a introducir varias consultas. Para solventar este inconveniente, y dado que SQL es un lenguaje textual, el método de creación de tablas más eficiente consiste en introducir las instrucciones de creación en un fichero de texto de extensión “.sql” (para distinguirlo de otros ficheros), e indicarle a MySQL que ejecute las instrucciones contenidas en dicho fichero. De esta manera, si se presenta algún error podremos editar su contenido fácilmente con cualquier editor de texto plano, y volver a ejecutar su contenido.

Este fichero deberá contener la selección de la base de datos y tantas consultas de creación (e inicialización mediante inserción de datos, para así tener contenido inicial) como se desee.

Ejemplo 5.3.7. Nuestra base de datos “curso_php” quedará definida así en el fichero “curso_php.sql”.

```
CREATE DATABASE IF NOT EXISTS curso_php;
USE curso_php;
CREATE TABLE IF NOT EXISTS empleados (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nombre CHAR(50),
  puesto CHAR(20),
  fecha_nacimiento DATE,
  salario INT
```

```
);
```

Se ha aplicado el modificador IF NOT EXISTS. Su efecto es que crea la tabla en el caso de que no exista y si existe no muestra ningún mensaje de error. Este código, como hemos observado, lo guardaremos en un archivo llamado "curso_php.sql". Una vez guardado el script, para ejecutarlo utilizaremos el comando SOURCE, indicando la ruta completa del fichero:

```
mysql> SOURCE c:/CursoPHP/htdocs/practicaspHP/sql/curso_php.sql;
Query OK, 0 rows affected, 1 warning (0.00 sec)

Database changed
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

5.3.5 Operaciones con datos.

Ya tenemos la estructura que va a dar soporte a los datos. Ahora nos falta saber como podemos añadir, consultar, modificar y eliminar la información que contiene.

5.3.5.1 Introducir datos en una tabla.

De momento, las tablas que hemos creado están vacías y, por tanto, resultan prácticamente inútiles. En realidad, hasta ahora sólo hemos creado la cubierta del almacén, pero aún no hemos introducido nada útil en él.

Para insertar un registro en una tabla se usa la sentencia INSERT de SQL.

```
INSERT INTO tabla (campo1, campo2, ...) VALUES (dato1, dato2, ...);
```

Ejemplo 5.3.8. Vamos a insertar un primer dato en la tabla empleados. Al igual que en PHP los datos que son cadenas de texto los ponemos entre comillas, y del mismo modo deberemos actuar con la fecha. El campo id no hace falta insertarlo ya que lo hemos definido como autoincremental, y por tanto se rellena automáticamente.

```
mysql> INSERT INTO empleados(nombre, puesto, fecha_nacimiento, salario)
-> VALUES ("Silvia", "Administrativo", "1976-11-16", 1300);
Query OK, 1 row affected (0.01 sec)
```

Ejemplo 5.3.9. Se pueden insertar varios registros (filas) en una tabla con una sentencia INSERT, separando los conjuntos de valores entre paréntesis por comas.

```
mysql> INSERT INTO empleados(nombre, puesto, fecha_nacimiento, salario)
-> VALUES ("Ana", "Programador", "1967-04-06", 1500),
-> ("Juan", "Abogado", "1971-08-30", 1200),
-> ("Berta", "Comercial", "1980-11-12", 2000);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
INSERT INTO empleados(nombre, puesto, fecha_nacimiento, salario)
VALUES ('Elisa', 'programador', '1980-11-16', 1500),
('Pedro', 'Administrativo', '1975-11-11', 1100);
```

5.3.5.2 Leyendo datos de una tabla.

Aquí es donde se despliega la potencia de SQL, que además tiene una sintaxis muy

fácil de comprender. Los datos se leen de la tabla a través de consultas, que son preguntas que le hacemos a la base de datos.

La sentencia básica para hacer consultas es SELECT.

```
SELECT campo1, campo2, ... FROM tabla;
```

```
mysql> SELECT id, nombre, puesto, fecha_nacimiento, salario FROM empleados;
+----+-----+-----+-----+-----+
| id | nombre | puesto      | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 1  | Silvia | Administrativo | 1976-11-16      | 1300   |
| 2  | Ana    | Programador  | 1967-04-06      | 1500   |
| 3  | Juan   | Abogado      | 1971-08-30      | 1200   |
| 4  | Berta  | Comercial    | 1980-11-12      | 2000   |
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Ejemplo 5.3.10. Una consulta sobre la tabla empleados.

Ejemplo 5.3.11. En muchos casos querremos seleccionar todos los campos de la tabla. Para ello se usa el comodín asterisco " * ", de tal forma que esta consulta muestra exactamente el mismo resultado que la anterior.

```
SELECT * FROM tabla;
```

```
mysql> SELECT * FROM empleados;
```

id	nombre	puesto	fecha_nacimiento	salario
1	Silvia	Administrativo	1976-11-16	1300
2	Ana	Programador	1967-04-06	1500
3	Juan	Abogado	1971-08-30	1200
4	Berta	Comercial	1980-11-12	2000

```
4 rows in set (0.00 sec)
```

Ejemplo 5.3.12. En este caso restringimos el rango de la consulta al campo nombre.

```
mysql> SELECT nombre FROM empleados;
```

nombre
Silvia
Ana
Juan
Berta

```
4 rows in set (0.00 sec)
```

5.3.5.3 Filtrando datos de una tabla.

La cláusula WHERE en una consulta SELECT permite poner condiciones de manera que se nos muestren solo los registros que la cumplen. Esta es la sintaxis de la sentencia SELECT ampliada con esta cláusula.

```
SELECT campos FROM tabla WHERE condiciones;
```

Ejemplo 5.3.13. Esta consulta devuelve todos los datos de los empleados cuyo salario es mayor de 1400 euros.

```
mysql> SELECT * FROM empleados WHERE salario > 1400;
```

id	nombre	puesto	fecha_nacimiento	salario
2	Ana	Programador	1967-04-06	1500
4	Berta	Comercial	1980-11-12	2000

```
2 rows in set (0.01 sec)
```

Se pueden unir varias condiciones mediante los operadores lógicos AND, OR y NOT, cuyo funcionamiento es muy similar a PHP. Se pueden usar paréntesis si se tienen dudas sobre las reglas de precedencia.

Ejemplo 5.3.14. Empleados que cobran más de 1400 euros y que nacieron antes de 1975.

```
mysql> SELECT * FROM empleados WHERE salario > 1400
-> AND fecha_nacimiento < "1975-01-01";
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 2 | Ana | Programador | 1967-04-06 | 1500 |
+----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Ejemplo 5.3.15. Empleados que trabajan de comerciales o programadores.

```
mysql> SELECT * FROM EMPLEADOS WHERE puesto = "Comercial"
-> OR puesto = "Programador";
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 2 | Ana | Programador | 1967-04-06 | 1500 |
| 4 | Berta | Comercial | 1980-11-12 | 2000 |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Ejemplo 5.3.16. Búsqueda textual. En ocasiones se quiere buscar los registros cuyo texto contiene una cadena en concreto. Para ello, en lugar de comparar con el operador igual " = " se utiliza el operador LIKE y la cadena son la que se compara se pone entre comillas y signos de porcentaje "%cadena%".

Esta consulta muestra los registros cuyo nombre contiene la cadena "an".

```
mysql> SELECT * FROM empleados WHERE nombre LIKE "%an%";
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 2 | Ana | Programador | 1967-04-06 | 1500 |
| 3 | Juan | Abogado | 1971-08-30 | 1200 |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

5.3.5.4 Ordenar datos de una consulta.

En muchas ocasiones queremos que los datos devueltos nos lleguen ordenados. Para ello usaremos con el SELECT la cláusula ORDER BY a la que le acompaña el campo por el que queremos ordenar. En el caso de que haya una cláusula WHERE, ORDER BY debe ir necesariamente detrás. La nueva sintaxis sería la siguiente:

```
SELECT campos FROM tabla WHERE condiciones ORDER BY campo;
```

Ejemplo 5.3.17. Aquí se muestran todos los registros de la tabla ordenados por el nombre del empleado en orden alfabético.

```
mysql> SELECT * FROM empleados ORDER BY nombre;
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 2 | Ana | Programador | 1967-04-06 | 1500 |
| 4 | Berta | Comercial | 1980-11-12 | 2000 |
| 3 | Juan | Abogado | 1971-08-30 | 1200 |
| 1 | Silvia | Administrativo | 1976-11-16 | 1300 |
+----+-----+-----+-----+-----+
4 rows in set (0.02 sec)
```

Ejemplo 5.3.18. Mediante el modificador DESC se pueden devolver los resultados en orden inverso.

```
mysql> SELECT * FROM empleados ORDER BY salario DESC;
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 4 | Berta | Comercial | 1980-11-12 | 2000 |
| 2 | Ana | Programador | 1967-04-06 | 1500 |
| 1 | Silvia | Administrativo | 1976-11-16 | 1300 |
| 3 | Juan | Abogado | 1971-08-30 | 1200 |
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

5.3.5.5 Actualizar datos.

Además de añadir datos, necesitaremos saber como modificar los existentes. La sentencia UPDATE se encarga de esto. Se suele usar en combinación con una condición, similar a las que se usan en los SELECT, que filtra los registros a modificar.

```
UPDATE tabla SET campo = valor WHERE condiciones;
```

Ejemplo 5.3.19. Vamos a usar la sentencia UPDATE para subir el sueldo al empleado Ana. Como la tabla tiene un campo identificador, id, es recomendable que lo usemos para realizar el filtrado. Al final se hace un SELECT para comprobar que la tabla se ha actualizado.

```
mysql> UPDATE empleados SET salario = 1800 WHERE id = 2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 1 | Silvia | Administrativo | 1976-11-16 | 1300 |
| 2 | Ana | Programador | 1967-04-06 | 1800 |
| 3 | Juan | Abogado | 1971-08-30 | 1200 |
| 4 | Berta | Comercial | 1980-11-12 | 2000 |
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Ejemplo 5.3.20. Si no filtramos la actualización se verán afectados todos los registros de la tabla.

```
mysql> UPDATE empleados SET salario = 2500;
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 1 | Silvia | Administrativo | 1976-11-16 | 2500 |
| 2 | Ana | Programador | 1967-04-06 | 2500 |
| 3 | Juan | Abogado | 1971-08-30 | 2500 |
| 4 | Berta | Comercial | 1980-11-12 | 2500 |
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

En una sentencia UPDATE se pueden sustituir varios campos a la vez separándolos por comas.

```
UPDATE tabla SET campo1 = valor1, campo2 = valor2, ...
```

5.3.5.6 Borrado de registros de una tabla.

Para el borrado de datos de una tabla se usa la sentencia DELETE. Se suele usar en combinación con una condición, que filtra los datos a borrar, ya que si se usara sola borraría todos los datos de la tabla. Su uso tiene cierta similitud con el de SELECT, con la diferencia de que no hay que indicar campos, ya que se borra toda la fila.

```
DELETE FROM tabla WHERE condiciones;
```

Ejemplo 5.3.21. Borrado del empleado Juan. Una vez más se selecciona mediante el campo id.

```
mysql> DELETE FROM empleados WHERE id = 3;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM empleados;
+----+-----+-----+-----+-----+
| id | nombre | puesto | fecha_nacimiento | salario |
+----+-----+-----+-----+-----+
| 1 | Silvia | Administrativo | 1976-11-16 | 2500 |
| 2 | Ana | Programador | 1967-04-06 | 2500 |
| 4 | Berta | Comercial | 1980-11-12 | 2500 |
+----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Ejemplo 5.3.22. Al usar la sentencia DELETE sin filtrar se están borrando todos los datos de la tabla, aunque permanecerá su definición o estructura. Tras hacer el posterior SELECT, el mensaje “Empty set” nos indica que la tabla está vacía.

```
mysql> DELETE FROM empleados;
Query OK, 3 rows affected (0.00 sec)

mysql> SELECT * FROM empleados;
Empty set (0.00 sec)
```

5.4 Funciones de PHP con MySQL.

Una vez que hemos recordado las características básicas de MySQL y las sentencias SQL que se pueden ejecutar, **vamos a ver como se integra todo esto con PHP**.

Las definiciones de bases de datos y tablas las crearemos desde la consola de MySQL o bien desde PhpMyadmin, preferiblemente a través de un script SQL con las sentencias de creación e inicialización de tablas. La manipulación de datos en cambio se realizará desde PHP. Por lo tanto, las operaciones que nos interesan son:

- Leer datos.
- Añadir datos.
- Modificar datos.
- Eliminar datos.

PHP accede a MySQL a través del API. Dispone de una extensión o biblioteca de funciones con múltiples funciones, que nos permiten acceder a la base de datos.

5.4.1 Conectar con la base de datos.

Antes de empezar a consultar información tenemos que conectar con el sistema gestor de bases de datos y declarar la base de datos que vamos a usar. Esto se deberá hacer en cualquier script que posteriormente quiera leer, añadir, modificar o eliminar datos.

5.4.1.1 `mysqli_connect()`.

Esta función establece una conexión contra la bases de datos:

```
mysqli_connect(servidor, usuario, contraseña, base_de_datos)
```

Los parámetros que recibe son:

- ***servidor***: nombre o dirección IP del servidor donde está alojada la base de datos. Si está instalado en la misma máquina que el servidor web, lo normal es que sea localhost. Se puede elegir un número de puerto diferente del que se usa por defecto, poniendo detrás del servidor dos puntos " : " y el número del puerto. P.ej.: localhost:3340.
- ***usuario***: nombre de usuario con privilegios para acceder a la base de datos.
- ***contraseña***: contraseña del usuario.
- ***base_de_datos***: nombre de la base de datos a la que queremos acceder

Esta función devuelve una referencia a la base de datos que guardaremos en una variable, para usarla en posteriores funciones de PHP con MySQL.

5.4.1.2 `mysqli_select_db()`.

Usa como parámetros la base de datos que queremos seleccionar y la variable en la que tengamos la referencia a la base de datos. Es equivalente en MySQL a la

sentencia USE.

```
mysqli_select_db(referencia_bd, nombre_base_datos)
```

Esta función no nos hace falta utilizarla si utilizamos mysqli_connect() tal como vemos más arriba, donde se le indica el nombre de la base de datos a la vez que se establece la conexión.

Ejemplo 5.4.2. Una buena idea es guardar el proceso de conexión a la base de datos, y luego utilizarlo en nuestros scripts mediante un include(). Aquí se ha creado un posible ejemplo del fichero "conectar_bd.php". Aunque casi mejor si nos hacemos una función conectar en nuestra librería, y que nos devuelva la conexión, como os cuento más abajo.

```
<?php
define ("HOSTNAME", "localhost");
define ("DATABASE", "clientes");
define ("USER_DB", "root"); //esto para probar que funciona, nunca usar root
define ("PASSWORD", "");

$con=mysqli_connect (HOSTNAME,USER_DB,PASSWORD, DATABASE);

/* Comprueba la conexión */
if (mysqli_connect_errno()) {
    die("error en la conexión");
    //esto ( die() ) sólo es para probar si todo va ok,
    //en un caso real, habrá que actuar en
    //consecuencia...devolver un valor si estamos en una
    //función, redirigir a otra página..etc
}

?>
```

OJO, ESTO ES UN EJEMPLO PARA PROBAR LA CONEXIÓN. NO ACOSTUMBRARSE A USAR EL USUARIO ROOT EN CONEXIONES A LA BBDD DESDE PHP, PUEDE SER UN GRAN FALLO EN LA SEGURIDAD. LO IDEAL ES QUE COMO ADMINISTRADORES DE LA BASE DE DATOS, CREEMOS UN USUARIO CON LOS PERMISOS ADECUADOS, Y ESE USUARIO SERÁ EL QUE USARÁ EL SCRIPT DE PHP.

Se ha utilizado la función de die() de PHP que termina la ejecución del script sacando un mensaje por pantalla. Ojo, esto NO es operativo en un trabajo real, ya que se muestra una página en blanco con un mensaje de error, y esto nunca deberá encontrárselo un "navegante" en nuestro sitio web. Debemos controlar todos los posibles errores, como mostrar un mensaje de que lo intente más tarde, manteniendo nuestro "imagen" web en todo momento.

Es una buena estrategia, y es lo que os aconsejo que hagáis, crear la conexión en una función llamada "conexión_bd", o algo similar. En este caso la función debería devolver la referencia a la base de datos que se crea dentro de la función, para poder utilizarla fuera. Probadlo y utilizadlo.

```
<?php
include ("funciones.php");
$con=conectar_bd();

<?
```

5.4.2 Leer datos.

Aquí nos encontramos con la función clave de este capítulo: **mysqli_query()**. Su potencia es enorme ya que permite enviar a la base de datos una instrucción SQL, como las vistas en el capítulo anterior.

5.4.2.1 mysqli_query().

La sintaxis de esta función es muy sencilla. La sentencia SQL es la misma que pondríamos en la consola de MySQL, aunque no es necesario poner el punto y coma final de la sentencia SQL.

```
mysqli_query(referencia_bd, sentencia_SQL)
```

Hay que tener en cuenta que **esta función** sólo ejecuta la sentencia, no muestra ningún dato. Por ello **devuelve un identificador del resultado**, con el que podremos consultar los valores devueltos.

Ejemplo 5.4.3. Aquí se muestra como se ejecuta una sentencia SQL.

```
<?php
    include("conectar_bd.php");
    $con=conectar_db(); //si me he creado la función
    $res = mysqli_query($con, "SELECT * FROM empleados");
?>
```

Ejemplo 5.4.4. Mejoraremos la claridad del código si guardamos la sentencia SQL en una variable.

```
<?php
    include("conectar_bd.php");
    $con=conectar_db();
    $sql = "SELECT * FROM empleados";
    $res = mysqli_query($con, $sql);
?>
```

5.4.2.2 mysqli_num_rows() / mysqli_affected_rows

Antes de acceder al resultado nos puede interesar saber cuantos registros se han obtenido. Concretamente una consulta podría no devolver ningún resultado, y esto podría causar un error en nuestro código.

Esta función nos dará el número de filas que tiene un resultado obtenido **a partir de una consulta de tipo SELECT**. Para las consultas de **inserción, borrado y actualización** usaremos **mysqli_affected_rows(referencia_bd)** en su lugar (que también funciona para sentencias select).


```
mysqli_num_rows(identificador_de_resultado)
```

Utilizas estas funciones para comprobar que realmente se han modificado los datos en la BD, ¿realmente se ha borrado algún registro, aunque no me devuelva ningún error la Base de Datos?...etc

5.4.2.4 `mysqli_fetch_array()`.

Usando esta función a la hora de recuperar datos, se obtiene un array asociativo por cada fila que tiene como índices los nombres de los campos. En cada llamada a esta función se recupera una de las filas resultado de la consulta, hasta que no queda ninguna y devuelve FALSE.

```
mysqli_fetch_array(identificador_de_resultado)
```

[Aquí](#) os explica su uso y sus posibilidades

Echad un vistazo también a `mysqli_fetch_assoc()` (¿qué diferencia hay entre ellas?)

Ejemplo 5.4.10. A la hora de utilizar esta función, como cada llamada a la función recupera una fila y pasa a la siguiente, no es necesario recuperar el número de filas que contiene el resultado mediante `mysqli_num_rows()`.

```
<?php
    include("conectar_bd.php"); ?>
<table border="1">
<tr>
<td>id</td>
<td>nombre</td>
<td>puesto</td>
<td>fecha_nacimiento</td>
<td>salario</td>
</tr>
<?php
    $con=conectar_db();
    $sql = "SELECT * FROM empleados";
    $res = mysqli_query($con,$sql);
    $fila = mysqli_fetch_array($res);
    while($fila) {
        echo "<tr>";
        echo "<td>{$fila["id"]}</td>";
        echo "<td>{$fila["nombre"]}</td>";
        echo "<td>{$fila["puesto"]}</td>";
        echo "<td>{$fila["fecha_nacimiento"]}</td>";
        echo "<td>{$fila["salario"]}</td>";
        echo "</tr>";
        $fila = mysqli_fetch_array($res);
    }
?>
</table>
```

o mejor todavía podemos hacer algo así:

```
<?php
    $sql = "SELECT * FROM empleados";
    $res = mysqli_query($con,$sql);
    $con=conectar_db();

    while($fila = mysqli_fetch_array($res)) {
        echo "<tr>";
        echo "<td>{$fila["id"]}</td>";
        echo "<td>{$fila["nombre"]}</td>";
        echo "<td>{$fila["puesto"]}</td>";
        echo "<td>{$fila["fecha_nacimiento"]}</td>";
        echo "<td>{$fila["salario"]}</td>";
        echo "</tr>";
    }
?>
```

5.4.3 Añadir datos.

Para añadir datos no necesitamos conocer nada nuevo. Se usa la función `mysqli_query()` pero en esta ocasión se encapsula una sentencia INSERT de SQL, para que añada datos.

Ejemplo 5.4.11. Vamos a añadir un nuevo empleado. Para ello se usará la sentencia SQL "parametrizada", como si se hubiesen recibido los datos de un formulario y se quisieran guardar en la base de datos. Como la consulta es muy larga, se ha troceado en varias líneas usando el operador concatenar cadena ".".

```
<?php
    include("conectar_bd.php");
    $con=conectar_db();
    $nombre = "Paola";
    $puesto = "Programador";
    $fecha_nacimiento = "1958-01-23";
    $salario = 1800;
    $sql = "INSERT INTO empleados "
        . "(nombre, puesto, fecha_nacimiento, salario) "
        . "VALUES "
        . "('$nombre', '$puesto', '$fecha_nacimiento', $salario)";
    $res = mysqli_query($con, $sql);
?>
```

Podemos comprobar que se ha realizado la inserción tecleando "SELECT * FROM empleados" desde la consola de MySQL.

5.4.4 Modificar datos.

Una vez más usamos la función `mysqli_query()`, con una sentencia SQL del tipo UPDATE.

Ejemplo 5.4.12. Vamos a modificar los datos del empleado que añadimos en la consulta anterior.

```
<?php
    include("conectar_bd.php");
    $con=conectar_db();

    $id = 5; //id del registro a modificar
    $nuevo_salario = 2100;
    $sql = "UPDATE empleados SET salario = $nuevo_salario "
        . "WHERE id = $id";
    $res = mysqli_query($con, $sql);

?>
```

5.4.5 Eliminar datos.

En esta ocasión usamos la función `mysqli_query()`, con una sentencia SQL del tipo DELETE.

Ejemplo 5.4.13. Vamos a modificar los datos del empleado que añadimos en la consulta anterior.

```
<?php
    include("conectar_bd.php");
    $con=conectar_db();
    $id = 5; //Id del registro a borrar
    $sql = "DELETE FROM empleados WHERE id = $id";
    $res = mysqli_query($con, $sql);

?>
```

Ante cualquier duda con las funciones, o posible errores en este documento, acudid como siempre al manual de [php on-line](#), ahí explica el uso de todas y cada una de las funciones php. Haced uso de él, ante cualquier problema o duda de cómo o cuándo se ha de utilizar una función.

Y es importante, cuando utilizamos la función `mysqli_query()`, **asegurarnos que se ha ejecutado correctamente**. Para ello podemos aprovechar el valor que nos devuelva esta función: Retornará FALSE en caso de error. Si la consulta del tipo Select es exitosa, retornará un objeto `mysqli_result` para que podamos navegar por sus resultados, como ya sabemos. Para otras consultas exitosas retornará TRUE.