TEMA 7. ACCESO A BASE DE DATOS CON PHP

PHP ofrece interfaces para el acceso a la mayoría de las bases de datos existentes. Hay bases de datos de código abierto, como MySQL, comerciales propietarias como Oracle y además tiene librerías para acceso a datos por ODBC, lo que nos permite comunicar con todas las bases de datos posibles en sistemas Microsoft, como Access o SQL Server.
Gracias a las funciones existentes para cada sistema gestor de base de datos, podremos realizar cualquier acción con los datos que necesitemos.
Esta interacción se realiza, por un lado, a partir de las funciones que PHP nos propone para cada tipo de base de datos y, por otro utilizando SQL.

MySQL es la base de datos más extendida en combinación con PHP debido a su gratuidad, eficiencia y simplicidad.

phpMyAdmin es una aplicación web desarrollada en PHP que ofrece una interfaz para gestionar MySQL.

charset y el cotejamiento

La mayoría de los datos que se almacenan en las tablas son textos (aunque también se pueden almacenar números y datos binarios), por lo que es fundamental establecer cómo se deben codificar esos datos (ANSI, UTF-8). En la nomenclatura de MySQL se denomina charset al método de codificación y cotejamiento (*collation*) al criterio utilizado para realizar comparaciones entre textos.

Para evitar problemas podríamos sentirnos tentados a almacenar todos nuestros datos de texto en formato UTF-8, pero debemos tener en cuenta que esto supondrá multiplicar por 3 las necesidades de almacenamiento de algunos campos inútilmente.

Tipos de campos en MySQL

Los tipos de campos en MySQL pueden clasificarse fundamentalmente en tres categorías: numéricos, fechas - horas y textos.

Es importante elegir el tipo adecuado para cada dato

- Numéricos

Se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

TinyInt: es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255

Bit ó Bool: un número entero que puede ser 0 ó 1

SmallInt: número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

MediumInt: número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.

Integer, Int: número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

BigInt: número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

Float: número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

xReal, Double: número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308

Decimal, Dec, Numeric: Número en coma flotante desempaquetado. El número se almacena como una cadena

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE	8 bytes

PRECISION	
REAL	8 bytes
DECIMAL(M,D	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

- **Fecha**

  A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.

  Date: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

  DateTime: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

  TimeStamp: Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

Tamaño	**Formato**
14	AñoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AñoMesDiaHoraMinutoSegundo aammddhhmmss
8	ñoMesDia aaaammdd
6	AñoMesDia aammdd
4	AñoMes aamm
2	Año aa

Time: almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

Year: almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155.

El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

- C**adena:**

  Char(n): almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

  VarChar(n): almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

  Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object)

  La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta.

  Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

  TinyText y TinyBlob: Columna con una longitud máxima de 255 caracteres.

  Blob y Text: un texto con un máximo de 65535 caracteres.

  MediumBlob y MediumText: un texto con un máximo de 16.777.215 caracteres.

  LongBlob y LongText: un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

  Enum: campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos

Set: un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.

Tipo de campo	Tamaño de Almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LONGBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

Operaciones básicas del lenguaje SQL

Podemos trabajar a nivel de registros, de tablas o de bases de datos:

Gestión de registros

Las principales instrucciones de gestión de registros en SQL son:

- SELECT: Seleccionar campos que cumplan un criterio.
- INSERT: Insertar registros en una tabla.
- UPDATE: Actualizar valores de campos dentro de registros.
- DELETE: Borrar registros.

Gestión de tablas

Las principales instrucciones de gestión de tablas son:

- CREATE TABLE: Crear tablas.

```
CREATE TABLE usuarios (id_credenciales INT NOT NULL AUTO_INCREMENT
PRIMARY KEY,identifivativo VARCHAR(255)NOT NULL ,clave CHAR(32) NOT
NULL);
```

- ALTER TABLE: modifica columnas: añadirlas, borrarlas, modificarlas.

```
ALTER TABLE usuarios ADD COLUMN nombre_real VARCHAR(255);
```

- DROP TABLE: Eliminar una tabla

```
DROP TABLE usuarios;
```

- TRUNCATE TABLE: Elimina sólo los registros de una tabla, no su estructura. Si la tabla contenía campos autonuméricos, se reinicializarán al valor 1.

```
TRUNCATE TABLE usuarios;
```

- RENAME TABLE: Cambia el nombre de una tabla.

```
RENAME TABLE usuarios TO usuarios_copia;
```

Nota: Para duplicar una tabla puede utilizarse una instrucción SELECT anidada dentro de una CREATE TABLE; por ejemplo: CREATE TABLE copia SELECT * FROM original;.

Conectar con una base de datos MySQL mediante PHP

PHP 5 y versiones posteriores pueden funcionar con una base de datos MySQL usando:

- **Extensión MySQLi** (la "i" significa mejorado). Que a su vez puede ser orientado a objetos o procedural. Ambos modos sólo son válidos para conectar con una base de datos de MySQL.
- **PDO (objetos de datos PHP):** Se puede conectar con 10 tipos distintos de bbdd sólo cambiando la cadena de conexión.

Nosotros usaremos MySqli procedural.

Usaremos las funciones de la librería MSQLi, y normalmente seguiremos el siguiente esquema:

1. Establecer la conexión con el servidor de bases de datos, se puede incluir la bbdd con la que conectamos como último parámetro:

   $conexion=mysqli_connect ($servidor, $usuario, $clave [, $base_de_datos);

2. Realizar una consulta. Se crea un String $query con la consulta a realizar (insert, delete, update, select…)

   $resultado=mysqli_query ($conexion, $query);

   Ten en cuenta que por motivos de seguridad mysqli_query envía una sola consulta (no puede enviar varias consultar a la vez).

3. Procesar los resultados de la consulta
4. Cerrar la conexión

   mysqli_close($conexión);

Gestión de los resultados de las consultas

A continuación se describen las funciones de la librería Mysqli más útiles para gestionar los resultados de una consulta:

mysqli_fetch_assoc	*array mysqli_fetch_assoc (resource $resultado)* Devuelve un array asociativo con una fila del resultado de la consulta, o FALSE si ya no quedan filas por extraer.
mysqli_fetch_array	*array mysqli_fetch_array (resource $resultado [, int $tipo_array = MYSQL_BOTH])* Produce exactamente el mismo resultado que la anterior, pero a través del parámetro opcional $tipo_array nos permite elegir entre un array numérico (MYSQL_NUM), un array asociativo (MYSQL_ASSOC) o ambos (MYSQL_BOTH).
mysqli_num_rows	*int mysql_num_rows (resource $resultado)* Nos indica cuántas filas tiene el resultado de una consulta SELECT.
mysqli_num_fields	*int mysql_num_fields (resource $result)*

	Nos indica cuántas columnas contiene el resultado de una consulta SELECT.
mysqli_affected_rows	*int mysqli_affected_rows ([resource $conexión])* Nos indica el número de filas afectadas por la última instrucción INSERT, UPDATE o DELETE realizada a través de conexión.
mysqli_insert_id	*int mysqli_insert_id ([resource $conexion])* Generalmente los campos autonuméricos no se especifican al insertar registros, sino que se permite que el propio gestor de bases de datos los genere. En este caso podemos saber cuál ha sido el último id generado mediante la instrucción mysql_insert_id.
mysqli_real_scape_string	*string mysql_real_escape_string ($conexion, string $dato)* Escapa el dato para que sea seguro utilizarlo en una consulta y evitar inyecciones SQL (encontrará más información en la sección siguiente).
mysqli_set_charset	*bool mysqli_set_charset (string $charset [, resource $conexión])* Establece qué codificación utilizará PHP para los datos que envíe a través de conexión. Puede consultar los juegos de caracteres que reconoce MySQL en http://dev.mysql.com/doc/refman/5.6/en/charset-charsets.html.
mysqli_error	*string mysqli_error ([resource $conexión])* Devuelve el último mensaje de error generado por el gestor de bases de datos.

Gestión de privilegios en MySQL

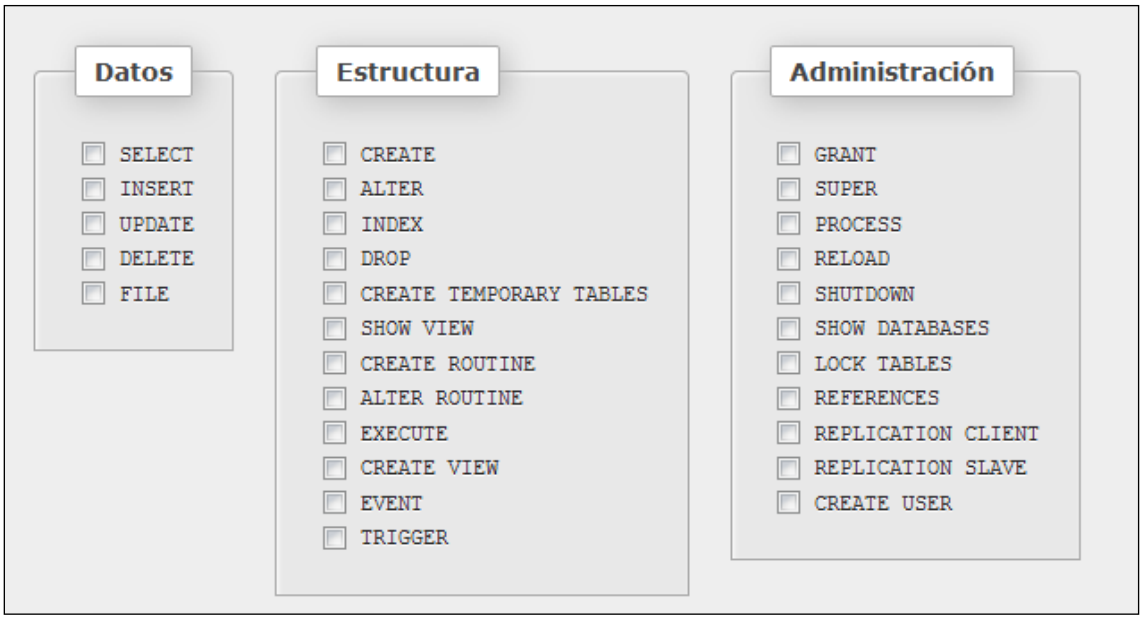
El control de acceso al gestor de bases de datos relacionales MySQL se basa en el uso de usuarios y credenciales, a los que se les pueden conceder privilegios a nivel global, a nivel de base de datos, a nivel de tabla y a nivel de columna.

Estos usuarios se almacenan en la tabla user de la base de datos mysql.

Una cuestión que suele causar confusión es la presencia del campo servidor (*host*) al crear un usuario nuevo en MySQL. Este campo puede rellenarse con el comodín %, que indica que un usuario concreto puede acceder desde cualquier servidor (incluido localhost), pero como por defecto MySQL incluye una definición de usuario más específica para los conexiones desde localhost (usuario: cualquiera, host: localhost) con unos privilegios muy limitados, se aplica esta definición y ese usuario concreto no puede acceder desde localhost. Para resolver esto podemos crear otra definición de ese usuario concreto eligiendo expresamente como servidor localhost (o borrar la definición más específica de MySQL).

Los privilegios de MySQL se clasifican en 3 grupos:

- Privilegios de administración.
- Privilegios de estructura.
- Privilegios de datos.



Los privilegios de datos están directamente relacionados con las instrucciones correspondientes de SQL (SELECT, INSERT, UPDATE y DELETE) explicadas anteriormente.

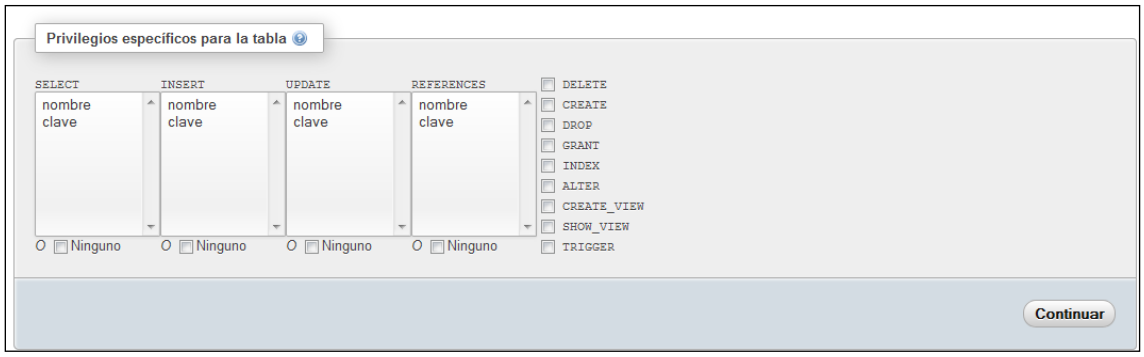
Los privilegios de estructura más importantes son CREATE, ALTER y DROP, que igualmente están relacionados con las correspondientes instrucciones SQL explicadas anteriormente. No obstante, el otorgamiento de estos privilegios no implica directamente que el usuario

vaya a poder realizar la operación correspondiente; por ejemplo, para poder hacer ALTER TABLE, el usuario deberá tener CREATE e INSERT además del ALTER; y para renombrar una tabla necesitará ALTER, DROP y CREATE.

Por último, el privilegio de administración más importante es GRANT, que nos permite otorgar o retirar (mediante la instrucción SQL REVOKE) a otros usuarios los permisos que nosotros poseemos.

Una limitación importante del sistema de privilegios de MySQL es que no nos permite otorgar los privilegios DROP y CREATE a nivel de tabla; si un usuario dispone de alguno de estos privilegios a nivel de tabla también podrá aplicarlo a nivel de la base de datos a la que pertenece la tabla, es decir podrá borrar o crear la base de datos (aunque aparentemente no pueda a través de la interfaz de phpMyAdmin, sí podría hacerlo a través de la consola).

En phpMyAdmin, para configurar los privilegios haremos clic sobre la lengüeta **Privilegios** en la página inicial, que nos conducirá a un formulario en el que podremos configurar los privilegios globales, o elegir una base de datos concreta sobre la que configurar sus privilegios. Si optamos por esta segunda posibilidad accederemos a otro formulario en el que podremos configurar los privilegios a nivel de base de datos, o elegir una tabla concreta sobre la que configurar sus privilegios. Si optamos por esta segunda posibilidad accederemos a un último formulario en el que podremos establecer los privilegios a nivel de tabla y de campo.



Seguridad

La seguridad en las bases de datos es un asunto crucial. Deberíamos aplicar siempre los siguientes principios generales:

- Validar los datos para asegurarnos de que antes de salir de PHP son del tipo que se espera en MySQL.

- Sanear los datos para asegurarnos de que antes de salir de PHP tienen el formato que espera MySQL (por ejemplo, las fechas en formato YYYY-MM-DD).
- Limitar la longitud de los datos. No permita que se introduzcan textos demasiado largos en su base de datos si no es estrictamente necesario; podrían ser códigos maliciosos.
- Escapar todas las cadenas de caracteres con mysql_real_scape_string.

  Inyección SQL: se inserta código SQL dentro del código ya programado con el fin de alterar el funcionamiento del programa y lograr que se ejecute el código invasor

```
consulta = "SELECT * FROM usuarios WHERE nombre='$nUsuario'";
```

  Si el operador escribe un nombre la aplicación generaría una sentencia correcta, en donde se seleccionarían todos los registros con dicho nombre en la base de datos:

```
SELECT * FROM usuarios WHERE nombre = 'xxxx';
```

  Pero si un operador malintencionado escribe como nombre de usuario a consultar:

```
"Pepe; SELECT * FROM datos WHERE nombre LIKE '%"
```

  se generaría la siguiente consulta SQL

```
SELECT * FROM usuarios WHERE nombre = 'Pepe';SELECT * FROM datos
WHERE nombre LIKE '%';
```

  En la base de datos se ejecutaría la consulta en el orden dado, se seleccionarían todos los registros con el nombre 'Pepe', y se seleccionaría toda la tabla "datos", que no debería estar disponible para los usuarios web comunes.

```
mysql_real_escape_string($nombre_usuario)
```

- Usar un usuario para la conexión de PHP que posea los privilegios mínimos para realizar las operaciones que necesitemos.
- No almacenar datos que realmente no sea necesario. Por ejemplo, no almacene un número de tarjeta de crédito si no va a usarlo en el futuro.
- Si sólo necesitamos saber si un dato se ha escrito correctamente (por ejemplo una contraseña), pero no necesitamos saber exactamente cuál es ese dato, podemos almacenarlo hasheado en la base de datos (por ejemplo con MD5).

  Si en nuestra página web tenemos un sistema de usuarios y queremos proteger las contraseñas para prevenir posibles vulnerabilidades en nuestro servidor, es una medida eficaz encriptar las contraseñas, de manera que si alguien puede acceder a ellas no pueda ver la contraseña si no su encriptación.

Es un algoritmo de encriptación de un solo sentido, no se puede desencriptar de ninguna manera.

Para guardar la contraseña encriptada en md5, usaremos la funcion md5() de PHP:

```
$contrasenna=md5($_REQUEST['contrasenna']);
```

Tendremos que encriptar la contraseña introducida para poderla comparar con la de la bbdd

Pero si no usamos una transmisión segura cuando el usuario envía los datos al servidor, la contraseña es enviada sin encriptar, en ese momento puede ser capturada por un tercero. Para evitar esto, podemos encriptar la clave en el ordenador del cliente usando JavaScript.

- Si hay que almacenar datos sensibles, y debemos conocerlos podemos encriptarlos previamente en PHP mediante la librería Mcrypt

  (http://php.net/manual/es/book.mcrypt.php).