estudios abiertos
SEAS
GRUPO SANVALERO

2 UNIDAD DIDACTICA

PHP

2. MySQL

ÍNDICE

OBJETIVOS 43
INTRODUCCIÓN 44
2.1. Primeros pasos con MySQL 45
2.1.1. Arrancar el servidor 46
2.1.2. Conexión y desconexión al servidor MySQL 49
2.1.3. Creación y manejo de bases de datos 52
2.2. Tipos de datos 62
2.2.1. Cadenas de caracteres 62
2.2.2. Datos numéricos 64
2.2.3. Fecha y hora 66
2.2.4. Imágenes 68
2.3. Operadores 69
2.3.1. Operadores lógicos 69
2.3.2. Operadores aritméticos 70
2.3.3. Operadores DE COMPARACIÓN 71
2.4. Funciones 73
2.4.1. Funciones de cadenas de texto 73
2.4.2. Funciones matemáticas 77
2.4.3. Funciones de fecha y hora 81
2.4.4. Manejo de fechas en MySQL 81
2.5. Usuarios y privilegios 87
2.5.1. Nombres de usuario y contraseñas 87
2.5.2. Sistema de privilegios 87
2.5.3. Añadir cuentas de usuario a MySQL 89
2.5.4. Eliminar cuentas de usuario a MySQL 91
2.5.5. Limitar cuentas de usuario 92
2.5.6. Mostrar permisos de usuario 92
2.5.7. Crear contraseñas a una cuenta 92
2.6. Motores de almacenamiento 94
2.6.1. Motor de almacenamiento MyISAM 94
2.6.2. Motor de almacenamiento InnoDB 96
2.7. Conjunto de caracteres y colaciones 98
2.7.1. Tipos 98
2.7.2. Elección de cotejamiento 99
2.8. Gestión gráfica de MySQL – PHPMyAdmin 101
2.8.1. Acceso a la aplicación 101
2.8.2. Crear una base de datos 101
2.8.3. Gestión de tablas 102
2.8.4. Gestión de registros 104
2.8.5. Importar y exportar bases de datos 106
2.8.6. Pestaña de operaciones 108
2.8.7. Generar código PHP 109
RESUMEN 111

OBJETIVOS

- Introducir el trabajo en modo consola para administrar bases de datos MySQL.
- Aprender a administrar usuarios y conceder o revocar permisos.
- Conocer los principales tipos de datos con los que podemos trabajar con MySQL.
- Conocer las principales operaciones que podemos realizar con bases de datos.
- Explicar características técnicas relativas a bases de datos.
- Trabajar con bases de datos MySQL desde entorno gráfico.

INTRODUCCIÓN

Antes de comenzar a crear scripts para programación en PHP, es necesario tener unos conocimientos básicos, pero sólidos en cuanto a las principales operaciones que podemos realizar con bases de datos, así como de la estructura con la que pueden ser creadas y sus diferentes tipos de datos.

En esta unidad vamos a aprender a crear bases de datos, tablas y registros. Veremos las diferentes operaciones de mantenimiento que podemos realizar sobre las mismas, desde la consola de MySQL o desde el entorno gráfico mediante la herramienta PHPMyAdmin.

2.1. Primeros pasos con MySQL

En la unidad 1 ya hemos instalado nuestro servidor de bases de datos MySQL. En esta unidad vamos a comenzar a trabajar con bases de datos.

Una base de datos es un conjunto de información relacionada y organizada con un propósito. Utilizando una base de datos podremos llevar el control de los préstamos de libros en una biblioteca, crear una agenda de contactos telefónicos y direcciones, llevar el control de pedidos, clientes y proveedores de nuestra empresa.

Las bases de datos permiten, para almacenar y estructurar la información de manera que sea fácil, acceder a los datos almacenados en las mismas. Las bases de datos electrónicas permiten trabajar cómodamente con grandes volúmenes de información, editar la información almacenada, realizar consultas, informes, etc.

Algunos de los términos que se suelen utilizar cuando hablamos de bases de datos, son:

- Tabla: es un conjunto de datos organizados en filas y columnas. Se debe crear una tabla para cada conjunto de datos en la base de datos: una tabla para la información relativa a clientes, otra para proveedores, otra para pedidos...
- Campo: es una unidad simple (o columna) de información en una tabla. Una tabla puede tener varios campos donde se almacenarán las diferentes características de cada elemento de la tabla. Por ejemplo, en la tabla Clientes, que contiene los datos de todos los clientes de nuestra empresa, podemos crear los campos Nombre, Apellidos, Teléfono, Dirección...
- Registro: es el conjunto de todos los campos de datos de una fila en una tabla. Todos los datos referidos al mismo cliente, campos Nombre, Apellidos, Teléfono, Dirección... formarían un registro.
- Base de datos: es el conjunto de todas las tablas y demás objetos, tales como formularios, consultas o informes, que se utilizan para gestionar datos. Una base de datos comprende un juego de ficheros de datos, a los cuales podemos acceder gracias a los Sistemas de Administración de Bases de Datos (DBMS).

En esta unidad, veremos todas las operaciones distintas que podremos realizar para administrar una base de datos, junto con sus tablas, registros y campos.

Para realizar todas estas operaciones, MySQL se apoya en el lenguaje estándar de consulta estructurado SQL. SQL tiene un conjunto de comandos predefinidos a partir de los cuales podremos realizar todas las operaciones relativas al mantenimiento de una base de datos.

SQL no es propiedad de MySQL, sino que al ser un estándar, también es utilizado por otras bases de datos como Oracle, PostgreSQL, Sybae y Microsoft SQL entre otros.

Existe un estándar ANSI de SQL, y los sistemas de bases de datos como MySQL suelen implementarlo. No obstante, existirán diferencias sutiles entre el SQL estándar y el SQL de MySQL. Nosotros utilizaremos este último para realizar todas nuestras operaciones.

2.1.1. Arrancar el servidor

En la unidad 1 ya instalamos MySQL como servidor de bases de datos. En este momento, está a la espera de que le enviemos comandos para realizar consultas.

Cuando lleguemos a la unidad 5, explicaremos cómo conectarse a un servidor MySQL desde nuestros propios scripts en PHP. Pero obviamente ésta no es la única manera de conectarse a un servidor de bases de datos. MySQL incorpora su propia consola a partir de la cual podremos realizar las mismas operaciones de conexión al servidor, realizar consultas o instrucciones de altas, bajas o modificaciones. Todos los comandos que vamos a ver en esta unidad podrán ser ejecutados desde dicha consola.

Para comenzar, será preciso levantar el servicio que hará posible que MySQL se ejecute en nuestro equipo. Con la utilidad XAMPP ya hemos explicado cómo instalarlo como servicio para que se inicie en el arranque del sistema, pero a continuación, vamos a explicar cómo realizar la misma operación, pero desde la línea de comandos.

El comando "*mysqld*", ejecutado desde la línea de comandos, será el que nos levantará el servicio MySQL. Para poder ejecutarlo, tendremos que acompañarlo de ruta absoluta (ruta completa en el directorio) o bien ejecutarlo desde su propio directorio.

Al ejecutarlo, queda trabajando en un segundo plano, tal y como podemos ver en la Figura 2.1, y esto significará que para seguir realizando operaciones, tendremos que abrir una nueva consola para seguir trabajando:

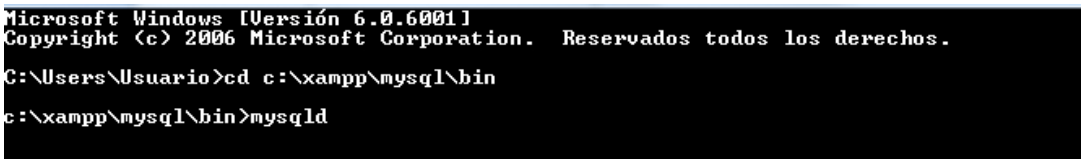

Figura 2.1. Comando MYSQLD.

Cuando queramos detener el servicio MySQL, tendremos que ejecutar el siguiente comando:

EJEMPLO

```
MYSQLADMIN -U ROOT -P SHUTDOWN
```

La utilidad administrativa ***mysqladmin***, se conectará al servidor con el usuario y password que le proporcionemos y finalizará su ejecución.

Si por alguna razón no consiguiéramos iniciar ***mysqld,*** podríamos buscar en el registro de errores las posibles causas del fallo. El registro tendrá como nombre el propio nombre del equipo en el que se esté ejecutando el servidor, con una extensión ***.err***, y lo podremos encontrar en el directorio ***DATA*** de MySQL:

Figura 2.2. Registro de errores.

Aún tendríamos dos opciones más de poder consultar posibles errores:

Si iniciamos el servidor con el comando "*mysqld –console*", podremos obtener alguna información extra en la pantalla.

Mediante la opción "*mysqld –standalone –debug"*, mysqld guardará un registro con nombre "mysqld.trace", con información extra sobre los errores que se estén produciendo.

Por último, si quisiéramos conocer la versión de MySQL que tenemos instalada para nuestro servidor, tendríamos que ejecutar el siguiente comando:

EJEMPLO

```
MYSQL --VERSION
```

Obteniendo un resultado como el siguiente:

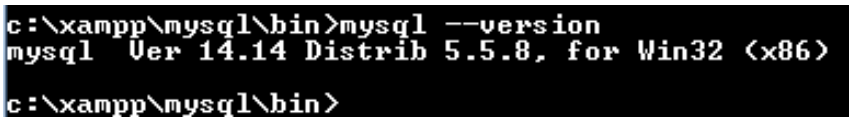

Figura 2.3. Versión de MySQL.

Antes de terminar este capítulo, será necesario conocer que no todas las operaciones tienen que por qué ser realizadas desde modo consola. Una de las ventajas de utilizar aplicaciones AMP como XAMPP, es que incorporan un panel de control donde, por ejemplo, poder gestionar los servicios de Apache y MySQL, así como arrancar los módulos de Apache y MySQL, y otros como el cliente FTP Filezilla.

En la siguiente imagen, podemos comprobar cómo tenemos detenidos los módulos de Apache y MySQL. Ambos están esperando a que pulsemos el botón "Start" para arrancarlos, y a partir de ese momento, poder comenzar a realizar nuestras operaciones:

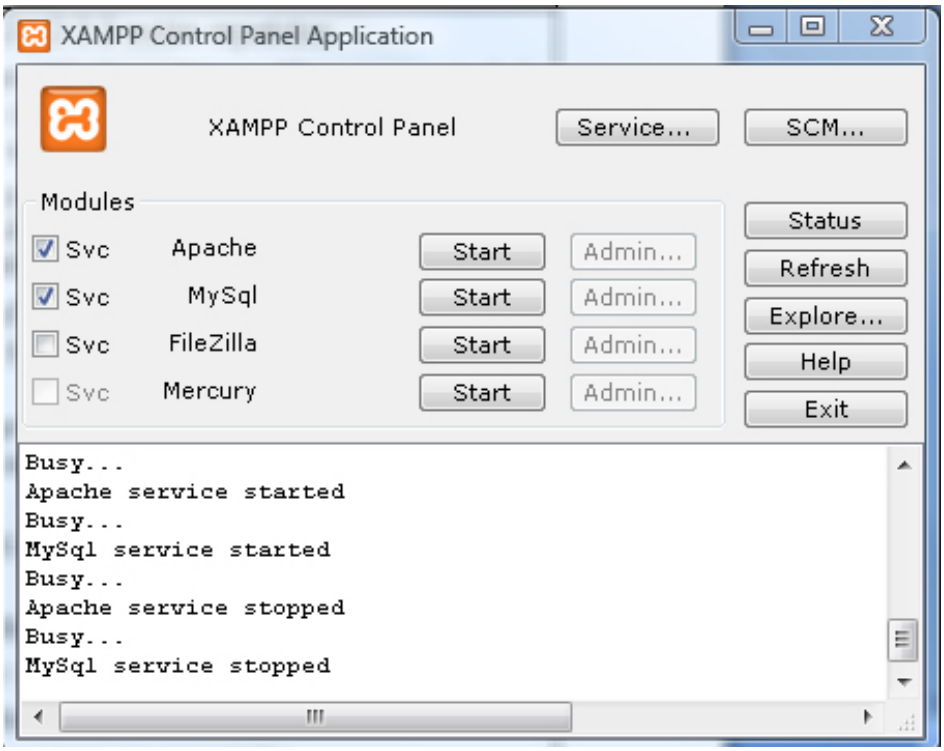

Figura 2.4. Administración desde el panel de control XAMPP.

Otra de las opciones importantes que incorpora este panel es la posibilidad de que XAMPP arranque al inicio del sistema, indicándole que servicios queremos que inicie al comienzo. Para acceder a esta opción tendremos que pulsar el botón "Service":

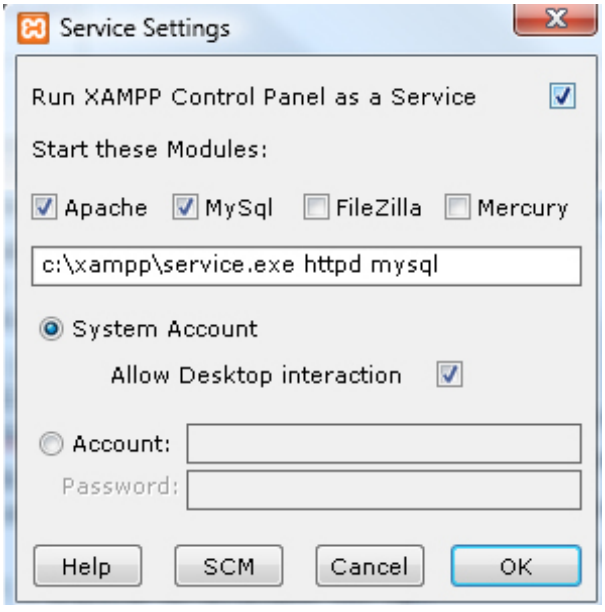

Figura 2.5. Administración de servicios en XAMPP.

2.1.2. Conexión y desconexión al servidor MySQL

El primer paso siempre será establecer una conexión con el servidor de nuestra base de datos. En el proceso de instalación del servidor MySQL, hemos tenido que observar que hemos obtenido un usuario y una contraseña, que serán necesarios para la conexión al servidor, junto con el propio nombre de "host" del servidor (por ejemplo, localhost).

Una vez que tenemos estos parámetros preparados, tendremos que conocer la sintaxis de conexión al servidor de bases de datos, que sería la siguiente:

EJEMPLO

```
mysql -h host -u user -p
```

En nuestro ejemplo, si nos queremos conectar al servidor que tenemos instalado en nuestro equipo, cuyo nombre de host es "localhost", el usuario es "root", y no tiene contraseña asignada de momento, tendríamos que escribir el siguiente comando:

EJEMPLO

```
mysql -h localhost -u root -p
```

Y a partir de ese momento, ya estaríamos conectados al servidor MySQL, el cual nos ofrece su propia consola para que le introduzcamos comandos, tal y como podemos ver en la siguiente imagen:

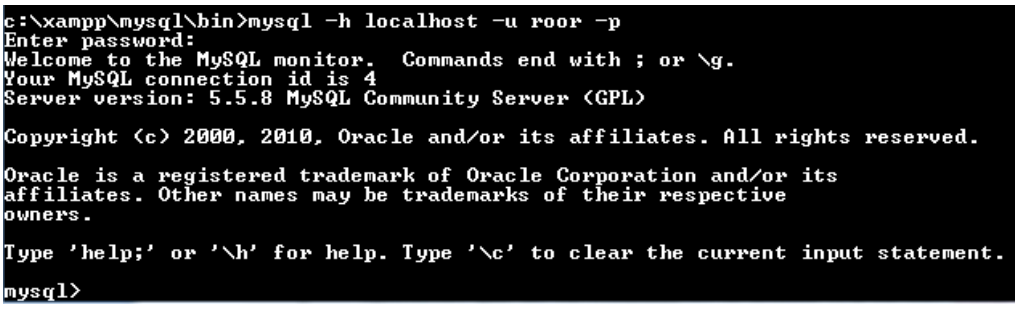

Figura 2.6. Conexión a consola MySQL del servidor.

En la unidad 5, donde explicaremos la forma de trabajo de PHP con bases de datos MySQL, comprobaremos que esta misma operación de conexión y desconexión al servidor de bases de datos, la podremos realizar con funciones como **mysql_connect($host,$user,$password),** donde cada uno de los argumentos que le pasamos a la función, son los mismos que hemos utilizado para realizar la misma conexión en modo consola. Es decir, el servidor, el usuario y su contraseña.

Por último, tenemos alternativas también para realizar esta conexión con aplicaciones de entorno gráfico como **MySQL Query Browser** o **MySQL Workbench.**

Si quisiéramos utilizar la herramienta **MySQL Query Browser,** en primer lugar, tendríamos que descargar la aplicación para el sistema operativo que estemos trabajando, desde el siguiente enlace:

http://dev.mysql.com/downloads/mirror.php?id=403402#mirrors

Una vez descargada, instalada e iniciada la aplicación, nos aparecerá una pantalla de conexión, donde nos pedirá que le introduzcamos los datos de servidor, puerto de conexión, usuario y contraseña, como en las anteriores soluciones. Por probar una manera distinta de conexión, imaginemos que el nombre de usuario es **service_manager.** Tendremos que configurar el acceso de la siguiente forma:

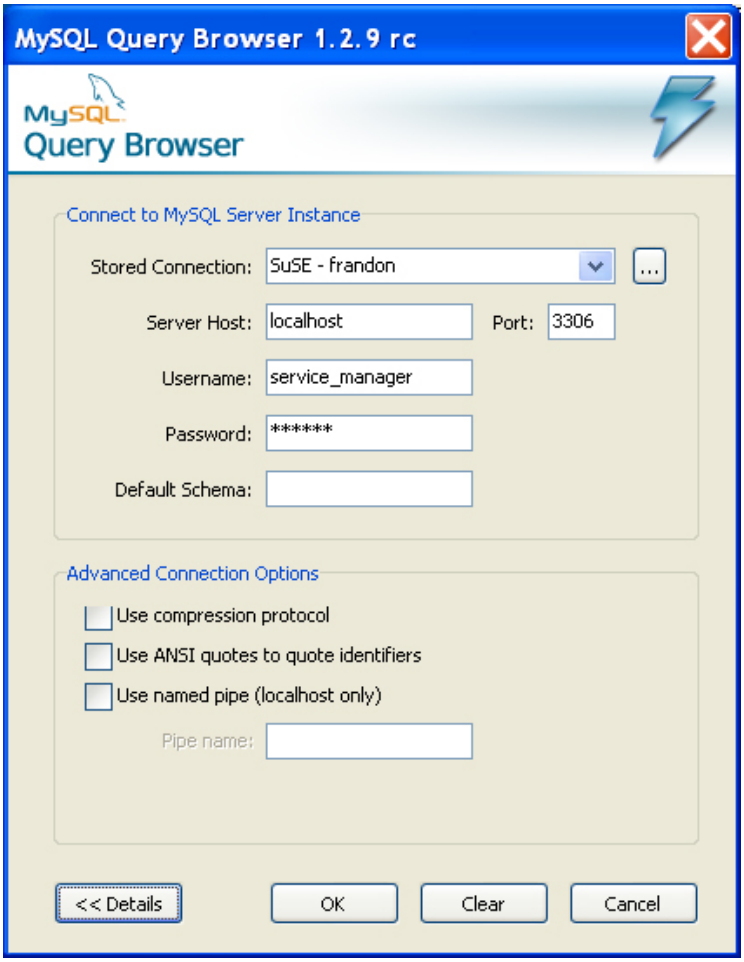

Figura 2.7. Conexión a MySQL desde entorno gráfico.

Si quisiéramos conectarnos a un servidor que no fuera el local y estuviera ubicado en un servidor externo, tendríamos que sustituir el valor “localhost” por el nombre de la máquina donde se está ejecutando MySQL o su dirección IP correspondiente.

Cuando decidamos que queremos terminar nuestra conexión con el servidor MySQL, simplemente tendremos que ejecutar cualquiera de los siguientes comandos:

EJEMPLO

```
QUIT
EXIT
\q
```

Hasta ahora hemos estado explicando cómo trabajar con la consola desde un sistema Windows. En las próximas líneas vamos a explicar cómo acceder a la consola en sistemas Linux y Mac OS.

Para acceder a la consola MySQL, por ejemplo, desde una distribución Ubuntu de Linux, tendremos que ejecutar los siguientes comandos:

EJEMPLO

```
cd /opt/lampp/bin
mysql –h localhost –u root -p
```

En este momento, se nos solicitará igualmente la contraseña de acceso.

Para acceder a la consola MySQL, por ejemplo, desde Mac OS, tendremos que ejecutar los siguientes comandos:

EJEMPLO

```
/Applications/XAMPP/
$ /Applications/XAMPP/xamppfiles/bin/mysql –h localhost –u
root -p
```

y en este momento, tendremos que introducir la contraseña de acceso.

2.1.3. Creación y manejo de bases de datos

En los siguientes capítulos vamos a conocer las principales operaciones que podremos realizar con nuestras bases de datos:

- Crear y seleccionar una base de datos en concreto para trabajar con ella.
- Crear tablas para una base de datos.
- Cargar datos de forma masiva en una tabla.
- Realizar consultas para recuperar información de una base de datos.

2.1.3.1. Creación y selección de una base de datos

En esta sección vamos a ver los comandos relacionados para conocer la forma en la que podemos crear bases de datos, y cómo podemos seleccionarlas para su uso posterior.

No obstante, en primer lugar, si queremos conocer cuáles son las bases de datos que tenemos instaladas en nuestro servidor MySQL, tendremos que ejecutar el siguiente comando:

EJEMPLO

```
SHOW DATABASES;
```

Y tendríamos que obtener un resultado como el siguiente:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| phpmyadmin         |
+--------------------+
3 rows in set (0.00 sec)

mysql>
```

Figura 2.8. BBDD existentes.

La base de datos "*information_squema*" es utilizada para mantener un registro del resto de las bases de datos en el servidor.

La base de datos "*mysql*" se encargará de mantener un registro de los usuarios que hemos creado para que tengan acceso al servidor, sus contraseñas y los privilegios que tienen cada uno de ellos.

La tercera base de datos "phpmyadmin", ha sido creada en proceso de instalación de la aplicación "XAMPP" y su utilización la estudiaremos al final de esta unidad, pero básicamente nos va a ofrecer la posibilidad de realizar todas las operaciones que vamos a explicar con comandos, de forma gráfica a través de un navegador web.

NOTA

Todas las instrucciones tienen que terminar con punto y coma ";".
No importa si las escribimos en mayúsculas o minúsculas.
En el caso de nombres, si que habrá diferencias entre mayúsculas y minúsculas, si estamos trabajando desde un sistemas UNIX.

Con la excusa de no alterar el contenido de estas tres bases de datos, vamos a aprender a crear una nueva, cuyo nombre será "EMPRESA". Para ello tendremos que emplear la instrucción "CREATE DATABASE", cuya sintaxis es la siguiente:

EJEMPLO

```
CREATE DATABASE EMPRESA;
```

Si volvemos a ejecutar la instrucción "*SHOW DATABASES*", podremos observar que se ha creado la nueva base de datos.

Para poder realizar todas las operaciones sobre una base de datos, primero tendremos que ejecutar una instrucción que le indique a MySQL la base de datos sobre la que queremos trabajar. Para ello, tendremos que ejecutar el siguiente comando:

EJEMPLO

```
USE EMPRESA;
```

La tarea contraria a la creación de una base de datos, es su eliminación, que se realiza mediante la instrucción "DROP DATABASE".

Una técnica de programación frecuentemente utilizada por desarrolladores suele ser que antes de ejecutar la instrucción de creación de la base de datos, se ejecuta una instrucción de eliminación de la misma, por si hubiera alguna copia con el mismo nombre.

De esta forma, habrá que escribir lo siguiente:

EJEMPLO

```
DROP EMPRESA;
CREATE DATABAE EMPRESA;
```

2.1.3.2. Creación de una tabla

La parte más complicada en el diseño de una base de datos, será decidir qué tablas queremos que formen parte de ella, y qué campos queremos que formen parte de cada tabla.

Una vez que ya tenemos creada nuestra primera base de datos, el siguiente paso será la creación de las tablas.

Para hacer ilustrar nuestro ejemplo, vamos a crear una primera tabla para nuestra base de datos EMPRESA, denominada ARTICULOS, y que va a contener los siguientes campos:

- ID_ART. Campo de tipo entero y clave primaria.
- ARTICULO. Campo de tipo carácter
- COD_FABRICANTE. Campo de tipo numérico.
- PESO. Campo de tipo numérico.
- CATEGORIA. Campo de tipo carácter.
- PRECIO_VENTA. Campo de tipo numérico con dos decimales.
- PRECIO_COSTO. Campo de tipo numérico con dos decimales.
- EXISTENCIAS. Campo de tipo numérico.
- FECHA_COMPRA. Campo de tipo fecha.

De momento nos vamos a olvidar de los tipos de los datos (numérico, carácter…), ya que los explicaremos más adelante. Ahora nos centramos en la forma en cómo se puede crear una tabla.

Para ello, utilizaremos el comando "CREATE TABLE" acompañado de todos los campos que van a formar parte de la misma, identificados por su nombre y por el tipo de dato que lo forma:

EJEMPLO

```
CREATE TABLE 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 ) ;
```

NOTA

Volvemos a recordar que al final de toda sentencia SQL tiene que haber un punto y coma ";".

MySQL va a aceptar comandos distribuidos en varias líneas, así que no tendremos que preocuparnos de si la anterior sentencia es demasiado extensa. Nosotros hemos introducido el ejemplo en varias líneas para mejorar su comprensión, pero podría también haberse escrito en una sola línea.

Si extraemos el contenido donde estamos creando el primer campo "*ID_ART INT NOT NULL AUTO_INCREMENT PRIMARY KEY*", podemos extraer tres conclusiones:

- Cuando creemos una nuevo registro en esta tabla, este campo no estará permitido que se quede en blanco (NOT NULL).
- Le estamos indicando que por cada registro que introduzcamos en esta tabla, el valor de este campo se verá incrementado automáticamente en 1 (AUTO_INCREMENT). Esto es muy útil para generar campos claves cuyos valores nos aseguramos que no estarán repetidos.
- Finalmente, estamos indicando que este campo va a ser el identificador único para las entradas en esta tabla (PRIMARY KEY).

Los tipos de fecha deben de ser introducidos con el formato "AAAA-MM-DD" (DATE).

A continuación, si quisiéramos comprobar que efectivamente se ha creado la tabla, podríamos utilizar el comando "SHOW TABLES". Este comando puede que no resulte muy útil si solo tenemos una tabla, pero cuando tengamos una base de datos con un número elevado de tablas, sí que será muy útil.

Si quisiéramos comprobar que la tabla se ha creado con la estructura que hemos diseñado, podemos utilizar el comando "DESCRIBE ARTICULOS", y tendríamos que ver algo como lo siguiente:

```
mysql> DESCRIBE ARTICULOS
    -> ;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| ID_ART         | int(11)      | NO   | PRI | NULL    | auto_increment |
| ARTICULO       | varchar(20)  | NO   |     | NULL    |                |
| COD_FABRICANTE | int(3)       | NO   |     | NULL    |                |
| PESO           | int(3)       | NO   |     | NULL    |                |
| CATEGORIA      | varchar(10)  | NO   |     | NULL    |                |
| PRECIO_VENTA   | decimal(6,2) | YES  |     | NULL    |                |
| PRECIO_COSTO   | decimal(6,2) | YES  |     | NULL    |                |
| EXISTENCIAS    | int(5)       | YES  |     | NULL    |                |
| FECHA_COMPRA   | date         | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
9 rows in set (0.02 sec)
```

Figura 2.9. Tablas existentes y descripción de estructura.

Por último, si quisiéramos eliminar esta tabla, tendríamos que ejecutar el comando "*DROP TABLE ARTICULOS"*.

2.1.3.3. Carga masiva de datos en una tabla

Para añadir datos a una tabla, podemos hacer uso de la instrucción "*INSERT*", cuyo funcionamiento veremos más adelante. Pero rellenar todos los datos de una tabla consulta a consulta, puede ser una labor interminable.

Para facilitarnos esta tarea, podemos hacer uso de la instrucción "*LOAD DATA*", mediante la cual podremos cargar directamente todos los datos que necesitamos, volcándolos desde un fichero que hayamos creado previamente.

Este fichero se regirá con las siguientes normas:

- Cada línea de este fichero será un nuevo registro a incluir en la tabla.
- Por cada línea, tendremos que introducir los campos que forman parte de ellas separados por el tabulador.
- Será muy importante que pongamos los campos en el mismo orden que los utilizamos para crear la tabla.
- Si algún campo lo queremos dejar vacío, podemos utilizar el valor NULL que se especificará mediante los caracteres "\N".

Utilizando la tabla que hemos creado anteriormente, veamos un ejemplo de un fichero, al cual denominaremos "artículos.txt":

EJEMPLO

1 MACARRONES	100 2	PRIMERA	10	5	400	2011-03-04
2 MACARRONES	101 3	\N	8	4	500	2010-08-15
3 JUDIAS	100 4	PRIMERA	3	2	600	2010-12-12
4 LENTEJAS	102 5	PRIMERA	2	1	200	2010-11-11
5 GARBANZOS	103 3	SEGUNDA	6	4	300	2011-01-01

Con este fichero ya preparado, el siguiente paso será ejecutar la instrucción "*LOAD DATA*" de la siguiente forma:

EJEMPLO

```
LOAD DATA LOCAL INFILE "D:\ARTICULOS.TXT" INTO TABLE ARTICU-
LOS;
```

La palabra clave "LOCAL" va a indicar al servidor que el archivo se encuentra incluido en el equipo cliente (el que estamos utilizando para realizar la conexión). Si lo hubiéramos omitido, buscaría el archivo en el servidor de la base de datos.

2.1.3.4. Recuperar información de una tabla

Para consultar la información que está almacenada en tablas. se utiliza la sentencia "*SELECT*". Su formato sería el siguiente:

SELECT campos

FROM tabla

WHERE condiciones;

La cláusula "WHERE" es opcional, pero si está presente, la utilizaremos para especificar las condiciones que queremos que cumplan los registros que estamos consultando.

A continuación, vamos a ver las diferentes formas de utilizar la sentencia "SELECT".

Para seleccionar todos los datos

Ésta será la forma más simple de extraer la información. La utilizaremos para extraer toda ella de una tabla, para comprobar que se ha introducido de forma correcta, o simplemente queremos consultarla toda.

Para representar que queremos extraer la información de todos los campos, utilizamos el carácter **"*"**, que representará a todos los campos de una tabla, y los mostrará en el mismo orden en el que fueron creados.

La consulta se realizaría de la siguiente forma:

EJEMPLO

```
SELECT * FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT * FROM ARTICULOS;
+--------+-----------+-----------------+------+-----------+--------------+---------------+-------------+--------------+
| ID_ART | ARTICULO  | COD_FABRICANTE  | PESO | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO  | EXISTENCIAS | FECHA_COMPRA |
+--------+-----------+-----------------+------+-----------+--------------+---------------+-------------+--------------+
|      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   |
+--------+-----------+-----------------+------+-----------+--------------+---------------+-------------+--------------+
5 rows in set (0.00 sec)
```

Figura 2.10. Extraer todos los registros.

Para seleccionar campos concretos de una tabla

Esta operación sería una variante de la anterior, pero en este caso, solo seleccionaremos los campos que acompañen al SELECT:

EJEMPLO

```
SELECT ARTICULO, FECHA_COMPRA FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT ARTICULO,FECHA_COMPRA FROM ARTICULOS;
+------------+---------------+
| ARTICULO   | FECHA_COMPRA  |
+------------+---------------+
| MACARRONES | 2011-03-04    |
| MACARRONES | 2010-08-15    |
| JUDIAS     | 2010-12-12    |
| LENTEJAS   | 2010-11-11    |
| GARBANZOS  | 2011-01-01    |
+------------+---------------+
5 rows in set (0.00 sec)
```

Figura 2.11. Extraer algunos registros.

Para seleccionar registros mediante una condición

En numerosas ocasiones no vamos a necesitar extraer todos los registros de una tabla. Solo necesitaremos aquellos que cumplan determinadas condiciones.

Para expresar las condiciones, utilizaremos la cláusula "WHERE" que podrá ir acompañada de operadores como "AND" y "OR", por ejemplo.

Un dato a tener en cuenta es que si utilizamos un nombre como condición, se diferenciarán mayúsculas y minúsculas. Más adelante veremos funciones que nos ayudarán a resolver estos problemas.

Por ejemplo, si queremos consultar aquellos registros cuyo artículo sea "MACARRONES" o su categoría sea igual a "PRIMERA", tendríamos que ejecutar el siguiente comando:

EJEMPLO

```
SELECT * FROM ARTICULOS WHERE ARTICULO=”MACARRONES” OR
CATEGORIA=”PRIMERA”;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR CATEGORIA="PRIMERA";
+--------+-----------+-----------------+------+-----------+--------------+--------------+-------------+--------------+
| ID_ART | ARTICULO  | COD_FABRICANTE  | PESO | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO | EXISTENCIAS | FECHA_COMPRA |
+--------+-----------+-----------------+------+-----------+--------------+--------------+-------------+--------------+
|      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   |
+--------+-----------+-----------------+------+-----------+--------------+--------------+-------------+--------------+
4 rows in set (0.00 sec)
```

Figura 2.12. Extraer registros con condición.

Para ordenar los resultados de una consulta

Hasta este momento, los resultados de nuestras consultas no están siguiendo ningún orden especial, pero en determinados casos, nos va a interesar ordenar la salida por un campo en concreto.

Para ello tendremos que utilizar la cláusula “*ORDER BY*”, la cual puede ir acompañada de la opción “*DESC*” si queremos que el resultado sea ordenador de mayor a menor, ya que por defecto es de menor a mayor.

Por ejemplo, si queremos consultar aquellos registros cuyo artículo sea “MACARRONES” o su categoría sea igual a “PRIMERA”, y el resultado necesitamos que esté ordenador por el campo fecha de forma descendente, tendríamos que ejecutar el siguiente comando:

EJEMPLO

```
SELECT * FROM ARTICULOS WHERE ARTICULO=”MACARRONES” OR
CATEGORIA=”PRIMERA” ORDER BY FECHA_COMPRA DESC;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR CATEGORIA="PRIMERA" ORDER BY FECHA_COMPRA DESC;
+--------+-----------+-----------------+------+-----------+--------------+--------------+-------------+--------------+
| ID_ART | ARTICULO  | COD_FABRICANTE  | PESO | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO | EXISTENCIAS | FECHA_COMPRA |
+--------+-----------+-----------------+------+-----------+--------------+--------------+-------------+--------------+
|      1 | MACARRONES |             100 |    2 | PRIMERA   |        10.00 |         5.00 |         400 | 2011-03-04   |
|      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   |
|      2 | MACARRONES |             101 |    3 |           |         8.00 |         4.00 |         500 | 2010-08-15   |
+--------+-----------+-----------------+------+-----------+--------------+--------------+-------------+--------------+
4 rows in set (0.05 sec)
```

Figura 2.13. Ordenar los registros de una consulta.

Para limitar los resultados de una consulta

Imaginemos que tenemos una tabla con numerosos registros, y queremos limitar el resultado de nuestra consulta solo a unos cuantos de ellos.

Podremos conseguir este objetivo con el uso de la cláusula "*LIMIT*".

Por ejemplo, si queremos consultar aquellos registros cuyo artículo sea "MACARRONES" o su categoría sea igual a "PRIMERA", y el resultado necesitamos que esté ordenador por el campo fecha de forma descendente, pero que solo nos muestre los dos primeros resultados, tendríamos que ejecutar el siguiente comando:

EJEMPLO

```
SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR
CATEGORIA="PRIMERA" ORDER BY FECHA_COMPRA DESC LIMIT 2;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR CATEGORIA="PRIMERA" ORDER BY FECHA_COMPRA DESC LIMIT 2;
+--------+------------+----------------+------+-----------+--------------+--------------+-------------+--------------+
| ID_ART | ARTICULO   | COD_FABRICANTE | PESO | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO | EXISTENCIAS | FECHA_COMPRA |
+--------+------------+----------------+------+-----------+--------------+--------------+-------------+--------------+
|      1 | MACARRONES |            100 |    2 | PRIMERA   |        10.00 |         5.00 |         400 | 2011-03-04   |
|      3 | JUDIAS     |            100 |    4 | PRIMERA   |         3.00 |         2.00 |         600 | 2010-12-12   |
+--------+------------+----------------+------+-----------+--------------+--------------+-------------+--------------+
2 rows in set (0.00 sec)
```

Figura 2.14. Limitar los registros de una consulta.

Esta cláusula también nos puede servir para introducir un desplazamiento en la consulta, es decir, desde que resultado queremos comenzar a realizar la operación de limitación. Mediante la inclusión de dos números tras la cláusula ***"LIMIT"***, el primero sería para el desplazamiento, y el segundo sería el límite de fila.

Realicemos el mismo ejemplo que el anterior, pero queremos que nos muestre a partir del segundo resultado, los dos siguientes:

EJEMPLO

```
SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR
CATEGORIA="PRIMERA" ORDER BY FECHA_COMPRA DESC LIMIT 2,2;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR CATEGORIA="PRIMERA" ORDER BY FECHA_COMPRA DESC LIMIT 2,2;
+--------+-----------+----------------+------+-----------+--------------+--------------+------------+--------------+
| ID_ART | ARTICULO  | COD_FABRICANTE | PESO | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO | EXISTENCIAS | FECHA_COMPRA |
+--------+-----------+----------------+------+-----------+--------------+--------------+------------+--------------+
|      4 | LENTEJAS  |            102 |    5 | PRIMERA   |         2.00 |         1.00 |         200 | 2010-11-11   |
|      2 | MACARRONES |           101 |    3 |           |         8.00 |         4.00 |         500 | 2010-08-15   |
+--------+-----------+----------------+------+-----------+--------------+--------------+------------+--------------+
2 rows in set (0.00 sec)
```

Figura 2.15. Limitar con desplazamiento los registros de una consulta.

Para recuperar resultados distintos en una consulta

En algún momento, puede que deseemos que no se obtengan resultados repetidos en la ejecución de una consulta.

Podremos conseguir este objetivo con el uso de la cláusula "*DISTINCT*".

Por ejemplo, si queremos consultar los diferentes artículos que tenemos en nuestra tabla, podemos observar rápidamente que en dos registros existe el artículo "MACARRONES". Puede que nos interese mostrar los artículos diferentes que tenemos en nuestra tabla. Para ello tendríamos que escribir la siguiente sentencia.

EJEMPLO

```
SELECT DISTINCT ARTICULO FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT DISTINCT ARTICULO FROM ARTICULOS;
+------------+
| ARTICULO   |
+------------+
| MACARRONES |
| JUDIAS     |
| LENTEJAS   |
| GARBANZOS  |
+------------+
4 rows in set (0.06 sec)
```

Figura 2.16. Mostrar resultados distintos de los registros de una consulta.

2.2. Tipos de datos

Anteriormente ya hemos definido nuestra primera tabla, con una serie de campos, los cuales albergaban tipos de datos diferentes.

Ahora ha llegado el momento de conocer más a fondo los diferentes tipos de datos con los que podemos trababar en MySQL.

Principalmente existen tres tipos diferentes de datos: cadenas, números y fechas. Aunque también explicaremos los tipos de datos imágenes, los anteriores serán los utilizados habitualmente.

Por norma general, la mejor práctica sería utilizar el tipo de campo de menor tamaño, para conseguir ahorrar espacios y una mayor velocidad de acceso y actualización de la información. No obstante, tendremos que prestar atención a no escoger un tipo de dato tan pequeño que con ello estemos perdiendo información. Por lo tanto, la elección del tipo correcto de dato, se convierte en una decisión fundamental que pueda afectar al uso posterior de la base de datos.

2.2.1. Cadenas de caracteres

Los campos de cadenas de caracteres están diseñados para almacenar todo tipo compuesto por caracteres. La primera clasificación que podemos hacer con este tipo de datos sería la siguiente:

- [NATIONAL] CHAR (M) [BINARY]. Carácter. Una cadena de longitud fija, con relleno de espacios a la derecha para la longitud especificada. De 0 a 255 caracteres (de 1 a 255 en versiones de MySQL anteriores a la 3.23). Los espacios en blanco se eliminan al recuperar el valor.

- [NATIONAL] VARCHAR (M) [BINARY]. Carácter de longitud variable. Una cadena de longitud variable, cuyos espacios en blanco se eliminan al almacenar el valor. De 0 a 255 caracteres (de 1 a 255 en versiones de MySQL anteriores a la 4.0.2).

- TINYBLOB. Objeto binario pequeño. El número de caracteres máximo es de 255 (28 - 1). Requiere una longitud de almacenamiento de + 1 bytes. Igual que TINYTEXT, con la salvedad de que la búsqueda discrimina entre mayúsculas y minúsculas. Es aconsejable utilizar VARCHAR BINARY en la mayor parte de las situaciones porque resulta más rápido.

- TINYTEXT. El número de caracteres máximo es de 255 (28 - 1). Requiere una longitud de almacenamiento de + 1 bytes. Igual que TINYBLOB con la salvedad de que la búsqueda no discrimina entre mayúsculas y minúsculas. Es aconsejable utilizar VARCHAR en la mayor parte de las situaciones porque resulta más rápido.

- BLOB. Objeto binario grande. Máximo de 65535 caracteres (216 - 1). Requiere una longitud de almacenamiento de + 2 bytes. Igual que TEXT, con la salvedad de que la búsqueda discrimina entre mayúsculas y minúsculas.

- TEXT. Máximo de 65535 caracteres (216 - 1). Requiere una longitud de almacenamiento de + 2 bytes. Igual que BLOB, con la salvedad de que la búsqueda no discrimina entre mayúsculas y minúsculas.

- MEDIUMBLOB. Objeto binario grande de tamaño medio. Máximo de 16777215 caracteres (224 - 1). Requiere una longitud de almacenamiento de + 3 bytes. Igual que MEDIUMTEXT con la salvedad de que la búsqueda discrimina entre mayúsculas y minúsculas.

- MEDIUMTEXT. Máximo de 16777215 caracteres (224- 1). Requiere una longitud de almacenamiento de + 3 bytes. Igual que MEDIUMBLOB, con la salvedad de que la búsqueda no discrimina entre mayúsculas y minúsculas.

- LONGBLOB. Objeto binario grande de gran tamaño. Máximo de 4294967295 caracteres (232 - 1). Requiere una longitud de almacenamiento de + 4 bytes. Igual que LONGTEXT, con la salvedad de que la búsqueda discrimina entre mayúsculas y minúsculas.

- LONGTEXT. Máximo de 4294967295 caracteres (232 - 1). Requiere una longitud de almacenamiento de + 4 bytes. Igual que LONGBLOB, con la salvedad de que la búsqueda no discrimina entre mayúsculas y minúsculas.

- ENUM('valor1','valor2' ...). Enumeración. Sólo puede tener uno de los valores especificados, NULL o "". Valores máximos de 65535.

- SET('valor1','valor2' ...). Un conjunto. Puede contener de 0 a 64 valores de la lista especificada.

A continuación, vamos a detallar una serie de consideraciones para utilizar un tipo de dato u otro:

- No utilizar números para cadenas.

- Para lograr mayor velocidad, utilizar columnas fijas, como CHAR.

- Para ahorrar espacio, columnas dinámicas, como VARCHAR.

- Para delimitar los opciones de un campo a unas cuantas opciones, utilizar ENUM.

- Para permitir más de un valor para un campo, utilizar SET.

- Si queremos buscar texto sin discriminar entre mayúsculas y minúsculas, utilizar TEXT.

- Si queremos discriminarlas, utilizar BLOB.

Para que comprendamos un poco mejor las diferencias entre los principales tipo de datos de cadenas (CHAR y VARCHAR), en el siguiente ejemplo comparativo, podremos observar sus diferencias:

Valor	Char(4)	Almacenamiento necesario	Varchar (4)	Almacenamiento necesario
''	' '	4 bytes	''	1 byte
'ab'	'ab'	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Figura 2.17. Ejemplo de tabla comparativa de tipos CHAR y VARCHAR.

2.2.2. Datos numéricos

Los campos numéricos están diseñados para almacenar todo tipo de datos numéricos. La primera clasificación que podemos hacer con este tipo de datos sería la siguiente:

- Tipos enteros. Números enteros sin decimales ni partes fraccionales.
- Tipo de coma flotante.

Además, los tipos numéricos admitirán dos opciones:

- Unsigned. No va a permitir el uso de números negativos, extendiendo el rango positivo del tipo de los tipos enteros.
- Zerofill. Rellenará el valor con ceros en lugar de los espacios habituales, además de asignar el tipo Unsigned de manera predeterminada.

Una vez explicados estos conceptos previos fundamentales, vamos a describir uno por uno todos los tipos numéricos que utiliza MySQL:

- BIT[(M)]. Tipo de datos bit. *M* va a indicar el número de bits por valor de 1 a 64. Su valor por defecto es 1 si se omite *M*. Requiere 1 byte de almacenamiento.
- TINYINT[(M)] [UNSIGNED] [ZEROFILL]. Un entero muy pequeño. El rango con signo es de -128 a 127. El rango sin signo es de 0 a 255. Requiere 1 byte de almacenamiento.
- BOOL, BOOLEAN. Son sinónimos para TINYINT(1). Un valor de cero se considera falso. Valores distintos a cero se consideran ciertos.
- SMALLINT[(M)] [UNSIGNED] [ZEROFILL]. Un entero pequeño. El rango con signo es de -32768 a 32767. El rango sin signo es de 0 a 65535. Requiere 2 bytes de almacenamiento.
- MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]. Entero de tamaño medio. El rango con signo es de -8388608 a 8388607. El rango sin singo es de 0 a 16777215. Requiere 3 bytes de almacenamiento.
- INT[(M)] [UNSIGNED] [ZEROFILL]. Un entero de tamaño normal. El rango con signo es de -2147483648 a 2147483647. El rango sin signo es de 0 a 4294967295. Requiere 4 bytes de almacenamiento.
- INTEGER[(M)] [UNSIGNED] [ZEROFILL]. Es un sinónimo de INT.

- BIGINT[(M)] [UNSIGNED] [ZEROFILL]. Un entero grande. El rango con signo es de -9223372036854775808 a 9223372036854775807. El rango sin signo es de 0 a 18446744073709551615. Requiere 8 bytes de almacenamiento.

- FLOAT(p) [UNSIGNED] [ZEROFILL]. Número con coma flotante. p representa la precisión. Puede ir de 0 a 24 para números de coma flotante de precisión sencilla y de 25 a 53 para números de coma flotante con doble precisión. Estos tipos son como los tipos FLOAT y DOUBLE descritos a continuación. FLOAT(p) tiene el mismo rango que los tipos correspondientes FLOAT y DOUBLE, pero la anchura de muestra y el número de decimales no están definidos. Requiere 4 bytes de almacenamiento.

- FLOAT[(M,D)] [UNSIGNED] [ZEROFILL].Un número de coma flotante pequeño (de precisión simple). Los valores permitidos son de -3.402823466E+38 a -1.175494351E-38, 0, y de 1.175494351E-38 a 3.402823466E+38. Si se especifica UNSIGNED, los valores negativos no se permiten. M es la anchura de muestra y D es el número de dígitos significativos. FLOAT sin argumentos o FLOAT(p) (donde p está en el rango de 0 a 24) es un número de coma flotante con precisión simple. Requiere 4 bytes de almacenamiento.

- DOUBLE[(M,B)] [UNSIGNED] [ZEROFILL]. Número de coma flotante de tamaño normal (precisión doble). Los valores permitidos son de -1.7976931348623157E+308 a -2.2250738585072014E-308, 0, y de 2.2250738585072014E-308 a 1.7976931348623157E+308. Si se especifica UNSIGNED, no se permiten valores negativos. M es la anchura de muestra y B es el número de bits de precisión. DOUBLE sin parámetros o FLOAT(p) (donde p está en el rango de 25 a 53) es un número de coma flotante con doble precisión. Un número de coma flotante con precisión sencilla tiene una precisión de 7 decimales aproximadamente; un número con coma flotante de doble precisión tiene una precisión aproximada de 15 decimales. Requiere 8 bytes de almacenamiento.

- DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL], REAL[(M,D)] [UNSIGNED] [ZEROFILL]. Son sinónimos de DOUBLE. Excepción: si el modo del servidor SQL incluye la opción REAL_AS_FLOAT, REAL es un sinónimo para FLOAT en lugar de DOUBLE.

- DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]. Un numero decimal almacenado como una cadena, con un byte de espacio para cada carácter. Oscila entre -1,7976931348623 l57E+308 y -2,225073858507201 4E-308, O y 2.2250738585072014E-308 y 1.7976931 3486231 57E+308. M se utiliza para indicar el número total de dígitos (excluyendo el signo y el punto decimal, salvo en las versiones anteriores a la 3.23). D indica el número de decimales. Debe ser siempre inferior al valor de M. El valor predeterminado de D es 0 si se omite. A diferencia de los tipos numéricos, M y D pueden limitar el rango de valores permitidos. Con UNSIGNED, los valores negativos no se permiten.

- DEC[(M[,D])] [UNSIGNED] [ZEROFILL], NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL], FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]. Son sinónimos para DECIMAL. El sinónimo FIXED está disponible por compatibilidad con otros servidores.

Para escoger el tipo numérico, tendríamos que tener en cuenta las siguientes consideraciones:

- Seleccionar el tipo más pequeño susceptible de aplicación. Por ejemplo, elegir TINYING en lugar en INT si su valor no va a superar a 127.
- Para números enteros, seleccionar el tipo entero.
- Cuando necesitemos mayor precisión en nuestros datos, mejor utilizar los tipos enteros en lugar de los tipos de coma flotante, ya que los errores de redondeo afectan a los números coma flotante.

2.2.3. Fecha y hora

Los tipos de fecha y hora han sido diseñados para trabajar con las necesidades especiales que tienen los datos de tipo temporal, pero su mayor uso será para almacenar horas del día o fechas.

Podremos escoger entre los siguientes tipos:

- DATETIME. YYYY-MM-DD HH:MM:SS desde 1000-01-01 00:00:00 a 9999-12-31 23:59:59.
- DATE. YYYY-MM-DD desde 1000-01-01 a 9999-12-31.
- TIMESTAMP. AAAAMMDDHHMMSS.
- TIME. HH:MM:SS.
- YEAR. YYYY.

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 está comprendido entre 0 y 12 y que el día está comprendido entre 0 y 31.

- DATE: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1000 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-día.
- DATETIME: combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1000 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-día 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	AñoMesDíaHoraMinutoSegundo aaaammddhhmmss
12	AñoMesDíaHoraMinutoSegundo aammddhhmmss
8	AñoMesDía aaaammdd
6	AñoMesDía aammdd
4	AñoMes aamm
2	Año aa

Figura 2.18. Tipos de datos TIMESTAMP.

- TIME: almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838 horas, 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 2 o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Por ejemplo, podemos definir valores DATETIME, DATE, y TIMESTAMP usando los siguientes formatos:

- Como cadena de caracteres en formato 'YYYY-MM-DD HH:MM:SS' o 'YY-MM-DD HH:MM:SS'. Por ejemplo, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45' y '98@12@31 11^30^45' son equivalentes.

- Como cadena de caracteres en formato 'YYYY-MM-DD' o 'YY-MM-DD'. Por ejemplo, '98-12-31', '98.12.31', '98/12/31' y '98@12@31' son equivalentes.

- Como cadena de caracteres sin delimitadores en formato 'YYYYMMDDHHMMSS' o 'YYMMDDHHMMSS', mientras que la cadena de caracteres tenga sentido como fecha. Por ejemplo, '19970523091528' y '970523091528' se interpretan como '1997-05-23 09:15:28', pero '971122129015' es ilegal (tiene una parte de minutos sin sentido) y se convierte en '0000-00-00 00:00:00'.

- Como cadena de caracteres sin delimitadores en formato 'YYYYMMDD' o 'YYMMDD', mientras que la cadena de caracteres tenga sentido como fecha. Por ejemplo, '19970523' y '970523' se interpretan como '1997-05-23', pero '971332' es ilegal (tiene una parte de mes y día sin sentido) y se convierte en '0000-00-00'.

- Como número en formato YYYYMMDDHHMMSS o YYMMDDHHMMSS, mientras que el número tenga sentido como fecha. Por ejemplo, 19830905132800 y 830905132800 se interpretan como '1983-09-05 13:28:00'.

- Como número en formato YYYYMMDD o YYMMDD, mientras que el número tenga sentido como fecha. Por ejemplo, 19830905 y 830905 se interpretan como '1983-09-05'.

2.2.4. Imágenes

Para poder almacenar imágenes u otro tipo de archivos, se puede conseguir de dos formas distintas:

- Almacenando imágenes directamente en la base de datos MySQL. En esta solución, tendríamos que almacenar las imágenes en campos de tipo BLOB.

  Supongamos que creamos una tabla con la siguiente estructura:

EJEMPLO

```
CREATE TABLE imagenes (
id INT NOT NULL DEFAULT 0 auto_increment,
imagen LONGBLOB,
tipo VARCHAR(255),
PRIMARY KEY (id)
);
```

  El campo “imagen” contendrá la imagen propiamente dicha (los datos binarios) y el campo “tipo” contendrá el tipo MIME que identifica el formato de la imagen (por ejemplo,o image/gif).

  Por supuesto, esta tabla puede ser alterada, modificada, etc. para cubrir nuestras necesidades específicas.

  El tipo de datos BLOB (BLOB, MEDIUMBLOB y LONGBLOB) es el usado por MySQL para almacenar datos binarios.

  Una vez que ya tenemos la tabla definida, solo nos restará realizar un código en PHP, para realizar operaciones como subida de imágenes, recuperación de imágenes, etc.

- Almacenando imágenes en el servidor de hosting. De esta forma, lo único que guardaríamos en la base datos sería la ruta relativa a la localización exacta de la imagen en el servidor que está ejecutando MySQL, de tal forma que dicha ruta pueda ser usada en cualquier instrucción de salida de datos por pantalla. La ventaja de este tipo de solución es que vamos a forzar mucho menos la carga de trabajo de la base de datos, ya que será el servidor web quien se encargue de enviar la imagen.

2.3. Operadores

Mediante la utilización de operadores podremos crear consultas más complejas.

Por un lado, encontraremos los operadores lógicos mediante los cuales podremos comparar valores y delimitar el conjunto del resultado final.

Por otro lado, con los operadores aritméticos podremos realizar operaciones matemáticas básicas que poder utilizar en las consultas.

2.3.1. Operadores lógicos

Las operaciones en las que intervienen los operadores lógicos se van a traducir en un resultado verdadero (true o 1), falso (false o 0) o nulo (NULL).

Para realizar estas operaciones utilizaremos los operadores AND, OR y NOT.

- Operador AND, &&. El resultado será verdadero o 1, solo en el caso de que ambas condiciones sean verdaderas o 1. Su sintaxis es CONDICION1 AND CONDICION 2 o CONDICION1 && CONDICION2.
- Operador OR, ||. El resultado será verdadero o 1, si cualquiera de las dos condiciones es verdadera o 1. Su sintaxis es CONDICION1 OR CONDICION 2 o CONDICION1 || CONDICION2.
- Operador NOT. El resultado será verdadero o 1, si CONDICION1 es falsa y será falso si CONDICION1 es verdadero.

Como en cualquier otro lenguaje de programación, las condiciones que están incluidas dentro de paréntesis más internos serán resueltas en primer lugar.

Veamos unos cuantos ejemplos:

Si queremos visualizar aquellos artículos cuyo nombre sea “MACARRONES” y el código de fabricante sea igual al 100, tendremos que escribir la siguiente consulta:

EJEMPLO

```
SELECT * FROM ARTICULOS WHERE ARTICULO=”MACARRONES” AND COD_
FABRICANTE=100;
```

Tendríamos que ver un resultado como el siguiente:

```
mysql> SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" AND COD_FABRICANTE=100;
+--------+-----------+----------------+------+-----------+--------------+--------------+------------+--------------+
| ID_ART | ARTICULO  | COD_FABRICANTE | PESO | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO | EXISTENCIAS | FECHA_COMPRA |
+--------+-----------+----------------+------+-----------+--------------+--------------+------------+--------------+
|      1 | MACARRONES |            100 |    2 | PRIMERA   |        10.00 |         5.00 |         400 | 2011-03-04   |
+--------+-----------+----------------+------+-----------+--------------+--------------+------------+--------------+
1 row in set (0.02 sec)
```

Figura 2.19. Operación AND.

Si queremos visualizar aquellos artículos cuyo nombre sea “MACARRONES” o que su código de fabricante sea igual al 100, tendremos que escribir la siguiente consulta:

EJEMPLO

```
SELECT * FROM ARTICULOS WHERE ARTICULO=”MACARRONES” OR COD_
FABRICANTE=100;
```

Podemos observar que el resultado varía:

```
mysql> SELECT * FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR COD_FABRICANTE=100;
+--------+-----------+-----------------+-------+-----------+--------------+--------------+-------------+--------------+
| ID_ART | ARTICULO  | COD_FABRICANTE  | PESO  | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO | EXISTENCIAS | FECHA_COMPRA |
+--------+-----------+-----------------+-------+-----------+--------------+--------------+-------------+--------------+
|      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   |
+--------+-----------+-----------------+-------+-----------+--------------+--------------+-------------+--------------+
3 rows in set (0.00 sec)
```

Figura 2.20. Operación OR.

Si queremos visualizar aquellos artículos que NO cumplan la condición de que su nombre sea “MACARRONES” o que su código de fabricante sea igual al 100, tendremos que escribir la siguiente consulta:

EJEMPLO

```
SELECT * FROM ARTICULOS WHERE NOT(ARTICULO=”MACARRONES” OR
COD_FABRICANTE=100);
```

Podemos observar que justamente nos ofrece el resultado contrario al anterior:

```
mysql> SELECT * FROM ARTICULOS WHERE NOT(ARTICULO="MACARRONES" OR COD_FABRICANTE=100);
+--------+-----------+-----------------+-------+-----------+--------------+--------------+-------------+--------------+
| ID_ART | ARTICULO  | COD_FABRICANTE  | PESO  | CATEGORIA | PRECIO_VENTA | PRECIO_COSTO | EXISTENCIAS | FECHA_COMPRA |
+--------+-----------+-----------------+-------+-----------+--------------+--------------+-------------+--------------+
|      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   |
+--------+-----------+-----------------+-------+-----------+--------------+--------------+-------------+--------------+
2 rows in set (0.07 sec)
```

Figura 2.21. Operación NOT.

2.3.2. Operadores aritméticos

Mediante los operadores aritméticos podremos realizar operaciones aritméticas elementales, pero muy útiles para nuestras consultas SQL.

Podremos utilizar los siguientes:

- Operador +. Devuelve la suma de los elementos. Su sintaxis sería a + b.
- Operador -. Devuelve la resta de los elementos. Su sintaxis sería a - b.

- Operador *. Devuelve el producto de los elementos. Su sintaxis sería a * b.
- Operador /. Devuelve el cociente de los elementos. Su sintaxis sería a / b.
- Operador %. Devuelve el resto de la división de los elementos. Su sintaxis sería a % b.

Realicemos unas cuantas operaciones básicas para ver su resultado:

EJEMPLO

```
SELECT 8+4,8-4,8*4,8/4,8%4;
```

Podemos observar que justamente nos ofrece el resultado contrario al anterior:

```
mysql> SELECT 8+4,8-4,8*4,8/4,8%4;
+-----+-----+-----+--------+------+
| 8+4 | 8-4 | 8*4 | 8/4    | 8%4  |
+-----+-----+-----+--------+------+
|  12 |   4 |  32 | 2.0000 |    0 |
+-----+-----+-----+--------+------+
1 row in set (0.00 sec)
```

Figura 2.22. Operadores aritméticos.

2.3.3. Operadores DE COMPARACIÓN

En muchas de las consultas, haremos uso también de los operadores aritméticos. Vamos a realizar un breve resumen de los que podemos utilizar en nuestras consultas:

- Operador =. Para comparar si a es igual a b. Su sintaxis será a=b.
- Operador <=>.Este operador realiza una comparación de igualdad como el operador =, pero devuelve 1 en lugar de NULL si ambos operandos son NULL, y 0 en lugar de NULL si un operando es NULL..
- Operador <>, !=. Para comparar si a es distinto a b. Su sintaxis será a<>b, a!=b.
- Operador <=. Para comparar si a es igual o menor que b. Su sintaxis será a<=b.
- Operador <. Para comparar si a es menor que b. Su sintaxis será a<b.
- Operador >=. Para comparar si a es mayor o igual que b. Su sintaxis será a>=b.
- Operador >. Para comparar si a es mayor que b. Su sintaxis será a>b.
- IS NULL. Para preguntar si tiene un valor nulo. Su sintaxis será a IS NULL.
- IS NOT NULL. Será verdad si no tiene un valor nulo. Su sintaxis será a IS NOT NULL.

- BETWEEN. Será verdad si a está entre los valores de b y c, ambos inclusive. Su sintaxis será a BETWEEN b and c.
- NOT BETWEEN. Será verdad si a no está entre los valores de b y c, ambos inclusive. Su sintaxis será a NOT BETWEEN b and c.
- LIKE. Será verdad si a equivale a b en una correspondencia de patrón SQL. Su sintaxis será a LIKE b.
- NOT LIKE. Será verdad si a no equivale a b en una correspondencia de patrón SQL. Su sintaxis será a NOT LIKE b.
- IN. Será verdad si a es igual a un elemento del conjunto. Su sintaxis será a IN (b1,b2,b3...).
- NOT IN. Será verdad si a no es igual a un elemento del conjunto. Su sintaxis será a NOT IN (b1,b2,b3...).

2.4. Funciones

En este capítulo vamos a explicar las funciones más relevantes que podemos utilizar para manipulación de cadenas de textos, realizar operaciones matemáticas, así como funciones relativas al manejo de fechas y horas.

La lista de funciones sería interminable, así que nos centraremos en las más utilizadas.

2.4.1. Funciones de cadenas de texto

Con MySQL tendremos a nuestra disposición un número amplio de funciones para manipular cadenas de texto. Vamos a comentar las más utilizadas:

Convertir mayúsculas/minúsculas

LOWER() o LCASE(), convierte una cadena a minúsculas.

UPPER() o UCASE(), convierte una cadena a mayúsculas.

EJEMPLO

```
SELECT LOWER(ARTICULO), UPPER(ARTICULO) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT LOWER(ARTICULO), UPPER(ARTICULO) FROM ARTICULOS;
+------------------+------------------+
| LOWER(ARTICULO)  | UPPER(ARTICULO)  |
+------------------+------------------+
| macarrones       | MACARRONES       |
| macarrones       | MACARRONES       |
| judias           | JUDIAS           |
| lentejas         | LENTEJAS         |
| garbanzos        | GARBANZOS        |
+------------------+------------------+
5 rows in set (0.00 sec)
```

Figura 2.23. Mayúsculas o minúsculas en los registros de una consulta.

Funciones de extracción de parte de la cadena

LEFT(cadena, longitud) extrae varios caracteres del comienzo de la cadena.

RIGHT(cadena, longitud) extrae varios caracteres del final de la cadena.

EJEMPLO

```
SELECT LEFT(ARTICULO,2), RIGHT(ARTICULO,2) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT LEFT(ARTICULO,2), RIGHT(ARTICULO,2) FROM ARTICULOS;
+------------------+-------------------+
| LEFT(ARTICULO,2) | RIGHT(ARTICULO,2) |
+------------------+-------------------+
| MA               | ES                |
| MA               | ES                |
| JU               | AS                |
| LE               | AS                |
| GA               | OS                |
+------------------+-------------------+
5 rows in set (0.05 sec)
```

Figura 2.24. Extraer caracteres de la izquierda o la derecha en los registros de una consulta.

MID(cadena, posición, longitud), SUBSTR(cadena, posición, longitud) o SUBSTRING(cadena, posición, longitud) extraen varios caracteres de cualquier posición de una cadena, tantos como se indique en "longitud".

EJEMPLO

```
SELECT SUBSTRING(ARTICULO,2,3) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT SUBSTRING(ARTICULO,2,3) FROM ARTICULOS;
+-------------------------+
| SUBSTRING(ARTICULO,2,3) |
+-------------------------+
| ACA                     |
| ACA                     |
| UDI                     |
| ENT                     |
| ARB                     |
+-------------------------+
5 rows in set (0.00 sec)
```

Figura 2.25. Extraer caracteres desde una posición en los registros de una consulta.

CONCAT() une (concatena) varias cadenas para formar una nueva.

CONCAT_WS() une (concatena) varias cadenas para formar una nueva, empleando un separador que se indique (*With Separator*).

EJEMPLO

```
SELECT CONCAT_WS(' de ',ARTICULO,CATEGORIA) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT CONCAT_WS(' de ',ARTICULO,CATEGORIA) FROM ARTICULOS;
+---------------------------------------+
| CONCAT_WS(' de ',ARTICULO,CATEGORIA) |
+---------------------------------------+
| MACARRONES de PRIMERA                 |
| MACARRONES de                         |
| JUDIAS de PRIMERA                     |
| LENTEJAS de PRIMERA                   |
| GARBANZOS de SEGUNDA                  |
+---------------------------------------+
5 rows in set (0.00 sec)
```

Figura 2.26. Concatenar cadenas con un separador en una consulta.

LTRIM() devuelve la cadena sin los espacios en blanco que pudiera contener al principio.

RTRIM() devuelve la cadena sin los espacios en blanco que pudiera contener al final.

TRIM() devuelve la cadena sin los espacios en blanco que pudiera contener al principio y al final.

EJEMPLO

```
SELECT TRIM('    Hola caracola    ');
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT TRIM('    Hola caracola    ');
+-------------------------------+
| TRIM('    Hola caracola    ') |
+-------------------------------+
| Hola caracola                 |
+-------------------------------+
1 row in set (0.00 sec)
```

Figura 2.27. Eliminación de espacios en blanco.

Otras funciones de modificación de la cadena

INSERT(cadena,posición,longitud,nuevaCadena) inserta una cadena en otra cadena.

REPLACE(cadena,de,a) devuelve la cadena, cambiando ciertas secuencias de caracteres por otras.

REPEAT(cadena,numero) devuelve la cadena repetida varias veces.

REVERSE(cadena) devuelve la cadena al revés.

SPACE(longitud) devuelve una cadena formada por varios espacios en blanco.

EJEMPLO

```
SELECT INSERT(ARTICULO,2,2,'hola'),REPLACE(ARTICULO,'a','*')
,SPACE(3), REPEAT(ARTICULO,2) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT INSERT(ARTICULO,2,2,'hola'),REPLACE(ARTICULO,'a','*'),SPACE(3),REPEAT(ARTICULO,2) FROM ARTICULOS;
+--------------------------+---------------------------+----------+----------------------+
| INSERT(ARTICULO,2,2,'hola') | REPLACE(ARTICULO,'a','*') | SPACE(3) | REPEAT(ARTICULO,2)   |
+--------------------------+---------------------------+----------+----------------------+
| MholaARRONES             | MACARRONES                |          | MACARRONESMACARRONES |
| MholaARRONES             | MACARRONES                |          | MACARRONESMACARRONES |
| JholaIAS                 | JUDIAS                    |          | JUDIASJUDIAS         |
| LholaTEJAS               | LENTEJAS                  |          | LENTEJASLENTEJAS     |
| GholaBANZOS              | GARBANZOS                 |          | GARBANZOSGARBANZOS   |
+--------------------------+---------------------------+----------+----------------------+
5 rows in set (0.03 sec)
```

Figura 2.28. Operaciones varias sobre cadenas.

Funciones de información sobre la cadena

CHAR_LENGTH() o CHARACTER_LENGTH() devuelven la longitud de la cadena en caracteres.

LENGTH() devuelve la longitud de la cadena en bytes.

BIT_LENGTH() devuelve la longitud de la cadena en bits.

INSTR(cadena,subcadena) o LOCATE(subcadena,cadena,posInicial) devuelve la posición de una subcadena dentro de la cadena.

EJEMPLO

```
SELECT ARTICULO, CHAR_LENGTH(ARTICULO),INSTR(ARTICULO,'AR')
FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT ARTICULO, CHAR_LENGTH(ARTICULO), INSTR(ARTICULO,'AR') FROM ARTICULOS;
+------------+----------------------+---------------------+
| ARTICULO   | CHAR_LENGTH(ARTICULO) | INSTR(ARTICULO,'AR') |
+------------+----------------------+---------------------+
| MACARRONES |                   10 |                   4 |
| MACARRONES |                   10 |                   4 |
| JUDIAS     |                    6 |                   0 |
| LENTEJAS   |                    8 |                   0 |
| GARBANZOS  |                    9 |                   2 |
+------------+----------------------+---------------------+
5 rows in set (0.00 sec)
```

Figura 2.29. Operaciones varias sobre cadenas.

2.4.2. Funciones matemáticas

Existen diferentes funciones que nos van a ayudar a realizar determinadas tareas matemáticas que de otra manera sería realmente costoso poder realizarlas. No hace falta aprenderse todas, nadie lo hace. Pero vamos a ver las más importantes:

Para obtener el valor máximo. Función MAX()

Mediante esta función, el resultado de la consulta nos devolverá el valor más alto de entre todos los que puedan ser seleccionados.

Por ejemplo, si queremos conocer cuánto pesa el artículo con más peso, tendríamos que realizar la siguiente consulta:

EJEMPLO

```
SELECT MAX(PESO) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT MAX(PESO) FROM ARTICULOS;
+-----------+
| MAX(PESO) |
+-----------+
|         5 |
+-----------+
1 row in set (0.00 sec)
```

Figura 2.30. Mostrar el mayor peso de nuestros artículos.

Para contar los registros de una tabla. Función COUNT(*)

Si queremos contar los registros que tenemos en una tabla, tendremos que utilizar la función “COUNT (*)”. Esta función puede sernos especialmente útil en situaciones donde simplemente necesitamos conocer el número total de registros en una tabla, o para utilizarlo como condición en consultas más avanzadas.

Por ejemplo, si quisiéramos contar todos los registros de una tabla, tendríamos que escribir el siguiente comando:

EJEMPLO

```
SELECT COUNT(*) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT COUNT(*) FROM ARTICULOS;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

Figura 2.31. Contar registros de una tabla.

También podemos utilizarlo en combinación de una cláusula "WHERE". Si utilizamos el ejemplo de antes, pero en vez de mostrar los campos, solo queremos que nos muestre el número total de registros cuyo artículo sea "MACARRONES" o su categoría sea igual a "PRIMERA", tendríamos que ejecutar el siguiente comando:

EJEMPLO

```
SELECT COUNT(*) FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR
CATEGORIA="PRIMERA";
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT COUNT(*) FROM ARTICULOS WHERE ARTICULO="MACARRONES" OR CATEGORIA="PRIMERA";
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Figura 2.32. Contar registros con condición.

Si en el ejemplo anterior nos mostraba todos los campos de los registros que cumplían las condiciones, en este caso, solo nos muestra el número total de registros que cumplen dichas condiciones.

Podríamos combinar la función *"COUNT(*)"* con la cláusula *"DISTINCT"*, para que la consulta nos contara el número de artículos distintos que tenemos en nuestra tabla, de la siguiente forma:

EJEMPLO

```
SELECT COUNT(DISTINCT ARTICULO) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT COUNT(DISTINCT ARTICULO) FROM ARTICULOS;
+--------------------------+
| COUNT(DISTINCT ARTICULO) |
+--------------------------+
|                        4 |
+--------------------------+
1 row in set (0.04 sec)
```

Figura 2.33. Contar registros distintos.

De esta forma, nos evitamos que nos cuente el artículo "MACARRONES" en dos ocasiones.

Para realizar las operaciones de MEDIA, MINIMO Y TOTAL. Funciones AVG(), MIN() y SUM()

Estas funciones las podremos utilizar de la misma forma que hemos utilizado la función MAX(). Es decir, dentro de los paréntesis tendremos que incluir aquellos campos sobre los que queramos realizar la operación.

Por ejemplo, si quisiéramos conocer el precio medio de venta de nuestros artículos, utilizando la función "*AVG()",* tendríamos que escribir el siguiente comando:

EJEMPLO

```
SELECT AVG(PRECIO_VENTA) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT AVG(PRECIO_VENTA) FROM ARTICULOS;
+--------------------+
| AVG(PRECIO_VENTA) |
+--------------------+
|           5.800000 |
+--------------------+
1 row in set (0.05 sec)
```

Figura 2.34. Realizar la media de un campo.

Si quisiéramos conocer el precio mínimo de venta de nuestros artículos, utilizando la función "*MIN()",* tendríamos que escribir el siguiente comando:

EJEMPLO

```
SELECT MIN(PRECIO_VENTA) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT MIN(PRECIO_VENTA) FROM ARTICULOS;
+--------------------+
| MIN(PRECIO_VENTA) |
+--------------------+
|               2.00 |
+--------------------+
1 row in set (0.04 sec)
```

Figura 2.35. Mostrar el valor mínimo de un campo de una tabla.

Si quisiéramos conocer la suma total de existencias que tenemos en el almacén, utilizando la función "*SUM()",* tendríamos que escribir el siguiente comando:

EJEMPLO

```
SELECT SUM(EXISTENCIAS) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT SUM(EXISTENCIAS) FROM ARTICULOS;
+-------------------+
| SUM(EXISTENCIAS) |
+-------------------+
|              2000 |
+-------------------+
1 row in set (0.00 sec)
```

Figura 2.36. Mostrar la suma total de un campo de una tabla.

Para realizar cálculos en una consulta

Podremos realizar cálculos directamente en nuestras consultas con la inclusión directa de una operación.

Por ejemplo, si quisiéramos conocer cuál sería el valor de existencias de todos nuestros artículos si compráramos 100 unidades más para todos, tendríamos que escribir el siguiente comando:

EJEMPLO

```
SELECT ARTICULO, EXISTENCIAS+100 FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> select articulo, existencias+100 from articulos;
+------------+-----------------+
| articulo   | existencias+100 |
+------------+-----------------+
| MACARRONES |             500 |
| MACARRONES |             600 |
| JUDIAS     |             700 |
| LENTEJAS   |             300 |
| GARBANZOS  |             400 |
+------------+-----------------+
5 rows in set (0.07 sec)
```

Figura 2.37. Realizar cálculos en las consultas.

2.4.3. Funciones de fecha y hora

Podremos determinar la fecha actual, obteniéndola de la que tenga configurado el servidor, mediante la función "*CURRENT_DATE()".*

Podremos determinar la hora actual, obteniéndola de la que tenga configurado el servidor, mediante la función "*CURRENT_TIME()".*

Para obtener la fecha y la hora mediante una sola función *"NOW()".*

Para utilizarlas, tendremos que hacer uso de la cláusula "SELECT". Veamos un ejemplo:

```
mysql> SELECT CURRENT_TIME(), CURRENT_DATE(),NOW();
+----------------+----------------+---------------------+
| CURRENT_TIME() | CURRENT_DATE() | NOW()               |
+----------------+----------------+---------------------+
| 12:51:18       | 2011-07-07     | 2011-07-07 12:51:18 |
+----------------+----------------+---------------------+
1 row in set (0.00 sec)
```

Figura 2.38. Fecha y hora actual.

2.4.4. Manejo de fechas en MySQL

En este capítulo vamos a explicar alguna de las operaciones de manejo de fechas que podemos realizar.

Dar formato a la fecha

MySQL permite devolver las fechas en un formato especial, en lugar de utilizar el formato estándar AAAA-MM-DD que hemos comentado cuando hemos definido nuestra tabla.

Para ello, utilizaremos la función *"DATE_FORMAT()".*

Por ejemplo, si queremos visualizar nuestro campo "FECHA_COMPRA" en un formato DD-MM-AAAA, y ordenado por el mismo campo, podríamos realizarlo de la siguiente forma:

EJEMPLO

```
SELECT  DATE_FORMAT(FECHA_COMPRA,"%d/%m/%Y")  FROM  ARTICULOS
ORDER BY FECHA_COMPRA;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT DATE_FORMAT(FECHA_COMPRA,"%d/%m/%Y") FROM ARTICULOS ORDER BY FECHA_COMPRA;
+-------------------------------------+
| DATE_FORMAT(FECHA_COMPRA,"%d/%m/%Y") |
+-------------------------------------+
| 15/08/2010                          |
| 11/11/2010                          |
| 12/12/2010                          |
| 01/01/2011                          |
| 04/03/2011                          |
+-------------------------------------+
5 rows in set (0.00 sec)
```

Figura 2.39. Visualización con formato de las fechas.

La cadena que utilizamos dentro de la función se denomina cadena de formato, y en ella habrá que utilizar los especificadores oportunos para dar el formato que necesitamos. El listado completo de especificadores para esta función sería el siguiente:

- %a Día de semana abreviado (Sun..Sat).
- %b Mes abreviado (Jan..Dec).
- %c Mes, numérico (0..12).
- %D Día del mes con sufijo inglés (0th, 1st, 2nd, 3rd, ...).
- %d Día del mes numérico (00..31).
- %e Día del mes numérico (0..31).
- %f Microsegundos (000000..999999).
- %H Hora (00..23).
- %h Hora (01..12).
- %I Hora (01..12).
- %i Minutos, numérico (00..59).
- %j Día del año (001..366).

- %k Hora (0..23).
- %l Hora (1..12).
- %M Nombre mes (January..December).
- %m Mes, numérico (00..12).
- %p AM o PM.
- %r Hora, 12 horas (hh:mm:ss seguido de AM o PM).
- %S Segundos (00..59).
- %s Segundos (00..59).
- %T Hora, 24 horas (hh:mm:ss).
- %U Semana (00..53), donde domingo es el primer día de la semana.
- %u Semana (00..53), donde lunes es el primer día de la semana.
- %V Semana (01..53), donde domingo es el primer día de la semana; usado con %X.
- %v Semana (01..53), donde lunes es el primer día de la semana; usado con %x.
- %W Nombre día semana (Sunday..Saturday).
- %w Día de la semana (0=Sunday..6=Saturday).
- %X Año para la semana, donde domingo es el primer día de la semana, numérico, cuatro dígitos; usado con %V.
- %x Año para la semana, donde lunes es el primer día de la semana, numérico, cuatro dígitos; usado con %v.
- %Y Año, numérico (cuatro dígitos).
- %y Año, numérico (dos dígitos).
- %% Carácter '%' literal.

Por ejemplo, si queremos visualizar nuestro campo "FECHA_COMPRA" de tal forma que nos devuelva el día de la semana (%W), el día con un formato de dos números (%e), con el nombre del mes (%M) y el año con un formato de dos números (%y), podríamos realizarlo de la siguiente forma:

EJEMPLO

```
SELECT DATE_FORMAT(FECHA_COMPRA,”%W %e %M %y”) FROM ARTICULOS
ORDER BY FECHA_COMPRA;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT DATE_FORMAT(FECHA_COMPRA,"%d/%m/%Y") FROM ARTICULOS ORDER BY FECHA_COMPRA;
+--------------------------------------+
| DATE_FORMAT(FECHA_COMPRA,"%d/%m/%Y") |
+--------------------------------------+
| 15/08/2010                           |
| 11/11/2010                           |
| 12/12/2010                           |
| 01/01/2011                           |
| 04/03/2011                           |
+--------------------------------------+
5 rows in set (0.00 sec)
```

Figura 2.40. Visualización con formato de las fechas.

Recuperar partes de una fecha

Podemos extraer información de fechas mediante las siguientes funciones:

- DAY(). Extrae el día de una fecha.
- DAYNAME(). Extrae el nombre del día de la fecha.
- MONTH(). Extrae el mes de una fecha.
- DAYOFMONTH(). Extrae el día del mes de la fecha.
- YEAR(). Extrae el año de una fecha.
- DAYOFYEAR(). Extrae el nombre del día de la fecha.
- DAY(). Extrae el día de una fecha.
- DAYOFWEEK(). Extrae el índice del día de la semana para una fecha (1 = domingo, 2 = lunes, ..., 7 = sábado).
- HOUR(). Extrae la hora de una fecha.

Veamos un ejemplo:

EJEMPLO

```
SELECT DAYNAME(FECHA_COMPRA), DAY(FECHA_COMPRA), MONTH(FECHA_
COMPRA), YEAR(FECHA_COMPRA) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT DAYNAME(FECHA_COMPRA), DAY(FECHA_COMPRA), MONTH(FECHA_COMPRA), YEAR(FECHA_COMPRA) FROM ARTICULOS;
+-----------------------+--------------------+---------------------+--------------------+
| DAYNAME(FECHA_COMPRA) | DAY(FECHA_COMPRA)  | MONTH(FECHA_COMPRA) | YEAR(FECHA_COMPRA) |
+-----------------------+--------------------+---------------------+--------------------+
| Friday                |                  4 |                   3 |               2011 |
| Sunday                |                 15 |                   8 |               2010 |
| Sunday                |                 12 |                  12 |               2010 |
| Thursday              |                 11 |                  11 |               2010 |
| Saturday              |                  1 |                   1 |               2011 |
+-----------------------+--------------------+---------------------+--------------------+
5 rows in set (0.00 sec)
```

Figura 2.41. Extracción de contenido de las fechas.

O si por ejemplo, quisiéramos calcular cuántos años han transcurrido desde la compra de cada uno de nuestros artículos, tendríamos que ejecutar la siguiente sentencia:

EJEMPLO

```
SELECT YEAR(NOW()) - YEAR(FECHA_COMPRA) FROM ARTICULOS;
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT YEAR(NOW()) - YEAR(FECHA_COMPRA) FROM ARTICULOS;
+-----------------------------------+
| YEAR(NOW()) - YEAR(FECHA_COMPRA)  |
+-----------------------------------+
|                                 0 |
|                                 1 |
|                                 1 |
|                                 1 |
|                                 0 |
+-----------------------------------+
5 rows in set (0.00 sec)
```

Figura 2.42. Cálculos con fechas.

Esta consulta puede también ser muy útil en una tabla donde tuviéramos fechas de nacimientos y quisiéramos calcular la edad actual de cada persona.

Consultar diferencia entre horas y fechas

Con la utilización de la función ***"TIMEDIFF()"*** podremos obtener el tiempo entre la hora de inicio de "hora1" y la hora final "hora2". "hora1" y "hora2" son expresiones de hora o de fecha/hora, pero ambas deben ser del mismo tipo.

Veamos un ejemplo, añadiendo una hora entre dos fechas iguales:

EJEMPLO

```
SELECT TIMEDIFF('2011:01:01 00:00:00','2011:01:01 01:00:00');
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT TIMEDIFF('2011:01:01 00:00:00','2011:01:01 01:00:00');
+----------------------------------------------------------+
| TIMEDIFF('2011:01:01 00:00:00','2011:01:01 01:00:00') |
+----------------------------------------------------------+
| -01:00:00                                                |
+----------------------------------------------------------+
1 row in set (0.05 sec)
```

Figura 2.43. Diferencia entre las fechas.

Sumas entre fechas

Para poder sumar una cantidad a una fecha, podemos utilizar la función "TIMESTAMP(expr)" o "TIMESTAMP(expr,expr2)".

Con un único argumento, esta función retorna la expresión de fecha o fecha/hora "expr" como valor fecha/hora.

Con dos argumentos, suma la expresión de hora "expr2" a la expresión de fecha o de fecha/hora "expr" y retorna el resultado como valor fecha/hora.

Veamos dos ejemplos, el primero con la función con un argumento, y el segundo con una suma de 12 horas a una fecha:

EJEMPLO

```
SELECT TIMESTAMP('2010-12-31');
SELECT TIMESTAMP('2010-12-31 12:00:00','12:00:00');
```

Tendríamos que obtener un resultado como el siguiente:

```
mysql> SELECT TIMESTAMP('2010-12-31');
+---------------------------+
| TIMESTAMP('2010-12-31') |
+---------------------------+
| 2010-12-31 00:00:00     |
+---------------------------+
1 row in set (0.00 sec)

mysql> SELECT TIMESTAMP('2010-12-31 12:00:00','12:00:00');
+---------------------------------------------+
| TIMESTAMP('2010-12-31 12:00:00','12:00:00') |
+---------------------------------------------+
| 2011-01-01 00:00:00                         |
+---------------------------------------------+
1 row in set (0.00 sec)
```

Figura 2.44. Sumas de fechas.

2.5. Usuarios y privilegios

A continuación, vamos a explicar los comandos MySQL que podemos utilizar para la administración básica de usuarios que tengan acceso a MySQL.

Mediante la concesión de estos permisos podremos permitir o prohibir que los usuarios puedan conectarse al servidor de bases de datos o que realicen determinadas operaciones en las bases de datos, tablas o incluso en columnas específicas de las tablas.

2.5.1. Nombres de usuario y contraseñas

Cuando creamos un nuevo usuario en MySQL, éste queda identificado por su nombre de usuario más el nombre o IP del ordenador desde el cual hemos dicho que accederá (podemos utilizar el carácter comodín '%' para representar varios ordenadores).

La sintaxis de los nombres de usuarios puede ser de varias formas en función de su localización:

- usuario
- usuario@'%'
- usuario@localhost
- usuario@'192.168.0.%'
- usuario@'%.midominio.org'

Por ejemplo, el usuario *'usuario@localhost'* se considera diferente del usuario *'usuario@192.168.0.%'*, aunque tengan el mismo nombre 'usuario', y por lo tanto, pueden tener permisos diferentes.

La segunda parte, que aparece separada de la primera por el carácter '@' es un nombre de máquina (host). Este nombre puede ser bien el de una máquina, por ejemplo, 'localhost' para referirse al ordenador local, o cualquier otro nombre, o bien una dirección IP.

Los nombres de usuario tendrán como máximo 16 caracteres de longitud.

MySQL cifra las contraseñas mediante un algoritmo propio. Es el mismo que el utilizado en la función PASSWORD().

2.5.2. Sistema de privilegios

Los privilegios que podemos asignar o revocar se pueden realizar a diferentes niveles:

- Nivel de base de datos. Estos permisos son aplicados para todos los objetos de una base de datos.
- Nivel de tabla. Los permisos serán aplicados a todas las columnas o campos de una tabla.

- Nivel de columna. Solo serán aplicados a esa columna en concreto.
- Nivel de rutina. Son aplicados a las rutinas que pudiera tener almacenadas.

Los privilegios que podrán utilizarse mediante los comandos GRANT y REVOKE son los siguientes:

- ALL [PRIVILEGES]. Da todos los permisos simples excepto GRANT OPTION.
- ALTER. Permite el uso de ALTER TABLE.
- ALTER ROUTINE. Modifica o borra rutinas almacenadas.
- CREATE. Permite el uso de CREATE TABLE.
- CREATE ROUTINE. Crea rutinas almacenadas.
- CREATE TEMPORARY TABLES. Permite el uso de CREATE TEMPORARY TABLE.
- CREATE USER. Permite el uso de CREATE USER, DROP USER, RENAME USER, y REVOKE ALL PRIVILEGES.
- CREATE VIEW. Permite el uso de CREATE VIEW.
- DELETE. Permite el uso de DELETE.
- DROP. Permite el uso de DROP TABLE.
- EXECUTE. Permite al usuario ejecutar rutinas almacenadas.
- FILE. Permite el uso de SELECT ... INTO OUTFILE y LOAD DATA INFILE.
- INDEX. Permite el uso de CREATE INDEX y DROP INDEX.
- INSERT. Permite el uso de INSERT.
- LOCK TABLES. Permite el uso de LOCK TABLES en tablas para las que tenga el permiso SELECT.
- PROCESS. Permite el uso de SHOW FULL PROCESSLIST.
- RELOAD. Permite el uso de FLUSH.
- REPLICATION CLIENT. Permite al usuario preguntar dónde están los servidores maestro o esclavo.
- REPLICATION SLAVE. Necesario para los esclavos de replicación (para leer eventos del log binario desde el maestro).
- SELECT. Permite el uso de SELECT.
- SHOW DATABASES. Muestra todas las bases de datos.

- SHOW VIEW. Permite el uso de SHOW CREATE VIEW.
- SHUTDOWN. Permite el uso de mysqladmin shutdown.
- SUPER. Permite el uso de comandos CHANGE MASTER, KILL, PURGE MASTER LOGS, y SET GLOBAL , el comando mysqladmin debug le permite conectar (una vez) incluso si se llega a max_connections.
- UPDATE. Permite el uso de UPDATE.
- USAGE. Sinónimo de "no privileges".
- GRANT OPTION. Permite dar permisos.

Podremos asignar permisos globales usando la sintaxis "ON *.*" o permisos a nivel de base de datos usando la sintaxis "ON db_name.*". Si utilizamos ON * y tenemos seleccionada una base de datos por defecto, los permisos se dan en esa base de datos. Si especificamos ON * y no hemos seleccionado una base de datos por defecto, los permisos dados son globales.

2.5.3. Añadir cuentas de usuario a MySQL

Para añadir cuentas de usuario a MySQL tenemos dos opciones disponibles desde la línea de comandos: las sentencias "*CREATE USER*" y "*GRANT*".

Creación de usuarios con CREATE USER

Desde la versión MySQL 5.0.2 tenemos la opción de crear usuarios a través del comando "CREATE USER". Más adelante le daremos los privilegios oportunos para delimitar lo que podrá hacer o no.

Para crear estas cuentas, el usuario que ejecute la sentencia deberá de tener permiso global "CREATE USER" o permiso "INSERT" para la base de datos MySQL.

Cada cuenta que creemos añadirá de forma automática un nuevo registro en la tabla mysql.user.

Su sintaxis sería la siguiente:

CREATE USER usuario [IDENTIFIED BY 'contraseña'] [, usuario [IDENTIFIED BY 'contraseña']] ...

Algunos ejemplos de creación de usuarios con este comando podrían ser los siguientes:

EJEMPLO

```
CREATE USER cristian IDENTIFIED BY 'mipassword';
CREATE USER anonimo@localhost;
CREATE USER alumno@'192.168.0.%' IDENTIFIED BY 'Alumno';
```

El contexto 'localhost' define que el usuario solamente se puede conectar desde el servidor de MySQL, y la cláusula IDENTIFIED BY define el password del usuario, se puede omitir, para un usuario sin password, siempre que el modo SQL no sea NO_AUTO_CREATE_USER.

Creación de usuarios con GRANT

Esta es la opción clásica de creación de usuario. Mediante esta instrucción no solo podremos crear un usuario, sino también asignarle privilegios sobre los diferentes objetos de una base de datos, o a la base de datos entera.

Cuando MySQL recibe una instrucción GRANT, en primer lugar tratará de comprobar si el usuario ya existe para otorgarle los permisos oportunos, de tal forma que si no está creado, lo hace en ese momento.

Su sintaxis es la siguiente:

GRANT privilege ON table-or-database-name TO user-name@hostname IDENTIFIED BY 'password'

Si quisiéramos que el usuario no tuviera un password, deberemos omitir la cláusula IDENTIFIED BY.

En el caso de que el modo SQL del servidor estuviera en NO_AUTO_CREATE_USER, la creación de usuarios no estaría permitida a no ser que tuvieran asignado un password no vacío.

Veamos un ejemplo:

EJEMPLO

```
GRANT SELECT, INSERT ON empresa.* TO 'cristian'@'localhost'
IDENTIFIED BY 'pass_cristian;
```

En este ejemplo damos privilegios al usuario cristian para que seleccione (SELECT) e inserte (INSERT) en todos los objetos (*) de la base de datos empresa, y además, indicamos que el contexto sea la máquina local de la base de datos (localhost), lo que impedirá que el usuario se conecte desde otras máquinas, y finalmente, asignamos un password mediante IDENTIFIED BY. Si el usuario no existe, es creado en este momento.

Para crear un usuario sin privilegios usaremos la siguiente sentencia:

EJEMPLO

```
GRANT USAGE ON *.* TO anonimo IDENTIFIED BY 'password';
```

Para conceder al usuario 'anonimo' el privilegio de ejecutar sentencias sobre la tabla 'articulos' de la base de datos 'empresa':

EJEMPLO

```
GRANT SELECT ON empresa.articulos TO anonimo;
```

Para conceder varios permisos en una sola sentencia, tendríamos que escribir lo siguiente:

EJEMPLO

```
GRANT SELECT, UPDATE ON empresa.articulos TO anónimo IDENTI-
FIED BY 'password';
```

2.5.4. Eliminar cuentas de usuario a MySQL

Para eliminar un usuario utilizaremos la sentencia "DROP USER".

Su sintaxis es la siguiente:

DROP USER usuario.

Por ejemplo, para eliminar al usuario "anonimo", usaremos la sentencia:

EJEMPLO

```
DROP USER anonimo;
```

2.5.5. Limitar cuentas de usuario

Para revocar privilegios que hayamos podido asignar previamente a los usuarios, podremos hacer uso de la sentencia "REVOKE".

Su sintaxis sería la siguiente:

REVOKE privilege ON table-or-database-name FROM user-name@hostname

Por ejemplo, para revocar el privilegio SELECT de nuestro usuario 'anonimo', usaremos la sentencia:

EJEMPLO

```
REVOKE SELECT ON empresa.articulos FROM anonimo;
```

2.5.6. Mostrar permisos de usuario

Para poder comprobar los permisos que puedan tener asignados un determinado usuario, haremos uso de la sentencia "SHOW GRANTS".

Su sintaxis es la siguiente:

SHOW GRANTS FOR usuario.

Por ejemplo, para conocer los permisos del usuario anónimo:

EJEMPLO

```
SHOW GRANTS FOR anonimo;
```

2.5.7. Crear contraseñas a una cuenta

Para cambiar contraseñas podemos utilizar la sentencia "SET PASSWORD".

Cualquier usuario que no sea anónimo podrá cambiar su propia contraseña con esta sentencia.

Por ejemplo, si estamos conectados con nuestro usuario, y queremos cambiar nuestra contraseña, utilizaremos la siguiente sentencia:

EJEMPLO

```
SET PASSWORD=PASSWORD('nuevapassword');
```

Por ejemplo, si tenemos permisos y queremos cambiar la contraseña de otro usuario, utilizaremos la siguiente sentencia:

EJEMPLO

```
SET PASSWORD FOR
cristian=PASSWORD('nuevapassword');
```

2.6. Motores de almacenamiento

MySQL puede trabajar con diferentes motores de almacenamiento que a su vez pueden trabajar con distintos tipos de tablas.

Podremos encontrar dos tipos de tablas de transacción:

- Transacción segura (InnoDB y BDB).
- No son de transacción segura (ISAM, MyISAM, MERGE y HEAP).

La elección de este tipo de tablas afectará al rendimiento.

A continuación, vamos a explicar los dos principales motores de almacenamiento: MyISAM e InnoDB.

2.6.1. Motor de almacenamiento MyISAM

MyISAM será el motor de almacenamiento por defecto.

Las tablas de tipo MyISAM sustituyeron a las tablas ISAM en la versión 3.23.0.

Los archivos de datos MyISAM llevan asignada la extensión .MYD y la extensión de los índices es .MYI. Las bases de datos MyISAM se almacenan en un directorio.

Arranque

Podremos utilizar *mysqld* al arrancar el servidor con unas opciones que van a determinar el comportamiento de las tablas de tipo MyISAM:

- --myisam-recover=mode. Cambia el modo para recuperación automática para tablas MyISAM.
- --delay-key-write=ALL. No vuelca buffers de clave entre escrituras para cualquier tabla MyISAM . Tendremos que tener en cuenta que si hacemos esto, no deberemos usar tablas MyISAM desde otro programa cuando la tabla está en uso. Hacerlo provocaría una corrupción de índice.

Espacio que ocupan

Las tablas MyISAM van a usar índices B-tree.

Podremos calcular su tamaño gracias a la fórmula (key_length+4)/0.67, sumada sobre todas las claves. Este es el peor caso en que todas las claves se insertan en orden ordenado y la tabla no tiene ninguna clave comprimida.

Los índices de cadenas de caracteres estarán siempre comprimidos. Si la primera parte del índice es una cadena de caracteres, también tiene el prefijo comprimido.

La compresión de espacio hace que el fichero índice sea menor que en el peor de los casos si la columna de la cadena de caracteres tiene muchos espacios finales o es una columna VARCHAR que no se usa siempre con la longitud total. La compresión de prefijo se usa en claves que comienzan con una cadena de caracteres. La compresión de prefijo ayuda si hay muchas cadenas de caracteres con un prefijo idéntico.

Formatos

Existen tres subtipos de tablas MyISAM: estáticas, dinámicas y comprimidas.

Cuando creamos las tablas, MySQL escoge entre el tipo dinámico o el tipo estático. El tipo predeterminado serán las tablas estáticas y se crean si no incluyen columnas VARCHAR, BLOB o TEXT. De lo contrario, la tabla se convierte en tabla dinámica.

Las tablas estáticas se caracterizan por lo siguiente:

- Tienen una longitud fija.
- Son más rápidas.
- Son más sencillas de almacenar en caché y de reconstruir tras un fallo.
- Necesitan más espacio en disco.

Las tablas dinámicas se caracterizan por lo siguiente:

- Sus campos tienen diferentes tamaños.
- Sus campos de cadena son dinámicos, a no ser que su tamaño sea inferior a 4 bytes.
- Ocupan menos espacio que las estáticas.
- Necesitan un mantenimiento regular para evitar su fragmentación.
- No son tan sencillas de reconstruir tras un fallo.

Las tablas comprimidas se caracterizan por lo siguiente:

- Son tablas de solo lectura que utilizan mucho menos espacio en disco.
- Son perfectas para su uso con datos comprimidos que no van a cambiar y donde no exista mucho espacio disponible.
- Son creadas mediante la utilidad "*myisampack*".
- Cada registro se comprimirá de forma separada, por lo que la carga de acceso es reducida.
- Cada columna se podría comprimir de forma diferente, mediante distintos algoritmos de compresión.
- Se pueden comprimir formatos de tabla fija y dinámica.

Ventajas e inconvenientes

Los índices MyISAM son mucho más pequeños que los índices ISAM. Debido a ello, el sistema utiliza menos recursos al realizar una operación de selección mediante un índice de una tabla MyISAM. Sin embargo, MyISAM requiere más potencia de procesador para insertar un registro dentro de un índice mucho más comprimido.

Otros problemas que nos podemos encontrar son los inconvenientes ante la corrupción de tablas de tipo MyISAM y problemas de difícil solución cuando las tablas no han sido cerradas correctamente.

Otro punto negativo será la ausencia de características de atomicidad, ya que no tiene que hacer comprobaciones de la integridad referencial, ni bloquear las tablas para realizar las operaciones. Sin embargo, esto tiene su parte positiva, ya que conseguimos una mayor velocidad.

En cuanto a las ventajas podemos comentar que tienen mayor velocidad en general a la hora de recuperar datos y que son más recomendables para aplicaciones en las que dominan las sentencias SELECT ante los INSERT / UPDATE.

2.6.2. Motor de almacenamiento InnoDB

Las tablas InnoDB son tablas de transacción segura. Cuando queremos realizar operaciones de inserción en una tabla MyISAM, la tabla se bloquea para que no pueda ser ejecutada ninguna otra instrucción sobre ella.

Sin embargo, en tablas InnoDB, se utilizan funciones de bloqueo en el nivel de fila de forma que solo se bloqueará esta fila y no toda la tabla, pudiendo entonces realizar otras instrucciones en otras filas.

Arranque

Para utilizar tablas InnoDB será necesario previamente compilar MySQL con compatibilidad InnoDB, y realizar una configuración de parámetros adicionales para obtener un buen rendimiento en las tablas.

Almacenamiento

Las tablas InnoDB se diferencian de las tablas MyISAM en que las bases de datos no van a ser almacenadas en un directorio, con las tablas como archivos. Las tablas y los índices se almacenarán en un espacio de tabla InnoDB que puede estar compuesto por una o varias tablas.

Por lo tanto, la restricción de tamaño de datos no vendrá dada por el límite del tamaño de archivos del sistema operativo.

Su tamaño inicial será de 16 Mb, y la responsabilidad de optimizar el crecimiento del espacio de las tablas recaerá en el administrador de las bases de datos.

Ventajas e inconvenientes

En cuanto a rendimiento, siempre será mejor utilizar tablas InnoDB en caso de necesitar realizar muchas operaciones de tipo insertar o actualizar sobre nuestras tablas, en detrimento de las operaciones de selección. En caso contrario, de utilizar sobre todo operaciones de tipo selección, será mejor utilizar tablas MyISAM.

Otra ventaja será la recuperación automática ante fallos. Si MySQL se cierra de una forma anormal, InnoDB automáticamente completará las transacciones que quedaron incompletas.

Además, el motor InnoDB implementa una técnica conocida como *multi-versioning* que elimina la necesidad de realizar bloqueos en consultas SELECT simples.

2.7. Conjunto de caracteres y colaciones

Un **conjunto de caracteres** es un conjunto de símbolos y codificaciones. Una **colación** es un conjunto de reglas para comparar caracteres en un conjunto de caracteres.

Un servidor MySQL puede soportar varios tipos de conjuntos de caracteres.

A su vez, un conjunto de caracteres podrá estar formado por un conjunto de colaciones.

Las colaciones tienen las siguientes características generales:

- Dos conjuntos de caracteres distintos no pueden tener la misma colación.
- Cada conjunto de caracteres tiene una colación que es la colación por defecto.
- Hay una convención para nombres de colaciones: empiezan con el nombre del conjunto de caracteres al que están asociados, normalmente incluyen el **nombre del idioma**, y acaban con **_ci** (no distingue entre mayúsculas y minúsculas), **_cs** (distingue entre mayúsculas y minúsculas) o **_bin** (binario).

2.7.1. Tipos

Un servidor MySQL puede soportar varios tipos de conjuntos de caracteres.

La lista es extensa, pero la podríamos clasificar de la siguiente forma:

- Conjuntos de caracteres Unicode.
- Conjuntos de caracteres de Europa occidental.
- Conjuntos de caracteres de Europa central.
- Conjuntos de caracteres del sur de Europa y de Oriente Medio.
- Conjuntos de caracteres bálticos.
- Conjuntos de caracteres cirílicos.
- Conjuntos de caracteres asiáticos.

En total, MySQL soporta más de 70 colaciones para más de 30 conjuntos de caracteres.

Si queremos ver la totalidad de los conjuntos de caracteres y sus colaciones por defecto, podremos ejecutar la sentencia "**SHOW CHARACTER SET**".

Para ver las colaciones de un conjunto de caracteres tendremos que hacer uso de la sentencia "**SHOW COLLATION**".

Si, por ejemplo, quisiéramos ver el conjunto de colaciones **latin1 ("ISO-8859-1 Europa Occidental"),** usaremos el siguiente comando:

EJEMPLO

```
SHOW COLLATION LIKE 'latin1%';
```

2.7.2. Elección de cotejamiento

Los siguientes ejemplos muestran cómo MySQL determina los valores del conjunto de caracteres y colación por defecto.

Primer ejemplo. Definición de tabla y columna

EJEMPLO

```
CREATE TABLE t1
(c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Aquí tenemos una columna con un conjunto de caracteres latin1 y una colación latin1_german1_ci.

Segundo ejemplo. Definición de tabla y columna

EJEMPLO

```
CREATE TABLE t1
(c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

En este ejemplo tenemos una columna con un conjunto de caracteres latin1 y la colación por defecto.

Tercer ejemplo. Definición de tabla y columna

EJEMPLO

```
CREATE TABLE t1
( c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

Tenemos una columna con un conjunto de caracteres y colación por defecto. Bajo esta circunstancia, MySQL busca a nivel de tabla para determinar el conjunto de caracteres y colación para la columna. Así, el conjunto de caracteres para la columna c1 es latin1 y su colación es latin1_danish_ci.

Cuarto ejemplo. Definición de base de datos, tabla y columna

EJEMPLO

```
CREATE DATABASE d1 DEFAULT CHARACTER SET latin2 COLLATE la-
tin2_czech_ci;
USE d1;
CREATE TABLE t1( c1 CHAR(10));
```

Creamos una columna sin especificar su conjunto de caracteres y colación. Tampoco especificamos un conjunto de caracteres y colación a nivel de tabla. En este caso, MySQL usará el nivel de base de datos. Así, el conjunto de caracteres para la columna c1 es latin2 y su colación es latin2_czech_ci.

2.8. Gestión gráfica de MySQL – PHPMyAdmin

Con PHPMyAdmin podremos gestionar nuestras bases de datos MySQL de forma sencilla e intuitiva, incluso sin que sea necesario tener conocimientos de programación en SQL para realizar todas las operaciones que hemos estado viendo en esta unidad.

2.8.1. Acceso a la aplicación

Para acceder a la aplicación, tendremos que abrir un navegador y escribir en la barra de direcciones la ruta donde tengamos instalado PHPMyAdmin. En nuestro caso, al haber instalado la aplicación EasyPHP, nos lo ha instalado automáticamente en la siguiente ruta: http://127.0.0.1:8888/home/mysql/.

La primera pantalla que tendríamos que obtener sería la siguiente:

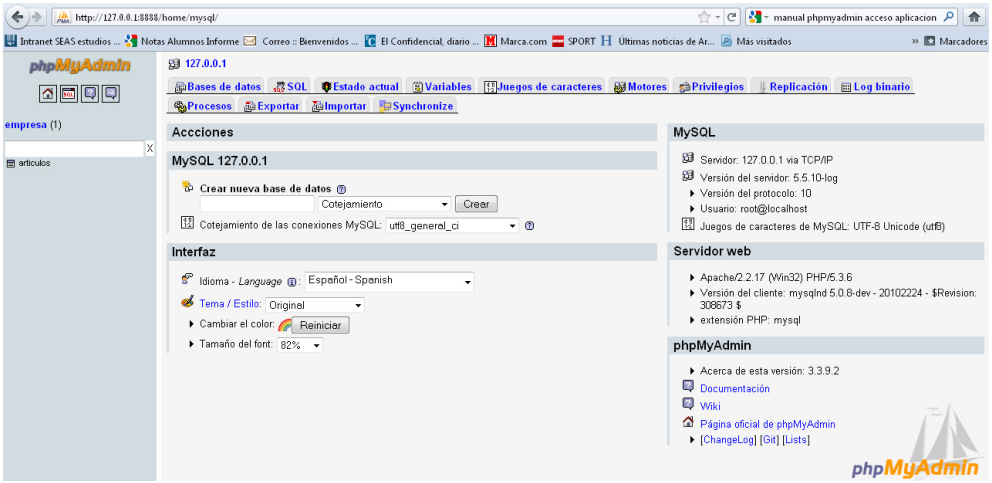

Figura 2.45. Pantalla de bienvenida a PHPMyAdmin.

2.8.2. Crear una base de datos

Desde la pantalla de bienvenida, ya podremos crear una base de datos. Creamos una bbdd denominada “colegio”:

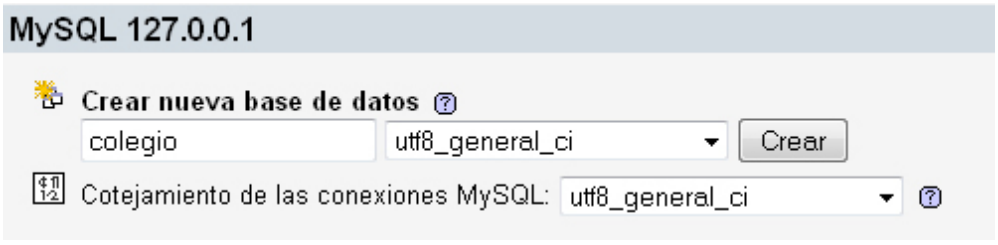

Figura 2.46. Creación de base de datos “colegio”.

2.8.3. Gestión de tablas

Una vez que ya tenemos creada nuestra base de datos, tendremos que proceder a crear las tablas que necesitemos.

Crear una tabla

Al crear la base de datos, aparecerá que no tenemos ninguna tabla creada. Creamos una tabla cuyo nombre sea “alumnos”, con 6 campos:

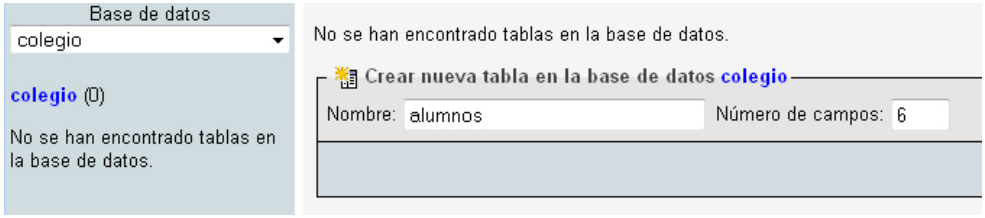

Figura 2.47. Creación de tabla.

Agregar un campo a una tabla

Una vez creada la tabla, automáticamente nos saldrá la pantalla con los 6 campos disponibles para introducir sus características. Para insertarlos, tendremos que pulsar el botón “Grabar”:

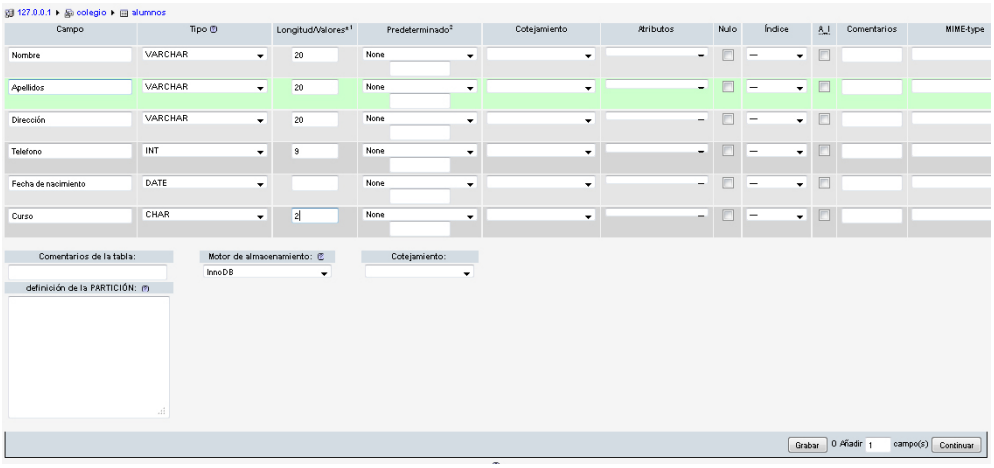

Figura 2.48. Insertar campos.

A continuación, nos mostrará el resumen de la tabla y las operaciones que podemos realizar sobre ella:

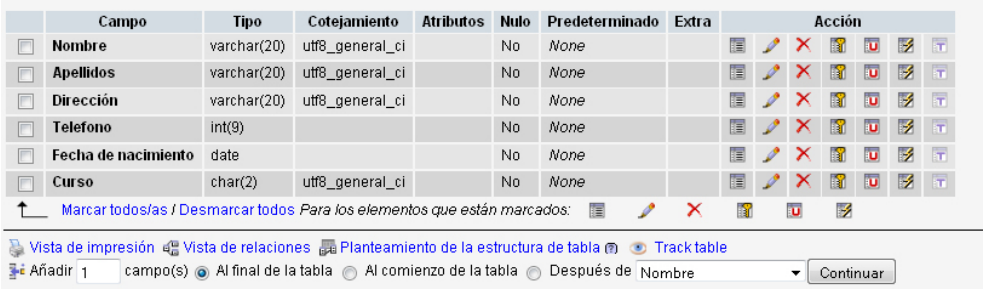

Figura 2.49. Tabla creación y operaciones posibles.

Modificar un campo de una tabla

Si quisiéramos modificar un campo de la tabla que acabamos de crear, tendríamos que elegir el campo y pulsar el botón "cambiar" representado por la imagen:  . A continuación, nos tendría que aparecer la pantalla disponible para la actualización del campo. Una vez realizados los cambios, tendremos que pulsar el botón "guardar":

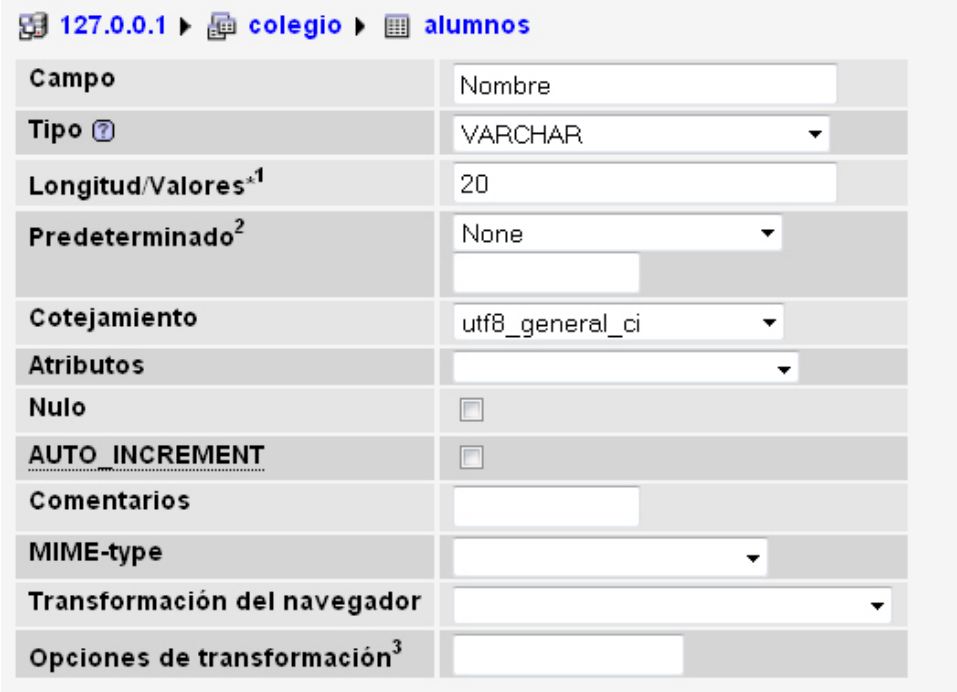

Figura 2.50. Modificar un campo.

Eliminar un campo de una tabla

Si quisiéramos eliminar un campo de la tabla que acabamos de crear, tendríamos que elegir el campo y pulsar el botón "eliminar" representado por la imagen:  . A continuación, nos tendría que aparecer en la pantalla la instrucción en formato SQL que queremos realizar, en este caso eliminar el campo "Curso" de la tabla "alumnos". Pulsaremos en el botón "Aceptar" para eliminar el campo:

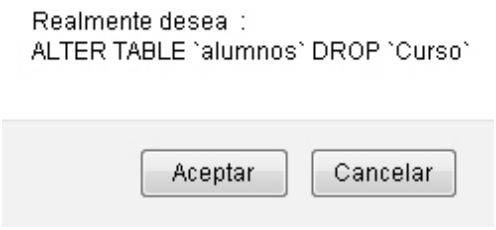

Figura 2.51. Eliminar un campo.

2.8.4. Gestión de registros

A continuación, vamos a ver las operaciones básicas que podremos realizar con los registros de una tabla: añadir, actualizar, eliminar y buscar.

Agregar un registro

Para añadir un nuevo registro a nuestra tabla, tendremos que pulsar en la opción "Insertar" del menú principal superior, para que nos muestre la pantalla de introducción de datos. Una vez introducidos, tendremos que pulsar el botón "Continuar":

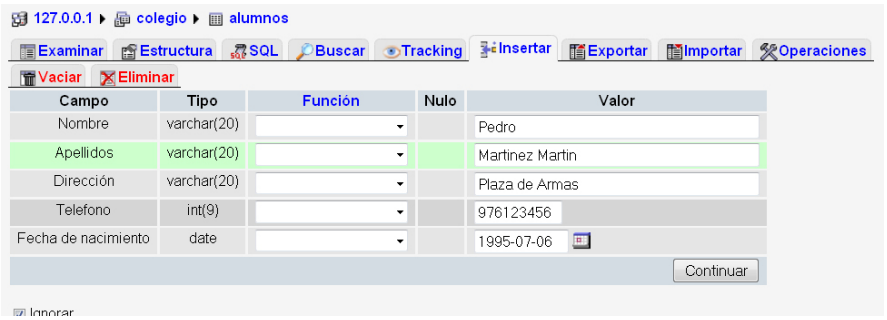

Figura 2.52. Insertar registro en una tabla.

Una vez introducidos los datos, si pinchamos sobre la tabla "alumnos" a la izquierda de la pantalla, podremos ver un resumen de los registros que tenemos en nuestra tabla:

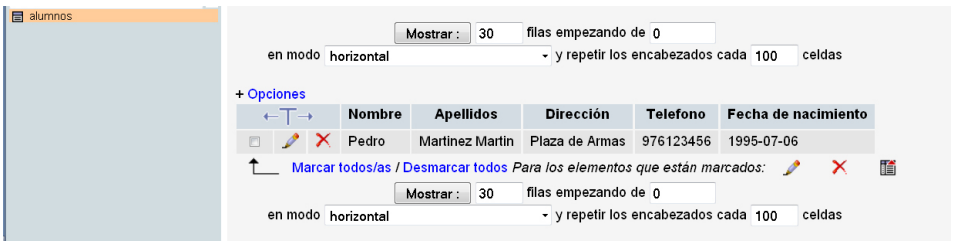

Figura 2.53. Ver registros en una tabla.

Modificar un registro

Para modificar un registro, tendremos que marcarlo en primer lugar a través de checkbox que se encuentra a la izquierda, y a continuación, pulsar el botón "cambiar" representado por la imagen:  .

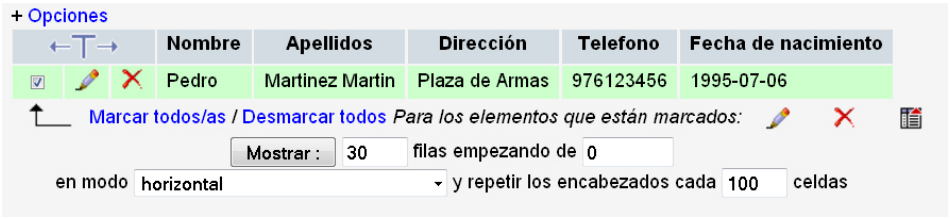

Figura 2.54. Elección de registro a modificar en una tabla.

Elegimos el campo o campos que queremos modificar (campo **Nombre** en nuestro caso), y pulsamos el botón "**Continuar**":

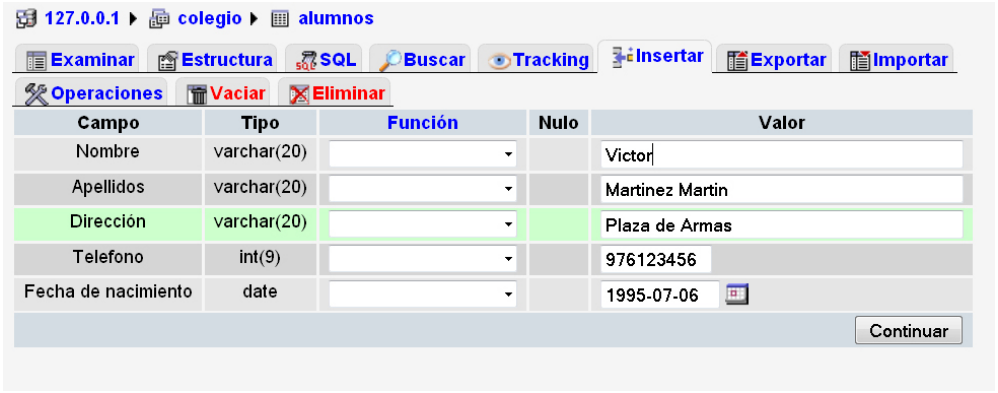

Figura 2.55. Modificación de registro en una tabla.

Eliminar un registro

Para eliminar un registro, tendremos que marcarlo en primer lugar a través de checkbox que se encuentra a la izquierda, y a continuación, pulsar el botón "**eliminar**" representado por la imagen:  .

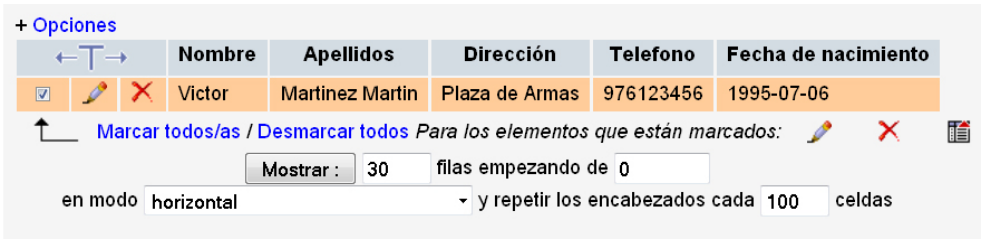

Figura 2.56. Elección de registro a modificar en una tabla.

A continuación, nos pedirá confirmación de la operación, mostrándonos de paso el equivalente a una sentencia SQL:

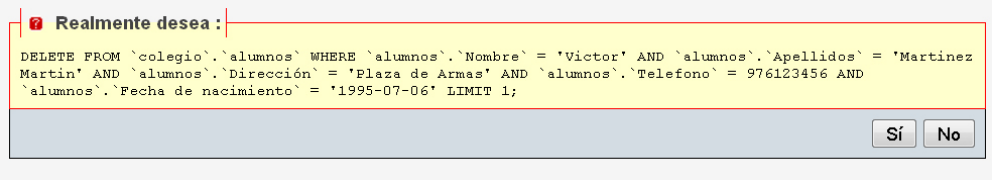

Figura 2.57. Confirmación de eliminación de registro.

Búsqueda de datos en los registros

Para poder localizar registros dentro de una tabla, tendremos que ir al menú principal localizado en la parte superior y escoger la opción "Buscar". A continuación, nos tendría que aparecer una pantalla con el total de campos de nuestra tabla para que introduzcamos el valor que queremos buscar, en el campo a través del cual queramos buscar. La condición de la búsqueda vendrá especificada por la columna "Operador" donde podremos encontrar una serie de operadores de comparación, como ya explicamos en capítulos anteriores:

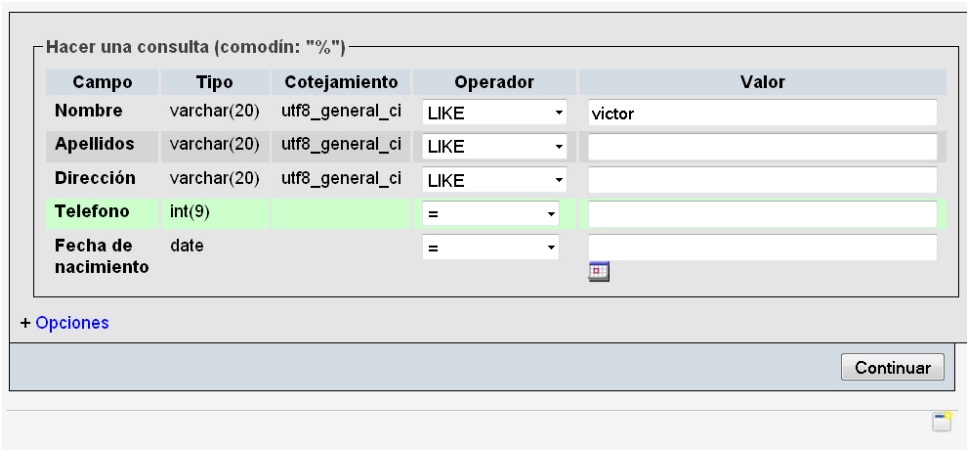

Figura 2.58. Ver registros en una tabla.

2.8.5. Importar y exportar bases de datos

Podremos importar datos a nuestra base de datos, mediante la importación de un fichero que tenga definidas todas las operaciones a realizar en el momento de la importación.

Para ello, tendremos que pulsar el botón "Importar" que tenemos en el menú principal en la zona superior:

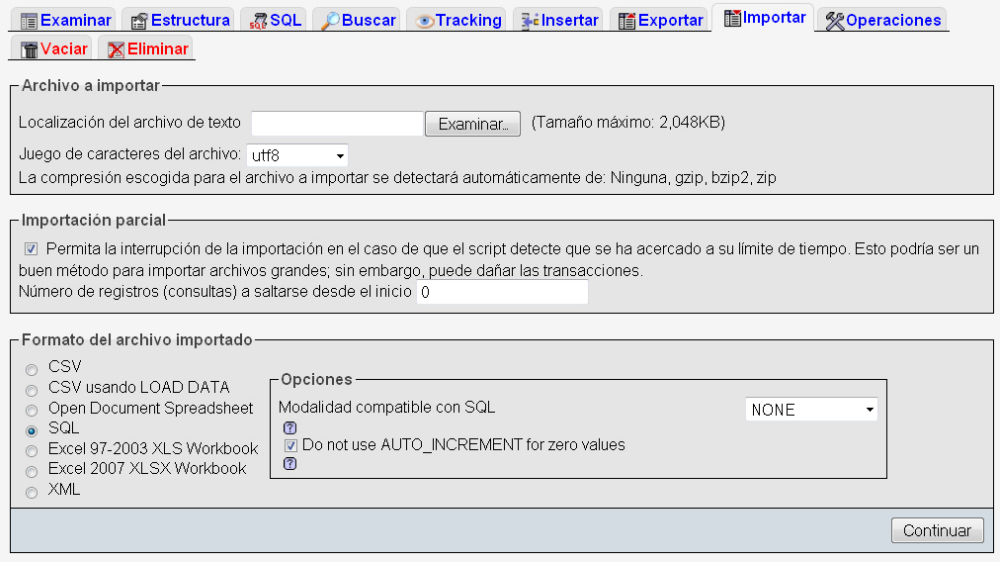

Figura 2.59. Importación de datos.

Podremos **exportar** datos y volcarlos a un fichero que determinemos, por ejemplo, para hacer una copia de seguridad, o para posteriormente importarlo en otra base de datos en otro servidor.

Para ello, tendremos que pulsar el botón "**Exportar**" que tenemos en el menú principal en la zona superior. Una vez situados, podemos comentar aspectos a tener en cuenta.

El primero, los formatos a los que podemos exportar:

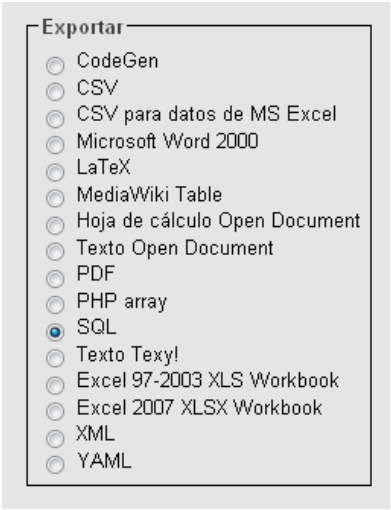

Figura 2.60. Tipos de fichero para exportar datos.

El segundo, las operaciones que podemos realizar en cuanto a la estructura y datos de la exportación:

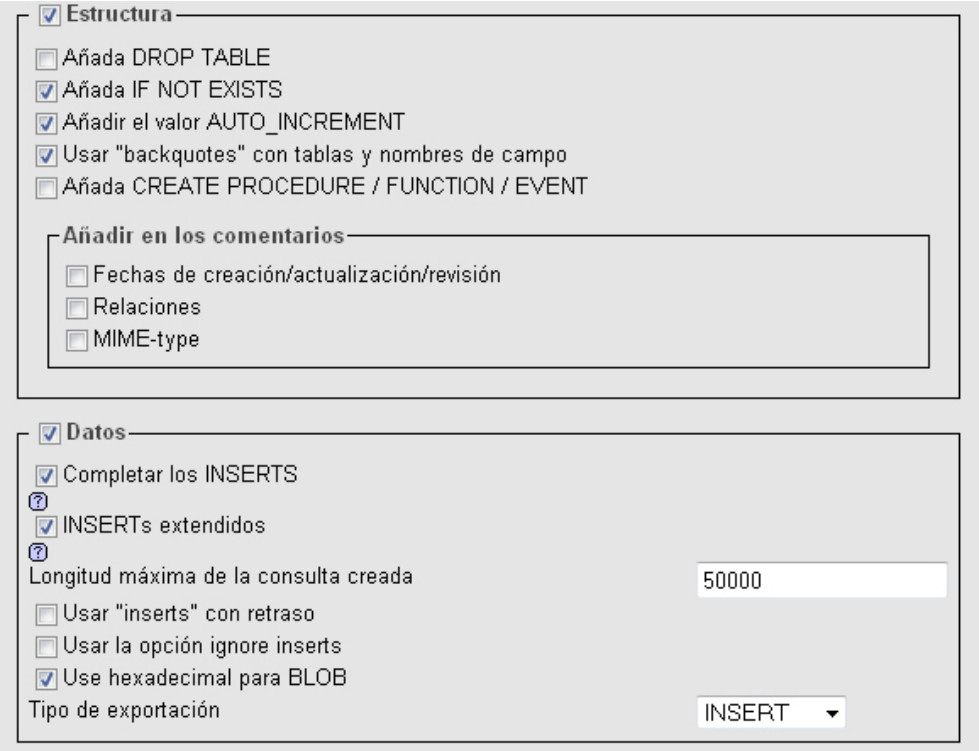

Figura 2.61. Opciones de exportación de datos.

Por ejemplo, podría añadir una sentencia "DROP TABLE" de forma previa a la creación de la misma, para asegurarnos que si está duplicada, primero la eliminaría.

Por último, podremos decir si queremos guardar el fichero en el servidor o el nombre con el que queremos nombrar al archivo:

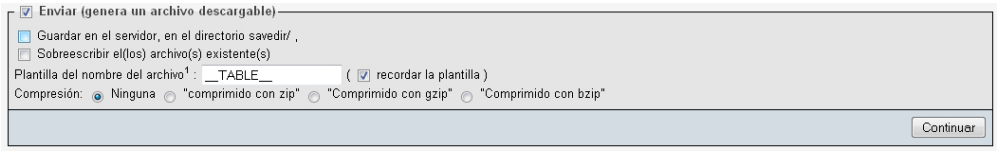

Figura 2.62. Opciones de envío de fichero exportado.

2.8.6. Pestaña de operaciones

En esta pestaña disponemos de una serie de operaciones básicas de modificación y mantenimiento de la base de datos.

Entre otras, podremos realizar estas operaciones:

- Modificar el orden de aparición de los campos en una tabla.
- Cambiar el nombre de la tabla.
- Mover tablas de una base de datos a otra.

- Copiar tablas de una base de datos a otra.
- Añadir comentarios a una tabla.
- Definir el tipo de tabla.
- Collation.

En lo referente a mantenimiento de la tabla, podremos:

- Revisar la tabla.
- Analizar la tabla.
- Reparar la tabla.
- Optimizar la tabla.
- Vaciar caché.

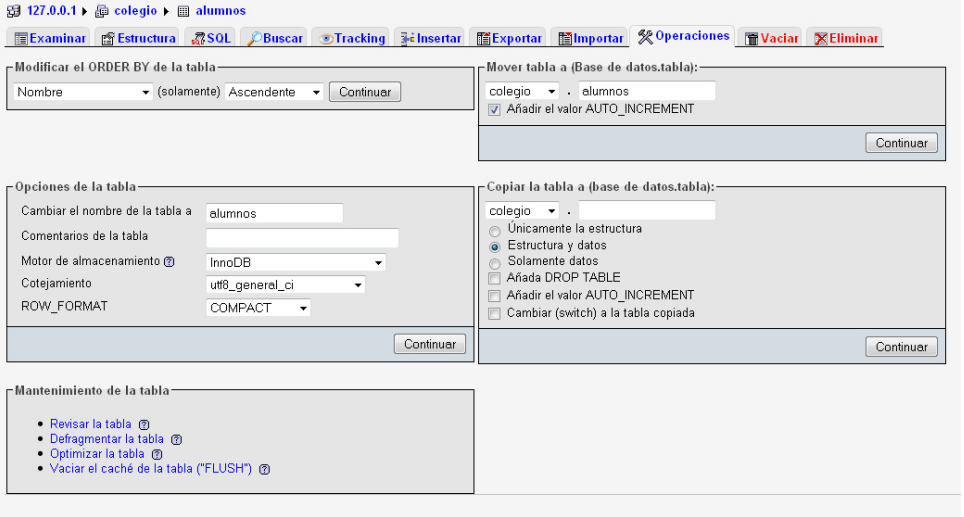

Figura 2.63. Posibles operaciones sobre tablas.

2.8.7. Generar código PHP

PHPMyAdmin incorpora una herramienta que nos posibilitará crear código PHP de una manera automática cuando estemos realizando determinadas operaciones.

Por ejemplo, si pinchamos sobre la tabla "alumnos", en el panel de la derecha nos tendría que aparecer un listado con los usuarios que tenemos en nuestra tabla ya insertados. Si nos fijamos, en la parte superior derecha, podremos encontrar una opción denominada "Crear código PHP":

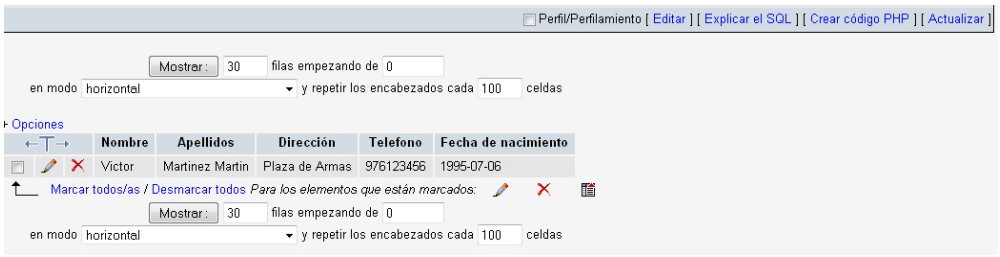

Figura 2.64. Opción crear código PHP.

Si lo ejecutamos, nos habrá creado una instrucción en PHP, donde define una variable que va a almacenar la instrucción a ejecutar. En este caso, como estamos visualizando un listado de la tabla, se transformaría en una operación de consulta SELECT que mostraría todos los campos de todos los registros de la tabla “alumnos” con una paginación de 30 registros por pantalla:

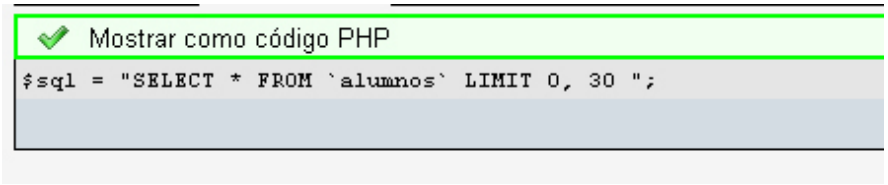

Figura 2.65. Código PHP.

RESUMEN

- Se ha explicado la forma de conectarse a un servidor de bases de datos para realizar tareas de creación de bases de datos, tablas y registros.
- Se han explicado los diferentes tipos de datos que podemos utilizar para la creación de registros.
- Se han explicado las principales funciones que podemos utilizar para el manejo de cadenas de caracteres, datos numéricos y fechas.
- Se ha explicado la forma en la que MySQL administra los privilegios para los usuarios y los diferentes comandos posibles para realizar tareas de mantenimiento de privilegios.
- Hemos introducido otras características técnicas importantes de las bases de datos como son los motores de almacenamiento y los cotejamientos.
- Se ha explicado la aplicación PHPMyAdmin para realizar todas las operaciones desde un navegador.