MySQL

guía rápida
(versión Windows)

Autor: Jorge Sánchez (www.jorgesanchez.net) año 2004

Manual breve para el manejo de la base de datos de código abierto MySQL (http://www.mysql.com)

índice

índice **1**
introducción **2**
mySQL y el código abierto 2
instalación 2
conexión y desconexión al servidor 4
entrada de comandos en el monitor 4
bases de datos **6**
comando *show databases* 6
utilizar una base de datos 6
crear una base de datos 6
borrar bases de datos 7
tablas 7
operaciones con tablas 11
tablas innoDB 12
introducir datos en una tabla 12
índices 14
consultas **17**
obtener registros. *select* 17
update 20
delete 20
uniones 21

© Jorge Sánchez' 2003

introducción

mySQL y el código abierto

MySQL es un sistema gestor de bases de datos. Pero la virtud fundamental y la clave de su éxito es que se trata de un **sistema de libre distribución** y de **código abierto**. Lo primero significa que se puede descargar libremente de Internet (por ejemplo de la dirección (www.mysql.com); lo segundo (código abierto) significa que cualquier programador puede remodelar el código de la aplicación para mejorarlo.

Esa es también la base del funcionamiento del sistema Linux, por eso MySQL se distribuye fundamentalmente para Linux, aunque también hay versiones para Windows.

Existen cuatro versiones de MySQL:

- **Estándar.** Incluye el motor estándar y la posibilidad de usar bases de datos **InnoDB.** Todo el potencial de MySQL, pero sin soporte completo para utilizar transacciones.
- **Max.** Para usuarios que quieran MySQL con herramientas de prueba para realizar opciones avanzadas de base de datos
- **Pro.** Versión comercial del MySQL estándar
- **Classic.** Igual que la estándar pero no dispone de soporte para **InnoDB**

El uso de MySQL (excepto en la versión Pro) está sujeto a licencia **GNU public license** (llamada GPL). Está licencia admite el uso de MySQL para crear cualquier tipo de aplicación. Se pueden distribuir copias de los archivos de MySQL, salvo esas copias se cobren a un tercer usuario. Se prohíbe cobrar por incluir MySQL. Se puede modificar el código fuente de MySQL, pero si se distribuye la aplicación con el código modificado, habrá que obtener una copia comercial y consultar sobre el cobro de la licencia. Al distribuir copias, se tiene que poder obtener información sobre las licencias GNU (más información en http://dev.mysql.com/doc/mysql/en/GPL_license.html

Se puede también obtener una licencia comercial que permitiría cobrar las instalaciones MySQL, incluir la base de datos en ordenadores y cobrar por ello, y otras situaciones no reflejadas en la licencia GNU.

instalación

instalación de los ficheros del programa

La descarga de los programas se realiza desde www.mysql.com, Hay dos posibilidades de instalación en Windows.

1 > La más sencilla es utilizar el programa de instalación, el cual colocará todos los elementos necesarios en las carpetas correspondientes. La instalación no diferirá demasiado respecto a cualquier otra instalación bajo Windows.

2> Bajarse los archivos comprimidos sin la instalación. Bastará con descomprimirlos en C:

Supondremos que la instalación se realiza con el primer método (que es lo más común). Normalmente el programa se instalará en la carpeta **MySQL** dentro del disco duro principal (el C:). Todos los ejecutables se encontrarán en la carpeta ***bin*** que estará dentro de la anterior.

No habrá ningún acceso al programa desde el apartado **programas** de Windows (debemos hacerlo nosotros manualmente si lo deseamos).

configuración del entorno

El entorno se puede configurar a mano a través de un archivo llamado `mysql.ini` que normalmente se encuentra en la carpeta **Windows** o **Winnt**.

No obstante es más sencillo hacerlo con el programa **WinMySQLAdmin**. Este es un programa realizado para Windows que nos permite editar mysql.ini de forma más sencilla.

La primera vez que este programa arranca nos pregunta nuestro nombre de usuario y contraseña. Los usuarios de MySQL no tienen nada que ver con los de Windows. Tras indicar el usuario, el programa colocará los valores más usuales de configuración. Cada vez que volvamos a arrancar este programa, se nos permitirá cambiar la configuración.

arrancar el servidor

Tras la instalación hace falta instalar el servidor MySQL de bases de datos. Hay varias posibilidades de servidores, son:

archivo (en la carpeta bin)	uso
mysqld	Compilado con depuración completa y chequeo automático de colocación de memoria, enlaces simbólicos, *InnoDB* y tablas BDB. Es el más genérico (para cualquier tipo de ordenador)
mysqld-opt	Optimizado para equipos Pentium. No tiene soporte de tablas transaccionales.
mysqld-nt	Optimizado para equipos con Windows NT, 2000 y XP con soporte para *pipe names*
mysql-max	Optimizado con soporte de enlaces simbólicos, *InnoDB* y tablas BDB.
mysql-max-nt	Igual que el anterior pero combinado las capacidades del servidor **mysql-nt**

Antes de seleccionar el servidor a arrancar hay que asegurarse de cerrar el servidor que estuviera en funcionamiento. Esto se realiza (desde la línea de comandos) con el nombre del servidor, seguido de la palabra **--remove.** Ejemplo:

```
c:\mysql\bin>mysqld --remove
```

Para arrancar el servidor deseado, se coloca la palabra **--install.** Ejemplo:

```
c:\mysqld\bin>mysqld-max-nt --install
```

Si está bien instalado. Aparecerá en la línea de comandos el texto ***"Service successfully installed"***.

Otra posibilidad es utilizar el administrador de MySQL y desde él decidir cómo arrancar el servidor. Eso se realiza en la línea **Server** dentro de la sección **[WinMySQLadmin].** Ejemplo:

```
[WinMySQLadmin]
Server=C:/mysql/bin/mysqld-max-nt.exe
```

El servidor se ejecuta en segundo plano, sólo el administrador de tareas permite ver su ejecución. No conviene parar el servidor desde el administrador de tareas ya que al cortar de repente los datos de seguridad podrían quedar malparados. En su lugar conviene cerrar desde la línea de comandos con el parámetro **--remove** comentado anteriormente.

conexión y desconexión al servidor

La conexión al servidor MySQL para crear, modificar o realizar cualquier otra operación sobre bases de datos. Se realiza mediante el programa **mysql** que también se encuentra en la carpeta **bin** del programa.

La ejecución de este programa nos lleva al llamado **monitor** de MySQL que es la línea de comandos desde la que podemos ejecutar instrucciones MySQL.

Este programa posee numerosas opciones de arranque. Para saber cuáles son, basta ejecutar la instrucción `mysql --help` desde la línea de comandos.

Normalmente la entrada normal al monitor se hace mediante:

```
c:\mysql\bin> mysql -h host -u user -p
```

Donde *host* es el nombre del servidor (si no se ejecuta desde red será *localhost*) y *user* es el nombre del usuario que desea utilizar el servidor. El último parámetro sirve para poder introducir la contraseña. Ésta será pedida inmediatamente después de haber sido pulsada la tecla `Intro`.

En la mayoría de instalaciones (sobre todo en Windows) se puede entrar simplemente poniendo **mysql** desde la línea de comandos. Esto asumirá como usuario anónimo y como servidor el servidor local. Hay un usuario de privilegios absolutos que se llama **root**.

En cualquier caso si hemos accedido bien al monitor, en la línea de comandos aparecerá el texto **`mysql>`**. Que indicará que el monitor está esperando comandos para ejecutar sobre el servidor.

Para abandonar el monitor basta escribir el comando **`quit`**.

entrada de comandos en el monitor

MySQL utiliza el lenguaje SQL de bases de datos para trabajar. Esas serán las sentencias que normalmente se utilizan en el monitor.

Sobre los comandos hay que tener en cuenta que:

1> Da lo mismo escribir en mayúsculas o en minúsculas

2> Todos los comandos terminan con el símbolo ";"

3> El comando termina su ejecución si en la línea de comandos observamos el texto mysql>

4> Se pueden realizar operaciones aritméticas (3*6)

5> En la misma línea se pueden colocar dos comandos (por ejemplo: `SELECT 3*6; SELECT SIN(PI());`) siempre y cuando los puntos y comas se coloquen de forma adecuada

6> Una instrucción puede abarcar más de 1 línea (para informar que no ha terminado la instrucción, el monitor coloca el símbolo "**->**", en lugar del normal "mysql>"). Ejemplo:

```
mysql> SELECT * FROM clientes
-->WHERE apellido="Jiménez";
```

7> Una instrucción se puede anular antes del punto y coma, colocando el texto "**\c**"

8> Las cadenas de texto literal puede ir entre símbolos de comilla simple o símbolos de comilla doble. Si se pulsa `Intro` antes de cerrar la cadena, el monitor lo índica mostrando `'>` o `">` en lugar del habitual `-->` o `mysql>`.

© Jorge Sánchez' 2003

bases de datos

MySQL almacena las bases de datos en la carpeta **data** que está en la carpeta raíz de la instalación del programa. Cada base de datos crea una carpeta en la que aparecen los archivos necesarios para el correcto funcionamiento de la aplicación.

comando *show databases*

El comando **show databases** permite visualizar las bases de datos actualmente activas. Ejemplo:

```
mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.01 sec)
```

En el ejemplo hay dos bases de datos activas, la principal llamada **mysql** y la base **test** (una base de datos de prueba). Hay bases de datos que no nos serán mostradas si no tenemos permiso para ello.

utilizar una base de datos

El comando **use** nos permite utilizar una base de datos. Es (junto a **quit**) el único comando que no requiere punto y coma.

```
mysql> use test
```

Eso hace que **test** sea la base de datos de trabajo actual.

También se puede seleccionar la base de datos para utilizar al arrancar el propio monitor. Para ello basta poner el nombre de la base de datos tras el comando **mysql**

crear una base de datos

Se realiza de esta forma:

```
mysql>create database prueba;
Query OK, 1 row affected (0.00 sec)
```

La base de datos recién creada estará representada por una carpeta dentro de la carpeta **data** de **mysql**.

Aunque la base esté creada, habrá que utilizar el comando **use** para trabajar con ella.

borrar bases de datos

Se trata del comando **drop database** al cual le sigue el nombre de la base de datos.

```
mysql>drop database prueba;
Query OK, 0 rows affected (0.00 sec)
```

tablas

mostrar tablas

El comando **show tables;** muestra las tablas de la base de datos actual. Ejemplo:

```
mysql> use mysql
Database changed
mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)
```

Se puede utilizar la sintaxis **show tables from** *bd,* donde *bd* es el nombre de una base de datos que no tiene porque estar en uso.

crear tablas

Esta es ya una operación muy importante. Es el comando **create table** el que la realiza. Este comando funciona con esta sintaxis:

```
create table nombreTabla (nombrecampo1 tipodatos(tamaño),
nombrecampo2 tipodatos(tamaño),....);
```

Ejemplo:

```
mysql> create table personas (nombre varchar(30),
  ->apellido1 varchar(30), apellido2 varchar(30),
  ->telefono varchar(9));
Query OK, 0 rows affected (0.01 sec)
```

indicar opciones en los campos en la creación

Durante la creación de campos se pueden indicar opciones (las opciones se enumeran más abajo) sobre los campos. Estas opciones se colocan tras el tipo de datos del campo. Ejemplo (**not null** y **unique**):

```
mysql> create table personas (nombre varchar(30) not null,
  ->apellido1 varchar(30), apellido2 varchar(30),
  ->telefono varchar(9) unique);
```

establecimiento de la clave durante la creación

Se puede añadir la palabra **primary key** tras el tipo de datos del campo que se desea sea la clave de la tabla. Si la clave es más de un campo se realiza colocando la palabra **primary key** como nombre de campo, seguida de los campos que forman la clave entre paréntesis. Ejemplo:

```
mysql> create table pieza (codigo1 varchar(5),
  ->codigo2 int(2), peso int,
  ->descripcion text,
  ->primary key (codigo1, codigo2);
```

estructura de las tablas

El comando **describe** seguido del nombre de una tabla, nos permite ver la estructura completa de una tabla. Ejemplo:

```
mysql> describe personas;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| nombre    | varchar(30) | YES  |     | NULL    |       |
| apellido1 | varchar(30) | YES  |     | NULL    |       |
| apellido2 | varchar(30) | YES  |     | NULL    |       |
| telefono  | varchar(9)  | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

tipos de datos

numéricos

tipo	Espacio	Rango	rango sin signo
TINYINT	1 byte	-128 a 127	0 a 255
SMALL INT	2 bytes	-32768 a 32767	0 a 65535
MEDIUM INT	3 bytes	-8388608 a 8388607	0 a 16777215
INT	4 bytes	-2147483648 a 2147483647	0 a 4294967295

tipo	Espacio	Rango	rango sin signo
BIG INT	8 bytes	-9,223 * 10^{18} a 9,223 *10^{18}	0 a 18,446 * 10^{36}
FLOAT(*M,D*)	4 bytes	varía según los parámetros	
DOUBLE(*M,D*)	8 BYTES	varía según los parámetros	
DECIMAL(*M,D*)	M+2 bytes	varía según los parámetros	

modificadores numéricos

Los números enteros se pueden colocar sin signo adelantando la palabra UNSIGNED. Los decimales permiten indicar el número de la mantisa (M) y el número de decimales (D).

A los números enteros se les puede asignar el modificador AUTO_INCREMENT para que el campo marcado almacene más rápido los valores.

Se puede declarar una anchura de columna de esta forma: INT(8). Si además se define la columna con ZEROFILL, entonces se colocan ceros a la izquierda hasta llegar a esa anchura.

texto

tipo	Espacio	Tamaño máximo
CHAR(*X*)	X bytes	255 bytes
VARCHAR(X)	X+1 byte	255 bytes
TINYTEXT	X+1 byte	255 bytes
TINYBLOB	X+1 byte	255 bytes
TEXT	X+2 bytes	65535 bytes
BLOB	X+2 bytes	65535 bytes
MEDIUMTEXT	X+3 bytes	1,6 MB
MEDIUMBLOB	X+ 3bytes	1,6 MB
LONGTEXT	X+4 bytes	4,2 GB
LONGBLOB	X+4 bytes	4,2 GB

Los que más se usan son CHAR y VARCHAR. Ambos almacenan textos pero CHAR los almacena de tamaño fijo y VARCHAR optimiza tamaño variable. Dicho de otra forma: si definimos un CHAR de tamaño 8, cualquier registro ocupa 8 bytes. Con VARCHAR si algún registro ocupa 4 caracteres, en disco ocupará 5 bytes independientemente del tamaño indicado inicialmente.

TEXT y BLOB se usan para grandes cantidades de texto variable. La diferencia es que los tipos BLOB distinguen entre mayúsculas y minúsculas.

lógicos

Son los tipos **BIT** o **BOOL** que admiten los valores 0 o 1.

© Jorge Sánchez' 2003

fechas

tipo	Rango	Formato
DATE	Del 1 de enero de 1001 al 31/12 del 9999	año-mes-día
DATETIME	De las 0 horas del 1/1/1001 a las 0 horas del 31/12/9999	Año-mes-día horas:minutos:segundos
TIMESTAMP	De las 0 horas del 1/1/1970 a las 0 horas del 31/12/2037	Año-mes-día horas:minutos:segundos Permite estos tamaños: ◉ 14 (aaaammddhhmmss) ◉ 12 (aaaammddhhmm) ◉ 8 (aaaammdd) ◉ 6 (aammdd) ◉ 4 (aamm) ◉ 2 (aa)
TIME	Una hora en formato HH:MM:SS	
YEAR	Año desde 1901 a 2037	

ENUM

Permite declarar campos cuyo contenido puede ser una de entre varias opciones. Ejemplo:

```
CREATE TABLE personas (Sexo ENUM('M', 'H'),.....
```

modificadores

modificador	Se aplica a	Uso
AUTO_INCREMENT	Enteros	El valor se va incrementando automáticamente en cada registro
BINARY	char y varchar	Convierte las cadenas a forma binaria en la que se distingue entre mayúsculas y minúsculas
DEFAULT	Todos menos TEXT y BLOB	Coloca un valor por defecto el valor se coloca justo detrás de esta palabra)
NOT NULL	Todos	Impide que un campo sea nulo
PRIMARY KEY	Todos	Hace que el campo se considere clave primaria

modificador	Se aplica a	Uso
UNIQUE	Todos	Evita la repetición de valores
UNSIGNED	Enteros	Sólo valen los valores positivos
ZEROFILL	Enteros	Rellena con ceros a la izquierda hasta llegar al ajuste.

Ejemplo:

```
mysql>create table vehiculos (Modelo VARCHAR(20) NOT NULL
DEFAULT "Megane");
```

operaciones con tablas

modificar tablas

Es el comando **alter table** el encargado. Con el atributo **change** modifica un campo. Con el atributo **rename** cambia de nombre una tabla

```
mysql>alter table personas change nombre nombre varchar(20);
```

En el ejemplo el nombre pasa a tener tamaño 20.

```
mysql>alter table personas change nombre identif
-->varchar(20);
```

El campo **nombre** pasa a llamarse **identif**.:

```
mysql>alter table personas change identif nombre
-->varchar(25) not null default "pepe";
```

Cambia el campo **identif** para que ahora sea pepe, tenga 25 caracteres de anchura, no admita nulos y establezca como valor por defecto, la cadena "pepe".

Cambio de nombre de la tabla:

```
mysql>alter table personas rename clientes;
```

borrar tablas

Se usa la sintaxis: **drop table** seguida del nombre de la tabla.

borrar columnas

La sintaxis es:

```
alter table tabla drop columnaABorrar;
```

añadir columnas

Se usa:

```
alter table tabla add nuevaColumna Tipo...;
```

varias operaciones a la vez

Se pueden realizar varias operaciones a la vez mediante la instrucción **alter table** si se separan con comas:

```
alter table ejemplo drop campo1, add campo2, change campo3
campo4 int(8);
```

De este modo, **alter table** se convierte en una instrucción muy versátil para modificar la estructura de las bases de datos.

tablas innoDB

MySQL proporciona un motor de gestión de base de datos llamado innoDB que permite una mayor potencia sobre el control y gestión de tablas. Las tablas normales, son creadas con el motor **myISAM**, para hacer que sean **innoDB**, basta con colocar el texto **type=innoDB** tras la lista de campos de la tabla en la creación.

También se puede convertir una tabla al tipo **innoDB** si se ejecuta la orden:

```
alter table tabla type=innoDB;
```

Las tablas innoDB permiten transacciones, operaciones concurrentes y control estricto de referencias (lo que se conoce como integridad referencial) entre otros detalles. Pero requieren más potencia por parte del sistema en el que se ejecuta MySQL.

introducir datos en una tabla

Hay dos métodos.

el comando *insert*

Permite añadir datos manualmente a una tabla. Ejemplo:

```
mysql> insert into personas values('Pedro',
  ->'Hernández','Crespo','979898989');
Query OK, 1 row affected (0.00 sec)
```

El orden de los datos debe corresponder con el de los campos de la tabla. También se puede indicar los campos:

```
mysql> insert into personas (nombre,apellido1,apellido2)
    -> values ('Pedro','Hernández',"crespo");
```

En ese caso la columna del teléfono tomaría el valor **null** indicando que está vacío.

insertar datos en una tabla desde un archivo externo

Otra manera más poderosa es utilizar un archivo externo en el que se colocan los datos de la tabla. En ese archivo, cada registro se separa con un carácter concreto (que suelen ser los caracteres \r \n, resultado de la tecla intro). A su vez en el mismo registro, cada

campo se separa con otro carácter (coma por ejemplo) y los valores de los campos podrían ir delimitados con otros caracteres (como las comillas).

Para conseguir los datos en este formato, se pueden colocar a mano, o bien se pueden crear a través de software especial que consiga colocar registros en este formato (programas como Excel, Access u otros muchos poseen herramientas para realizar estas operaciones.

Una vez creado el archivo, desde MySQL se podría usar el comando **load data:**

```
load data local infile "ruta" into table tabla;
```

Ejemplo:

```
mysql>load data infile
  ->"C:\\mysql\data\prueba\texto.txt"
  ->into table personas;
```

La *ruta* es la ruta completa al archivo de texto que posee los datos. La *tabla* es la tabla a la cual se desean añadir dichos datos. Para que esta instrucción funcione correctamente, el orden de los campos en la tabla debe ser el mismo que en el archivo. Salvo que se indique una lista con el orden de los campos según aparecen en el archivo externo. Esta lista se coloca entre paréntesis, al final de la instrucción **load data.** Ejemplo:

```
mysql>load data infile
  ->"C:\\mysql\data\prueba\texto.txt"
  ->into table personas
  ->(nombre, dni, apellido1, apellido2);
```

En numerosas instalaciones podría fallar el uso de **load data** por que MySQL podría no darnos permiso para su uso. Esa situación se puede arreglar entrando en el monitor añadiendo la opción **--local-infile=1**. Lo cual significa que habilitamos la posibilidad de importar datos. Ejemplo (de llamada al monitor habilitando archivos externos):

```
mysql -u administrador -p -h localhost --local-infile=1
```

A la hora de importar los datos se pueden reemplazar los valores duplicados adelantando la palabra **REPLACE** a la palabra INTO.

El texto **FIELDS TERMINATED BY** seguido de un carácter entrecomillado indica qué signo delimita cada campo, el texto **ENCLOSED BY** seguido de otro carácter, indica qué signo envuelve a cada campo. Ejemplo:

```
mysql>load data infile "/ejemplos/datos.txt" into table
personas fields terminated by ',' enclosed by '"';
```

También se puede utilizar el texto **lines terminated by** seguido del carácter de fin de línea (normalmente el fin de línea es "\r\n", pero habrá que tenerlo en cuenta si fallara la exportación:

```
mysql>load data infile "/ejemplos/datos.txt" into table
personas fields terminated by ',' enclosed by '"'
```

```
lines terminated by "\r\n";
```

Hay que tener en cuenta que el juego de caracteres entre la base de datos y MySQL deben de ser compatibles. El juego de caracteres de MySQL se modifica entrando en el servidor con la opción **--default-character-set**=*codigos*. Ejemplo:

```
mysql ----default-character-set=latin1
```

En el ejemplo se arranca el servidor con la opción de juego de caracteres Latin1 (europeo occidental). No vale la codificación de Windows, habrá que utilizar la codificación de MSDOS (sobre todo importando de Access a MySQL).

índices

creación

Para crear índices, se utiliza:

```
create index nombreÍndice ON tabla (lista_campos);
```

la lista de campos es una lista (separada por comas de los campos que forman la clave). Se puede indicar tras el nombre del índice, la palabra **UNIQUE** que indicaría que no se admiten valores duplicados en el índice (por lo que los campos que forman parte del índice no podrán repetir valores).

Desde la versión 3.23 de MySQL es más recomendable la sintaxis:

```
alter table tabla add index (lista_campos)
```

Ejemplo:

```
alter table clientes add index (apellidos, nombre);
```

La instrucción anterior crea un índice sobre la lista de campos apellidos y nombre. Se le puede poner nombre al índice colocando el nombre antes del paréntesis de los campos. Ejemplo:

```
alter table clientes add index nombreCompleto(apellidos,
nombre);
```

En la lista de campos, tras cada campo se puede colocar la palabra **ASC** (orden ascendente, se toma por defecto) o **DESC** (orden descendente).

Se pueden crear índices únicos si coloca la palabra **unique** antes de la palabra index. Los índices únicos prohíben la repetición de valores en los campos que forman el índice:

```
alter table clientes add unique index
nombreCompleto(apellidos, nombre);
```

claves principales

Si se pone como nombre del índice la palabra **primary key**, entonces el índice crea la clave principal de la tabla:

```
alter table clientes add primary key (campos);
```

Si se desea eliminar el índice:

```
drop index (índice) ON tabla;
```

También (recomendada desde la versión 3.23):

```
alter table tabla drop index (índice);
alter table tabla drop primary key;
```

En la creación de una tabla se pueden crear también los índices. Ejemplo:

```
create table personas (
  dni char(10),  nombre varchar(25) not null,
  apellidos varchar(50) not null,
  dirección varchar(50),
  primary key  (dni),
  key datosCompletos (apellidos,nombre));
```

En el ejemplo, *datosCompletos* es el nombre del índice basado en los campos apellidos y nombre.

claves externas

Sólo funcionan correctamente si las tablas son **innoDB.** También se pueden crear claves secundarias. Las claves secundarias se crean para campos de una tabla relacionados con campos que forman índices en otras tablas (normalmente forman claves principales, es decir son los campos que permiten relacionar tablas). La creación de estas claves se puede hacer desde **create table.** Sintaxis:

```
create table tabla (
  lista y propiedades de campo e índices,
  constraint nombreDeClave
  foreign key (camposQueFormaClave)
  references  tabla (camposClaveDeLatabla));
```

La palabra **constraint** es opcional y permite indicar un nombre para la clave externa. **foreign key** indica los campos de esta tabla relacionados con campos de otra. **references** indica el nombre de la tabla relacionada y el nombre de los campos relacionados.

Se pueden indicar al final las palabras**:**

© Jorge Sánchez' 2003

- **on delete cascade.** Que hace que al borrar registros en la tabla principal, se borren registros en la relacionada.
- **on update cascade.** Que permite que al cambiar los datos de los campos en la tabla principal, se actualicen en la relacionada.
- **on delete set null.** Hace que si se borra un registro de la tabla principal, los valores de los campos relacionados se coloque a **null**
- **on update set null.** Al cambiar valores principales se dejan a **null** los relacionados.
- **on update restrict.** No permite modificar valores en la tabla principal si hay registros relacionados en otra tabla.
- **on delete restrict.** No permite eliminar valores en la tabla principal si hay registros relacionados en otra tabla.
- **on update no action**
- **on delete no action**

consultas

obtener registros. *select*

La instrucción **select** es la fundamental del lenguaje SQL y por tanto de MySQL. Esta instrucción permite realizar consultas sobre la base de datos. El formato básico de la instrucción es:

```
select ListaDecampos from tablas where condición;
```

seleccionar todos los datos

El campo especial "*" sirve para representar todos los campos de una tabla. Así la instrucción:

```
select * from personas;
```

muestra todos los campos de la tabla personas.

seleccionar campos concretos

Se puede indicar la lista exacta de campos que queremos que aparezcan en la consulta. Por ejemplo:

```
select nombre, apellido1, apellido2 from personas;
```

condiciones

El apartado **where** de la instrucción **select** nos permite poner una condición de modo que sólo aparezcan en la consulta los registros que cumplan la condición. Se pueden utilizar operadores de comparación. Ejemplos:

```
select nombre, apellido1, apellido2 from personas
  where edad=25;
select nombre, apellido1, apellido2 from personas
  where edad>25;
```

También se puede realizar consultas con el operador OR o AND:

```
select nombre, apellido1, apellido2 from personas
  where (edad>25 AND edad<50);
select nombre, apellido1, apellido2 from personas
  where (nombre="Luis" OR nombre="Pedro");
```

La primera consulta obtiene nombre y apellidos de las personas cuya edad esté entre 26 y 49 años. La segunda saca a todos los luises y pedros.

También se pueden usar patrones mediante la cláusula **like**. Por ejemplo:

```
select * from personas where apellido1 like "g%";
```

Sacaría las personas cuyo primer apellido empiece por "g". Es decir el símbolo "%" hace de comodín. Otras expresiones posibles para **like** son:

expresión *like*	significado
"g%"	Que empiece por **g**
"%g"	Que termine por **g**
"%g%"	Que tenga una **g**
"_____"	Que tenga cinco caracteres

Esto se puede extender de forma más poderosa utilizando **regexp** en lugar de **like**. **regexp** permite utilizar expresiones regulares. Algunas posibilidades son:

expresión regular	significado
"."	Cualquier carácter, pero sólo uno
"[xyz**]"**	El carácter **x**, el **y** o el **z**
"[x-z**]**	Igual que el anterior
"[0-9**]"**	Cualquier número
"x*"	Una o más equis
".*"	Cualquier número de caracteres
"^b"	Que empiece por **b**
"b$"	Que termine por **b**
"[69].*"	Que empiece por 6 o por 9
"^[69]"	Que empiece por 6 o por 9
"^.....$"	Que tenga exactamente cinco caracteres
"^.{5}$"	Que tenga exactamente cinco caracteres

ordenar

La cláusula **order by** sirve para ordenar en base a una o más columnas. Ejemplo:

```
select * from personas order by apellido1, apellido2,
nombre;
```

Normalmente la ordenación se realiza en ascendente (de la A a la Z o de menor a mayor en el caso de los números), pero se puede colocar la palabra clave **desc** tras el nombre del campo por el que se ordena, para indicar que ese campo irá en descendente:

```
select * from personas order by edad desc;
```

consultas de campos calculados

Esto permite realizar cálculos con las columnas de consulta. El resultado de cada cálculo se coloca en una nueva columna de la tabla. Para los cálculos se pueden utilizar operadores aritméticos y funciones. Ejemplo:

```
select precio, precio * 0.16 from articulos;
```

El resultado sería:

```
+--------+-------------+
| precio | precio*0.16 |
+--------+-------------+
|   1.50 |        0.24 |
|   2.00 |        0.32 |
|  10.00 |        1.60 |
+--------+-------------+
3 rows in set (0.01 sec)
```

A la columna (o columnas) de cálculo se le puede poner nombre haciendo uso de **as**. Ejemplo:

```
select precio, precio * 0.16 as iva from articulos;
+--------+-------------+
| precio |     iva     |
+--------+-------------+
|   1.50 |        0.24 |
|   2.00 |        0.32 |
|  10.00 |        1.60 |
+--------+-------------+
3 rows in set (0.01 sec)
```

Se pueden usar expresiones que usen funciones internas. Ejemplo:

```
select concat(nombre," ",apellidos) from personas;
```

consultas de totales

Se pueden agrupar los resultados según uno o más campos. Eso se realiza mediante el operador **group by**. Ejemplo;

```
select provincia from localidades group by provincia;
```

El ejemplo anterior, enseña las provincias presentes en la tabla de localidades. Si no hubiera apartado **group by** también saldrían las provincias, pero cada provincia saldría tantas veces como localidades incluya.

La mayor ventaja que ofrecen estas consultas es que se pueden hacer cálculos sobre los grupos:

```
select provincia, count(*) from localidades
  group by provincia;
```

En este caso aparece una segunda columna que contará los registros de cada grupo (es decir las localidades de cada provincia). Otros operadores son **sum**, **max** y **min.** Ejemplo:

```
select provincia, sum(habitantes) from localidades
  group by provincia;
```

Suma los habitantes de cada localidad por cada provincia (es decir calcula los habitantes de cada provincia).

Se pueden usar también las funciones **max** (máximo), **min** (mínimo) o **avg** (media).

uso de consultas para almacenar valores a archivos externos

La potente instrucción **select** admite la posibilidad de ser utilizada para almacenar valores en archivos externos. Para ello se utiliza la palabra clave **into outfile** seguida de la ruta al archivo externo. A esta palabra le siguen los modificadores de terminación de campos y líneas ya vistos en la instrucción **load data**. Ejemplo:

```
select nombre, apellido1, apellido2 into outfile
  "exportacion.txt" fields terminated by ',' enclosed by '"'
  lines terminated by '\r\n'
  from clientes;
```

update

Permite modificar campos. Su sintaxis básica es:

```
update tabla set columna1=valor1, columna2=valor2,... where
condición;
```

Ejemplos:

```
update artículos set iva=0.12;
update personas set nacionalidad="estados unidos" where
nacionalidad="USA";
update personas set edad=edad+1;
update artículos set precio=precio*1,16,
  descuento=descuento/2;
```

delete

Permite borrar registros de las tablas. Su sintaxis básica es:

```
delete from tabla where condición;
```

Ejemplos:

```
delete from clientes where deudor='y';
```

uniones

Se trata de consultas realizadas sobre datos de varias tablas. para ello esas tablas deben estar relacionadas por al menos un campo. Ejemplo clásico:

```
select nombre, apellidos, fecha_alquiler from cliente,
alquiler where cliente.dni=alquiler.dni;
```

A veces el nombre de los campos es ambiguo (porque el mismo nombre se emplea es más de una de las tablas implicadas en la consulta) y entonces se debe indicar la tabla junto al nombre del campo, separados por un punto:

```
select cliente.nombre, cliente.apellidos, fecha_alquiler
from cliente, alquiler where cliente.dni=alquiler.dni;
```

Se puede poner un alias a las tablas con la palabra **as:**

```
select c.nombre, c.apellidos, fecha_alquiler
from cliente as c, alquiler where cliente.dni=alquiler.dni;
```

Eso muestra las fechas de cada alquiler junto con nombre y apellidos del cliente que alquiló. Para ello ambas tablas deben estar relacionadas por el DNI. Esta misma consulta se puede hacer en el formato SQL ANSI-92 (totalmente soportado por MySQL) de esta forma:

```
select nombre, apellidos, fecha_alquiler from clientes join
alquiler on clientes.dni=alquiler.dni;
```

Es más recomendable esta segunda forma ya que permite realizar asociaciones avanzadas. De hecho es posible usar estas formas de unión en el apartado **join**

- **cross join.** Producto cruzado. Combina cada registro de la primera tabla con cada registro de la tabla relacionada.
- **inner join.** Unión normal**.** Muestra sólo registros de ambas tablas que estén relacionados.
- **left join.** Muestra todos los registros de la primera tabla y sólo los registros relacionados en la segunda.
- **right join.** Muestra todos los registros de la segunda tabla y sólo los registros relacionados en la primera.