Bases de datos en MySQL

Luis Alberto Casillas Santillán
Marc Gibert Ginestà
Óscar Pérez Mora

P06/M2109/02151

UOC
www.uoc.edu

Índice

Introducción

MySQL es un sistema gestor de bases de datos (SGBD, DBMS por sus siglas en inglés) muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales (no menos importantes) contar con un alto grado de estabilidad y un rápido desarrollo.

Nota

Podremos utilizar la licencia GPL de MySQL siempre que el programa que lo use también lo sea, en caso contrario se debe adquirir la "licencia comercial", entre 250 y 500 €, en el momento de escribir este material.

MySQL está disponible para múltiples plataformas, la seleccionada para los ejemplos de este libro es GNU/Linux. Sin embargo, las diferencias con cualquier otra plataforma son prácticamente nulas, ya que la herramienta utilizada en este caso es el cliente *mysql-client,* que permite interactuar con un servidor MySQL (local o remoto) en modo texto. De este modo es posible realizar todos los ejercicios sobre un servidor instalado localmente o, a través de Internet, sobre un servidor remoto.

Para la realización de todas las actividades, es imprescindible que dispongamos de los datos de acceso del usuario administrador de la base de datos. Aunque en algunos de ellos los privilegios necesarios serán menores, para los capítulos que tratan la administración del SGBD será imprescindible disponer de las credenciales de administrador.

Nota

Las sentencias o comandos escritos por el usuario estarán en `fuente monoespaciada`, y las palabras que tienen un significado especial en MySQL estarán en **negrita**. Es importante hacer notar que estas últimas no siempre son palabras reservadas, sino comandos o sentencias de *mysql-client*.

La versión de MySQL que se ha utilizado durante la redacción de este material, y en los ejemplos, es la 4.1, la última versión estable en ese momento, aunque no habrá ningún problema en ejecutarlos en versiones anteriores, hasta la 3.23.

Objetivos

Adquirir las habilidades y conocimientos de MySQL necesarios para utilizar y administrar este SGBD (sistema gestor de bases de datos).

1. Características de MySQL

En este apartado enumeraremos las prestaciones que caracterizan a este SGBD, así como las deficiencias de diseño, limitaciones o partes del estándar aún no implementadas.

1.1. Prestaciones

MySQL es un SGBD que ha ganado popularidad por una serie de atractivas características:

- Está desarrollado en C/C++.
- Se distribuyen ejecutables para cerca de diecinueve plataformas diferentes.
- La API se encuentra disponible en C, C++, Eiffel , Java, Perl, PHP, Python, Ruby y TCL.
- Está optimizado para equipos de múltiples procesadores.
- Es muy destacable su velocidad de respuesta.
- Se puede utilizar como cliente-servidor o incrustado en aplicaciones.
- Cuenta con un rico conjunto de tipos de datos.
- Soporta múltiples métodos de almacenamiento de las tablas, con prestaciones y rendimiento diferentes para poder optimizar el SGBD a cada caso concreto.
- Su administración se basa en usuarios y privilegios.
- Se tiene constancia de casos en los que maneja cincuenta millones de registros, sesenta mil tablas y cinco millones de columnas.
- Sus opciones de conectividad abarcan TCP/IP, *sockets* UNIX y *sockets* NT, además de soportar completamente ODBC.
- Los mensajes de error pueden estar en español y hacer ordenaciones correctas con palabras acentuadas o con la letra 'ñ'.
- Es altamente confiable en cuanto a estabilidad se refiere.

Para todos aquellos que son adeptos a la filosofía de UNIX y del lenguaje C/C++, el uso de MySQL les será muy familiar, ya que su diseño y sus interfaces son acordes a esa filosofía: "crear herramientas que hagan una sola cosa y que la hagan bien". MySQL tiene como principal objetivo ser una base de datos fiable y eficiente. Ninguna característica es implementada en MySQL si antes no se tiene la certeza que funcionará con la mejor velocidad de respuesta y, por supuesto, sin causar problemas de estabilidad.

La influencia de C/C++ y UNIX se puede observar de igual manera en su sintaxis. Por ejemplo, la utilización de expresiones regulares, la diferenciación de funciones por los paréntesis, los valores lógicos como 0 y 1, la utilización del tabulador para completar sentencias, por mencionar algunos.

1.2. Limitaciones

Al comprender sus principios de diseño, se puede explicar mejor las razones de algunas de sus carencias. Por ejemplo, el soporte de transacciones o la integridad referencial (la gestión de claves foráneas) en MySQL está condicionado a un esquema de almacenamiento de tabla concreto, de forma que si el usuario no va a usar transacciones, puede usar el esquema de almacenamiento "tradicional" (MyISAM) y obtendrá mayor rendimiento, mientras que si su aplicación requiere transacciones, deberá usar el esquema que lo permite (InnoDB), sin ninguna otra restricción o implicación.

Nota

El esquema de tabla que hay que usar se decide para cada una en el momento de su creación, aunque puede cambiarse posteriormente. Actualmente, MySQL soporta varios esquemas y permite la incorporación de esquemas definidos por el usuario.

Otras limitaciones son las siguientes:

- No soporta procedimientos almacenados (se incluirán en la próxima versión 5.0).
- No incluye disparadores (se incluirán en la próxima versión 5.0).
- No incluye vistas (se incluirán en la próxima versión 5.0).
- No incluye características de objetos como tipos de datos estructurados definidos por el usuario, herencia etc.

2. Acceso a un servidor MySQL

En este apartado veremos las distintas formas de acceso a un servidor MySQL existente que nos proporciona el propio SGBD. El acceso desde lenguajes de programación o herramientas en modo gráfico se tratará en otros apartados.

2.1. Conectándose con el servidor

Para conectarse con el servidor deberemos asegurarnos de que éste está funcionando y de que admite conexiones, sean éstas locales (el SGBD se está ejecutando en la misma máquina que intenta la conexión) o remotas.

Adicionalmente, deberemos disponer de las credenciales necesarias para la conexión. Distintos tipos de credenciales nos permitirán distintos niveles de acceso. Para simplificar, supondremos que disponemos de las credenciales (usuario y contraseña) del administrador de la base de datos (normalmente, usuario *root* y su contraseña). En el apartado que concierne a la administración de MySQL, se comenta detalladamente los aspectos relacionados con el sistema de usuarios, contraseñas y privilegios del SGBD.

2.1.1. Servidores y clientes

El servidor MySQL es el servicio *mysqld*, que puede recibir solicitudes de clientes locales o remotos a través TCP/IP, *sockets* o *pipes* en forma de ficheros locales a la máquina en que se está ejecutando. En la distribución se incluye un cliente llamado *mysql-client*, al que en adelante nos referiremos simplemente como *mysql* (así es como se llama el programa ejecutable). Si se invoca sin parámetros, *mysql* realiza una conexión al servidor local utilizando el nombre del usuario UNIX que lo ha invocado, y supone que este usuario no requiere contraseña. La conexión a un servidor remoto y un nombre de usuario específicos requiere de al menos dos argumentos:

Nota

El servidor MySQL es *mysqld*. A él se pueden conectar múltiples clientes. *mysql* es el cliente en modo texto que proporciona el propio SGBD.

- -h para especificar el nombre del servidor.
- -u para el nombre del usuario.

Para que el programa cliente pregunte la contraseña de conexión al usuario, deberemos proporcionar adicionalmente el parámetro -p.

```
$ mysql -h servidor.misitio.org -u <usuario> -p
```

2.1.2. Conectarse y desconectarse

Si se tiene algún problema para realizar la conexión, es necesario consultar con el administrador del sistema, que nos proporcionará un nombre de usuario, contraseña y el nombre del servidor, según sea necesario, y nos informará de las restricciones que tiene nuestra cuenta.

Nota

Los usuarios del sistema operativo y los de MySQL no son los mismos, aunque el administrador de MySQL (con fines prácticos) pueda utilizar los mismos nombres para las cuentas de los usuarios MySQL.

La administración y seguridad de MySQL está diseñada sobre un esquema de usuarios y privilegios. Los usuarios deben ser creados por el administrador con sus respectivos privilegios y restricciones. Es el administrador quien decide si los nombres de los usuarios de MySQL se corresponden o no a los del sistema operativo.

Apariencia de *mysql* al ingresar en el modo interactivo:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.49-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Con el comando *help* obtenemos una serie de opciones (veremos las más utilizadas).

Para salir del cliente podemos escribir '*\q*' o '*quit*':

```
mysql> quit;
```

Tanto para el comando *quit* como para el comando *help*, el punto y coma al final es opcional.

2.2. Introducción de sentencias

El cliente de MySQL en modo interactivo nos permite tanto la introducción de sentencias SQL para trabajar con la base de datos (crear tablas, hacer consultas y ver sus resultados, etc.) como la ejecución de comandos propios del SGBD para obtener información sobre las tablas, índices, etc. o ejecutar operaciones de administración.

Sentencias

Las sentencias en *mysql* pueden abarcar múltiples líneas y terminan con punto y coma.

2.2.1. Sentencias

A continuación presentamos una ejecución de la sentencia *select* con cuatro columnas de datos:

```
mysql> select user(), connection_id(), version(), database();
+--------------+-----------------+-------------+------------+
|   user()     | CONNECTION_ID() | version()   | database()  |
+--------------+-----------------+-------------+------------+
| yo@localhost |    4            | 3.23.49-log |            |
+--------------+-----------------+-------------+------------+
1 row in set (0.00 sec)
mysql>
```

En esta consulta se solicita, a través de funciones incorporadas en el SGBD, el nombre del usuario actual de MySQL, el número de conexión al servidor, la versión del servidor y la base de datos en uso. Las funciones se reconocen por los paréntesis al final. *mysql* entrega sus resultados en tablas, en la que el primer renglón son los encabezados de las columnas. Es importante no dejar espacio entre el nombre de una función y los paréntesis, de otro modo, *mysql* marcará un mensaje de error.

La última línea entregada por *mysql* informa sobre el número de filas encontrado como resultado de la consulta y el tiempo estimado que llevó su realización. Esta medida de tiempo no se debe considerar muy precisa para medir el rendimiento del servidor, se trata simplemente de un valor aproximado que puede verse alterado por múltiples factores.

Observamos que la columna con el nombre de la base de datos actual esta vacía. Esto es natural, ya que no hemos creado aún ninguna base de datos ni le hemos indicado al gestor sobre cuál queremos trabajar.

2.2.2. Comandos en múltiples líneas

Los comandos pueden expandirse en varias líneas por comodidad, sobre todo al escribir largas sentencias SQL. El cliente no enviará la sentencia SQL al servidor hasta encontrar el punto y coma, de este modo, el comando anterior puede escribirse así:

```
mysql> select user(),
-> connection_id(),
-> version(),
-> database();
+--------------+-----------------+-------------+------------+
|   user()     | CONNECTION_ID() | version()   | database()  |
+--------------+-----------------+-------------+------------+
| yo@localhost |    4            | 3.23.49-log |            |
+--------------+-----------------+-------------+------------+
1 row in set (0.00 sec)
mysql>
```

Obsérvese el indicador de *mysql* que se transforma en `->`, signo que significa que el comando aún no está completo. También pueden escribirse varios comandos en una sola línea, cada uno debe llevar su respectivo punto y coma:

```
mysql> select now(); select user();
+----------------------+
| CONNECTION_ID()      |
+----------------------+
|    4                 |
+----------------------+
1 row in set (0.00 sec)
+----------------------+
|   user()             |
+----------------------+
|  yo@localhost        |
+----------------------+
1 row in set (0.01 sec)
mysql>
```

Se ejecutarán en el orden que están escritos. Los comandos se pueden cancelar con la combinación `\c`, con lo que el cliente nos volverá a mostrar el indicador para que escribamos de nuevo la sentencia.

```
mysql> select now(),
-> uso
-> ver \c
mysql>
```

Indicadores de *mysql*

Indicador	Significado
mysql>	Espera una nueva sentencia
->	La sentencia aún no se ha terminado con ;
">	Una cadena en comillas dobles no se ha cerrado
'>	Una cadena en comillas simples no se ha cerrado

2.2.3. Cadenas de caracteres

Las cadenas de caracteres pueden delimitarse mediante comillas dobles o simples. Evidentemente, deben cerrarse con el mismo delimitador con el que se han abierto.

```
mysql> select "Hola mundo",'Felicidades';
```

y pueden escribirse en diversas líneas:

```
mysql> select "Éste es un texto
       "> en dos renglones";
```

Al principio, es común olvidar el punto y coma al introducir un comando y, también, olvidar cerrar las comillas. Si éste es el caso, hay que recordar que *mysql* no interpreta lo que está entre comillas, de tal modo que para utilizar el comando de cancelación '\c' es preciso antes cerrar las comillas abiertas:

```
mysql> select "Éste es un texto
"> \c
"> " \c
mysql>
```

2.2.4. Expresiones y variables

MySQL dispone de variables de sesión, visibles únicamente durante la conexión actual. Éstas pueden almacenar valores de tipos enteros, flotantes o cadenas, pero no tablas. Se definen como en el siguiente ejemplo:

```
mysql> select @x := 1;
```

La variable local @x tiene ahora el valor 1 y puede utilizarse en expresiones:

```
mysql> select @x, sqrt(@x), sin(@x), @x + 10, @x > 10;
+------+----------+----------+---------+---------+
| @x   | sqrt(@x) | sin(@x)  | @x + 10 | @x > 10 |
+------+----------+----------+---------+---------+
| 1    | 1.000000 | 0.841471 |  11     |  0      |
+------+----------+----------+---------+---------+
1 row in set (0.01 sec)
```

Las variables locales permiten almacenar datos entre consultas y, en la práctica, es recomendable utilizarlas exclusivamente con este fin, por ejemplo:

```
mysql> select @hora_ingreso := now();
mysql> mysql> select now() - @ingreso;
+-------------------+
| now() - @ingreso  |
+-------------------+
|  20040124138051   |
+-------------------+
1 row in set (0.00 sec)
```

2.2.5. Expresiones

Hay que tener cuidado con el uso de las variables locales por los motivos siguientes:

- Se evalúan en el servidor al ser enviadas por el cliente.
- Se realizan conversiones de tipo implícitas.

```
mysql> do @ingreso := now();
```

Nota

El comando **do** evalúa expresiones sin mostrar los resultados en pantalla. Se puede evaluar cualquier expresión que admite el comando **select**.

Las variables no requieren declaración y, por omisión, contienen el valor NULL que significa "ausencia de valor", observad en la siguiente consulta los resultados de utilizar valores nulos:

```
mysql> select @y,
-> sqrt( @y ),
-> @y + 10,
-> @y < 1 ;
+------+------------+---------+--------+
| @y   | sqrt( @y ) | @y + 10 | @y < 1 |
+------+------------+---------+--------+
| NULL |  NULL      |  NULL   | NULL   |
+------+------------+---------+--------+
1 row in set (0.00 sec)
```

La razón de este comportamiento es que no es posible realizar ninguna operación cuando se desconoce algún valor. La entrada de valores NULL siempre significará salida de valores NULL.

Nota

En una expresión donde cualquiera de sus elementos sea NULL, automáticamente entregará como resultado el valor NULL.

2.3. Proceso por lotes

MySQL puede procesar por lotes las sentencias contenidas en un archivo de texto. Cada sentencia deberá terminar en ';' igual que si la escribiéramos en el cliente. La sintaxis es la siguiente:

```
$ mysql -u juan -h servidor.misitio.org -p < demo.sql
```

En este caso, se realizará una conexión con el servidor, nos pedirá la contraseña del usuario 'juan' y, si ésta es correcta, ejecutará los comandos incluidos en el archivo *demo.sql*, uno a uno y por el mismo orden. Imprimirá los resultados (o errores) en la salida estándar (o de error) y terminará. De este modo evitaremos la molestia de procesarlos uno por uno de forma interactiva.

Otra forma de procesar un archivo es mediante el comando `source` desde el indicador interactivo de MySQL:

```
mysql> source demo.sql
```

El archivo *demo.sql* crea una nueva base de datos.

El usuario debe tener permisos para crear bases de datos si quiere que sea procesado el archivo *demo.sql*. Si el administrador crea la base de datos por nosotros, será necesario editarlo, comentando la línea donde se crea la base de datos con el símbolo '#' al inicio:

```
# create database demo;
```

Es necesario procesar el contenido del fichero *demo.sql* tal como los transcribimos aquí, con el fin de poder realizar los ejemplos del resto del apartado. Si se observa su contenido, posiblemente muchas cosas se expliquen por sí mismas, de cualquier manera, serán explicadas en este apartado. También pueden ejecutarse sus órdenes en el cliente directamente.

Contenido del fichero demo.sql

```
#drop database demo;
create database demo;
use demo;
---
--- Estructura de la tabla productos
---

create table productos (
 parte    varchar(20),
 tipo     varchar(20) ,
 especificación varchar(20) ,
 psugerido   float(6,2),
 clave int(3)  zerofill not null auto_increment,
 primary key (clave)
);
insert into productos (parte,tipo,especificación,psugerido) values
    ('Procesador','2 GHz','32 bits',null),
    ('Procesador','2.4 GHz','32 bits',35),
    ('Procesador','1.7 GHz','64 bits',205),
    ('Procesador','3 GHz','64 bits',560),
    ('RAM','128MB','333 MHz',10),
    ('RAM','256MB','400 MHz',35),
    ('Disco Duro','80 GB','7200 rpm',60),
```

```
    ('Disco Duro','120 GB','7200 rpm',78),
    ('Disco Duro','200 GB','7200 rpm',110),
    ('Disco Duro','40 GB','4200 rpm',null),
    ('Monitor','1024x876','75 Hz',80),
    ('Monitor','1024x876','60 Hz',67)
    ;
--
-- Estructura de la tabla 'proveedor'
--
create table proveedores (
 empresa   varchar(20) not null,
 pago    set('crédito','efectivo'),
 primary key (empresa)
);
--
-- Valores de la tabla 'proveedor'
--
insert into proveedores (empresa,pago) values
   ('Tecno-k','crédito'),
   ('Patito','efectivo'),
   ('Nacional','crédito,efectivo')
;
create table ganancia(
  venta  enum('Por mayor','Por menor'),
   factor  decimal(2,2)
);
insert into ganancia values
  ('Por mayor',1.05),
    ('Por menor',1.12)
;
create table precios (
   empresa  varchar(20) not null,
   clave  int(3) zerofill not null,
   precio  float(6,2),

   foreign key (empresa)  references proveedores,
   foreign key (clave)   references productos
);

insert into precios values
   ('Nacional',001,30.82),
   ('Nacional',002,32.73),
   ('Nacional',003,202.25),
   ('Nacional',005,9.76),
   ('Nacional',006,31.52),
   ('Nacional',007,58.41),
   ('Nacional',010,64.38),
   ('Patito',001,30.40),
   ('Patito',002,33.63),
   ('Patito',003,195.59),
   ('Patito',005,9.78),
   ('Patito',006,32.44),
```

```
   ('Patito',007,59.99),
   ('Patito',010,62.02),
   ('Tecno-k',003,198.34),
   ('Tecno-k',005,9.27),
   ('Tecno-k',006,34.85),
   ('Tecno-k',007,59.95),
   ('Tecno-k',010,61.22),
   ('Tecno-k',012,62.29)
;
```

Si se desea llevar un registro de todas las operaciones de una sesión, se puede utilizar la expresión siguiente; de este modo se guardarán todos los comandos y sus resultados en *archivo_registro.txt*:

```
mysql> tee archivo_registro.txt
```

Para cancelar la captura, basta con teclear lo siguiente:

```
mysql> notee
```

2.4. Usar bases de datos

La siguiente consulta informa sobre la base de datos actualmente en uso.

```
mysql> select database();
+------------+
| database() |
+------------+
|            |
+------------+
1 row in set (0.13 sec)
```

El campo esta vacío porque no estamos haciendo uso de ninguna base de datos. Para ver las bases de datos existentes en el sistema, se debe efectuar la siguiente consulta:

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

MySQL nos muestra el listado de las bases de datos definidas en el servidor. Debe aparecer la base de datos *demo* que creamos con el archivo *demo.sql*. Para poder trabajar con ella, tenemos que abrirla:

```
mysql> use demo;
```

use

El comando *use base_de_datos* permite abrir una base de datos para su uso.

Nota

Es posible realizar consultas en una base de datos sin utilizar el comando **use**, en ese caso, todos los nombres de las tablas deben llevar el nombre de la base de datos a que pertenecen de la forma: *demo.productos*.

Otra posibilidad consiste en proporcionar el nombre de la base de datos al iniciar una sesión interactiva con *mysql:*

```
$ mysql demo -u juan -p
```

La consulta de las tablas que contiene la base de datos `demo` se realiza con la sentencia `show` de la siguiente manera:

```
mysql> show tables;
+----------------+
| Tables_in_demo |
+----------------+
|   partes       |
|  proveedores   |
+----------------+
2 rows in set (0.00 sec)
```

Nota

El comando *show* es útil para mostrar información sobre las bases de datos, tablas, variables y otra información sobre el SGBD. Podemos utilizar `help show` en el intérprete de comandos para obtener todas las variantes de esta sentencia.

Asimismo, podemos consultar las columnas de cada una de las tablas:

```
mysql> describe productos;
+---------------+----------------------+------+------+----------+-------------------+
| Field         |      Type            | Null | Key  | Default  | Extra             |
+---------------+----------------------+------+------+----------+-------------------+
| parte         | varchar(20)          | YES  |      | NULL     |                   |
| tipo          | varchar(20)          | YES  |      | NULL     |                   |
| especificicación | varchar(20)       | YES  |      | NULL     |                   |
| Field         | float(6,2)           | YES  |      | NULL     |                   |
| Field         | int(3) unsigned zerofill | YES  | PRI  | NULL     | auto_increment    |
+---------------+----------------------+------+------+----------+-------------------+
5 rows in set (0.00 sec)
```

Para crear una nueva base de datos usaremos la sentencia `create database`:

```
mysql> create database prueba;
```

Para eliminar una base de datos, usaremos la sentencia `drop database`:

```
mysql> drop database prueba;
```

MySQL es sensible al uso de mayúsculas y minúsculas, tanto en la definición de bases de datos, como de tablas o columnas.

3. Creación y manipulación de tablas

3.1. Crear tablas

Una vez realizada la conexión con el servidor MySQL y después de abrir una base de datos, podemos crear tablas en ella de la siguiente manera:

```
mysql> create table personas (
-> nombre char(30),
-> dirección char(40),
-> teléfono char(15)
-> );
Query OK, 0 rows affected (0.02 sec)
```

En este caso, la sentencia `create table` construye una nueva tabla en la base de datos en uso. La tabla contiene tres columnas, *nombre*, *dirección* y *teléfono*, todas de tipo carácter y de longitudes 30, 40 y 15 respectivamente. Si se intenta guardar en ellas valores que sobrepasen esos límites, serán truncados para poderlos almacenar. Por ese motivo, es importante reservar espacio suficiente para cada columna. Si se prevé que muchos registros ocuparán sólo una fracción del espacio reservado, se puede utilizar el tipo **varchar**, similar a **char**, con la diferencia de que el valor ocupará un espacio menor al especificado si la cadena es más corta que el máximo indicado, ahorrando así espacio de almacenamiento.

Los nombres de las columnas admiten caracteres acentuados. 

Las tablas pueden eliminarse con **drop table**:

```
mysql> drop table personas;
Query OK, 0 rows affected (0.01 sec)
```

Alternativamente, se puede utilizar la sintaxis siguiente:

```
mysql> drop table if exists personas;
```

Atributos de columna

Atributo	Significado
null	Se permiten valores nulos, atributo por omisión si no se especifica lo contrario.
not null	No se permiten valores nulos.
default valor	Valor por omisión que se asigna a la columna.
auto_increment	El valor se asigna automáticamente incrementando en uno el máximo valor registrado hasta ahora. Se aplica sólo a las columnas marcadas como clave primaria.
primary key	Señala al campo como clave primaria, implícitamente también lo declara como **not null**.

Veámoslo con un ejemplo:

```
mysql> create table personas (
-> nombre varchar(40) not null,
-> dirección varchar(50) null,
-> edo_civil char(13) default 'Soltero',
-> num_registro int primary key auto_increment,
-> );
Query OK, 0 rows affected (0.01 sec)
```

En este caso la tabla contiene cuatro columnas, de las cuales *nombre* y *edo_civil* permiten valores nulos, en *edo_civil* está implícito al no declarar lo contrario. La columna *num_registro* no acepta valores nulos porque está definida como clave primaria.

Nota

La definición de columnas tiene el siguiente formato:

nombre_columna **tipo atributos**.

Aunque la creación de una clave primaria puede declararse como atributo de columna, es conveniente definirla como restricción de tabla, como se verá enseguida.

También es posible indicar restricciones sobre la tabla y no sobre columnas especificas:

```
mysql> create table personas (
-> nombre varchar(40) not null,
-> nacimiento date not null,
-> pareja varchar(40),
-> proveedor int not null,
->
-> primary key (nombre,nacimiento),
-> unique (pareja),
-> foreign key (proveedor) references proveedores
-> );
Query OK, 0 rows affected (0.01 sec)
```

Restricciones de tabla

Restricción	Significado
primary key	Define la o las columnas que servirán como clave primaria. Las columnas que forman parte de la clave primaria deben de ser **not null**.
unique	Define las columnas en las que no pueden duplicarse valores. Serán las claves candidatas del modelo relacional.
foreign key (*columna*) **references** ***tabla*** (*columna2*)	Define que los valores de *columna* se permitirán sólo si existen en *tabla(columna2).* Es decir, *columna* hace referencia a los registros de *tabla*, esto asegura que no se realicen referencias a registros que no existen.

Se definen tres restricciones sobre la tabla después de la definición de cuatro columnas:

- La primera restricción se refiere a la clave primaria, compuesta por las columnas *nombre* y *nacimiento*: no puede haber dos personas que se llamen igual y que hayan nacido en la misma fecha. La clave primaria permite identificar de manera unívoca cada registro de la tabla.

- La segunda restricción define que la pareja de una persona debe ser única: dos personas no pueden tener la misma pareja. Todo intento de insertar un nuevo registro donde el nombre de la pareja ya exista, será rechazado. Cuando se restringe una columna con **unique**, los valores **null** reciben un trato especial, pues se permiten múltiples valores nulos.

- La tercera restricción afecta a la columna *proveedor*, sólo puede tomar valores que existan en la clave primaria de la tabla *proveedores*.

Claves foráneas

Las restricciones de tabla **foreign key** no tienen efecto alguno en MySQL 4.0 y anteriores, ya que esta característica no está implementada. Se admite en la sintaxis por compatibilidad, ya que será implementada en una versión posterior. En la versión 4.1, está soportada si se utiliza el tipo de tabla InnoDB.

Las restricciones de tabla pueden definirse con un identificador útil para hacer referencias posteriores a la restricción:

```
mysql> create table personas (
-> nombre varchar(40) not null,
-> nacimiento date not null,
-> pareja varchar(40),
-> proveedor int not null,
->
-> constraint clave primary key (nombre,nacimiento),
-> constraint monogamo unique (pareja),
-> constraint trabaja_en foreign key (proveedor) references
proveedores
-> );
```

key / index

La definición de índices puede hacerse también en el momento de creación de la tabla, mediante la palabra clave key (o index), a la que deberemos proporcionar el nombre que vamos a asignar a esta clave y las columnas que la forman, entre paréntesis. Existen modificadores opcionales sobre el índice que nos permiten especificar si se trata de un índice único o múltiple (según puedan existir o no varios valores iguales del índice en la tabla).

En versiones recientes de MySQL existen otros tipos de índices (espaciales, de texto completo, etc.) para tipos de datos concretos y que ofrecen prestaciones adicionales.

3.2. Tipos de datos

MySQL cuenta con un rico conjunto de tipos de datos para las columnas, que es necesario conocer para elegir mejor cómo definir las tablas. Los tipos de datos se pueden clasificar en tres grupos:

- Numéricos.
- Cadenas de caracteres
- Fechas y horas

El valor **null** es un caso especial de dato, ya que al significar *ausencia de valor* se aplica a todos los tipos de columna. Los siguientes símbolos se utilizan en la definición y descripción de los tipos de datos en MySQL:

- **M** - El ancho de la columna en número de caracteres.
- **D** - Número de decimales que hay que mostrar.
- **L** - Longitud o tamaño real de una cadena.
- **[ ]** - Lo que se escriba entre ellos es opcional.

3.2.1. Tipos de datos numéricos

Los tipos de datos numéricos comprenden dos categorías, los enteros y los números con punto flotante.

Números enteros

La principal diferencia entre cada uno de los tipos de enteros es su tamaño, que va desde 1 byte de almacenamiento hasta los 8 bytes. Las columnas de tipo entero pueden recibir dos atributos adicionales, que deben especificarse inmediatamente después del nombre del tipo:

- **unsigned**. Indica que el entero no podrá almacenar valores negativos. Es responsabilidad del usuario verificar, en este caso, que los resultados de las restas no sean negativos, porque MySQL los convierte en positivos.

- **zerofill.** Indica que la columna, al ser mostrada, rellenará con ceros a la izquierda los espacios vacíos. Esto de acuerdo al valor especificado por **M** en la declaración del tipo. Una columna con el atributo **zerofill** es al mismo tiempo **unsigned** aunque no se especifique.

Ejemplo

```
create table números (
x int(4) zerofill not null,
y int(5) unsigned
);
```

El comando anterior crea una tabla con dos columnas. Ambas ocuparán un espacio de 4 bytes, pero al mostrarse, la columna *x* ocupará un espacio de 4 dígitos y la columna *y*, de 5.

Tanto **zerofill** como **unsigned** deben escribirse siempre antes que cualquier otro atributo de columna.

Tipos enteros

Tipo	Espacio de almacenamiento	Significado
tinyint[(M)]	1 byte	Entero muy pequeño
smallint[(M)]	2 bytes	Entero pequeño
mediumint[(M)]	3 bytes	Entero mediano
int[(M)]	4 bytes	Entero
bigint[(M)]	8 bytes	Entero grande

Números con punto flotante

MySQL cuenta con los tipos **float** y **double**, de 4 y 8 bytes de almacenamiento. Además incluye el tipo decimal, que se almacena como una cadena de caracteres y no en formato binario.

Números de punto flotante

Tipo	Espacio de almacenamiento	Significado
float	4 bytes	Simple precisión
double	8 bytes	Doble precisión
decimal	M + 2 bytes	Cadena de caracteres representando un número flotante

3.2.2. Cadenas de caracteres

Cadenas de caracteres

Tipo	Equivalente	Tamaño máximo	Espacio de almacenamiento
char[(M)]		M bytes	M bytes
varchar[(M)]		M bytes	L+1 bytes
tinytext	tinyblob	2^8–1 bytes	L+1 bytes
text	blob	2^{16}–1 bytes	L+2 bytes
mediumtext	mediumblob	2^{24}–1 bytes	L+3 bytes
longtext	longblob	2^{32}–1 bytes	L+4 bytes
enum('v1','v2',. . .)		65535 valores	1 o 2 bytes
set('v1','v2',. . .)		64 valores	1 a 8 bytes

Si observamos la tabla, vemos que el único tipo de dato que siempre utiliza el tamaño especificado por M es el tipo **char**. Por este motivo, se ofrece el tipo **varchar** que ocupa sólo el espacio requerido por el valor de la columna.

Ejemplo

```
create table persona(
comentario char(250),
recado varchar(250)
);
```

La columna *comentario* ocupará 250 bytes de espacio de almacenamiento, sin importar el valor almacenado. Por el contrario, la columna *recado* ocupará sólo el espacio necesario según el valor asignado; por ejemplo, la cadena *"Instalar MySQL"* tiene 14 bytes de longitud, y el campo *recado* ocuparía 15 bytes para almacenarla.

Los tipos text y blob son equivalentes, pero text respeta las mayúsculas, minúsculas y caracteres acentuados en la ordenación.

Ejemplo del uso de los tipos enumerados o enum

```
create table persona(
edo_civil enum('soltero','casado','viudo','divorciado')
);
```

La columna *edo_civil* de la tabla en la sentencia anterior, solo podrá almacenar los valores *'soltero'*, *'casado'*, *'viudo'*, *'divorciado'*, que son especificados por el tipo **enum**. La columna ocupará el espacio de un byte, ya que los valores enum son representados internamente por números.

3.2.3. Fechas y horas

Fechas y horas

Tipo	Espacio de almacenamiento	Rango
date	3 bytes	'1000-01-01' al '9999-12-31'
time	3 bytes	'-838:59:59' a '838:59:59'
datetime	8 bytes	'1000-01-01 00:00:00' a '9999-12-31 23:59:59'
timestamp[(M)]	4 bytes	19700101000000 al año 2037
year[(M)]	1 byte	1901 a 2155

3.3. Modificar tablas

3.3.1. Agregar y eliminar columnas

Alterar la estructura de una tabla es una tarea más frecuente de lo que uno puede imaginar en un principio. La sentencia ***alter table*** permite una amplia gama de formas de modificar una tabla. La siguiente sentencia nos recuerda un poco a la estructura de la sentencia ***create table***, en donde modificamos la tabla *personal* creada en la sección anterior.

```
mysql> alter table personal add (
-> mascota char(30) default 'perro',
-> pasatiempo char (20) not null
-> );
```

Nota

Siempre es posible consultar la estructura de una tabla con el comando ***describe*** *tabla*.

Después de ejecutar la sentencia anterior, aparecen dos nuevas columnas en la tabla. Si queremos agregar una sola columna, podemos usar la sintaxis siguiente:

```
mysql> alter table personal add capital int not null
-> after nom;
```

Este formato de **alter table** permite, además, insertar las columnas antes (**before**) o después (**after**) de una columna en cuestión.

Las columnas no deseadas pueden eliminarse con la opción **drop**.

```
mysql> alter table personal drop pasatiempo;
```

3.3.2. Modificar columnas

La modificación de una columna con la opción **modify** es parecida a volver a definirla.

```
mysql> alter table personal modify
-> mascota char (14) default 'gato';
```

Después de la sentencia anterior, los atributos y tipo de la columna han cambiado por los especificados. Lo que no se puede cambiar con esta sintaxis es el nombre de la columna. Para ello, se debe utilizar la opción **change:**

```
mysql> alter table personal change nom
-> nombre char(20);
```

La columna que se llamaba *nom* cambia a *nombre*.

Con el mismo comando **alter table** podemos incluso realizar la ordenación física de una tabla bajo una columna específica:

```
mysql> alter table personal order by nom;
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Nota

En general, una tabla no puede durar mucho tiempo con un `order` respecto a una columna, ya que las inserciones no se realizarán respetando el orden establecido. Solamente en tablas que no van a ser actualizadas es útil aplicar este comando.

Finalmente, podemos cambiar de nombre la tabla:

```
mysql> alter table personal rename gente;
```

rename table

El comando **rename table** *viejo_nombre* to *nuevo_nombre* es una forma alternativa de cambiar el nombre a una tabla.

3.4. Otras opciones

3.4.1. Copiar tablas

Aunque no existe un comando explícito para copiar tablas de una base de datos a otra, es posible utilizar el comando **rename table** para este propósito; basta con especificar la base de datos a la que pertenece una tabla:

```
mysql> rename table base_uno.tabla to base_dos.tabla;
```

También es posible crear una tabla nueva con el contenido de otra ya existente (copiando los datos):

```
mysql> create table nueva_tabla select * from otra_tabla;
```

La siguiente sentencia es equivalente, pero no copia los datos de la tabla origen:

```
mysql> create table nueva_tabla like otra_tabla;
```

3.4.2. Tablas temporales

MySQL permite la creación de tablas temporales, visibles exclusivamente en la sesión abierta, y guardar datos entre consultas. La creación de una tabla temporal sólo requiere la utilización de la palabra ***temporary*** en cualquier formato del comando ***create table***. La utilidad de las tablas temporales se limita a consultas complejas que deben generar resultados intermedios que debemos consultar (hacer 'join' con ellas) varias veces o en consultas separadas. Internamente, MySQL genera también tablas temporales para resolver determinadas consultas:

```
mysql> create temporary table nueva_tabla ...
```

4. Consultas

Como ya hemos explicado, las consultas sobre la base de datos se ejecutan mediante sentencias SELECT introducidas en el propio programa cliente y los resultados se presentan en forma de tabla.

4.1. La base de datos demo

En esta sección utilizaremos la base de datos *demo* que hemos creado con el comando source *demo.sql*. Así que, antes de estudiar las consultas en MySQL, revisaremos brevemente la estructura de esta base de datos, que consta de las siguientes tablas:

Podeís ver la creación de la base de datos *demo* en el apartado "Proceso por lotes" de esta misma unidad didáctica.

```
mysql> show tables;
+----------------+
| Tables_in_demo |
+----------------+
|  ganancia      |
|   precios      |
|  productos     |
|  proveedores   |
+----------------+
```

Las cuatro tablas representan, de manera ficticia, la base de datos de un distribuidor de equipos de procesamiento. Están diseñadas para servir de ejemplo a los casos presentados en este capítulo, por lo que no necesariamente serán útiles en la vida real.

En nuestro ejemplo imaginario representamos la siguiente situación.

- Nuestro vendedor tiene una relación de proveedores que venden sus productos a crédito, en efectivo o ambos. Las compras a crédito pagan intereses, pero son útiles porque no siempre es posible pagar en efectivo. Se utiliza una columna de tipo conjunto para *pago*, que puede tomar los valores *'crédito'*, *'efectivo'* o ambos:

```
create table proveedores (
empresa varchar(20) not null,
pago set('crédito','efectivo'),
primary key (empresa)
);
```

Los productos que se distribuyen son partes de equipo de cómputo. Para la mayoría de los productos en el mercado, los fabricantes sugieren un precio de venta al público que, aunque no es obligatorio, los consumidores no están dispuestos a pagar más. Las claves de los productos son asignadas para control

interno con un número consecutivo. Con estas especificaciones, la tabla *productos* se define de la manera siguiente:

```
create table productos (
parte varchar(20),
tipo varchar(20) ,
especificación varchar (20) ,
psugerido float(6,2),
clave int(3) zerofill not null auto_increment,
primary key (clave)
);
```

- La empresa define una política para las ganancias mínimas que se deben obtener en ventas: el 5% al por mayor y el 12% al por menor. Estos valores se almacenan en la tabla *ganancias*, donde se decidió incluir una columna de nombre *factor*, con el número por el que se multiplica el precio de compra para obtener el precio de venta. Los tipos de venta '*Por mayor*' y '*Por menor*' se definen con un tipo de datos **enum**:

```
create table ganancia(
venta enum('Por mayor','Por menor'),
factor decimal (2,2)
);
```

- La lista de precios se define a partir de la empresa proveedor y el producto, asignándole un precio. Por ese motivo, las columnas *empresa* y *clave* se definen como **foreign key**.

```
create table precios (
empresa varchar(20) not null,
clave int(3) zerofill not null,
precio float(6,2),
foreign key (empresa) references proveedores,
foreign key (clave) references productos
);
```

4.2. Consultar información

MySQL ofrece un conjunto muy amplio de funciones auxiliares (tanto estándares como propias) que nos pueden ayudar mucho en determinados momentos, dejando parte del trabajo de manipular los resultados al propio gestor. Debido al rápido ritmo en el desarrollo de este SGBD, es muy conveniente consultar siempre la documentación de nuestra versión para conocer sus posibilidades concretas.

En el módulo 3 de este curso ya estudiamos en detalle el Lenguaje SQL, por lo que no vamos a extendernos aquí en su uso y posibilidades. Únicamente mostraremos los aspectos destacables, facilidades o limitaciones que ofrece MySQL respecto a él.

4.2.1. Funciones auxiliares

Las funciones auxiliares que podemos utilizar en nuestras consultas (tanto en la proyección de las columnas como en condiciones en su selección) se pueden clasificar según el tipo de datos con el que trabajan.

Ejemplo

```
mysql> select concat(parte,' ',tipo) as producto,
-> psugerido as 'precio sugerido',
-> psugerido + 10 as precio_con_envio
-> from productos;
+--------------------+-----------------+------------------+
|   producto         | precio sugerido | precio_con_envio |
+--------------------+-----------------+------------------+
| Procesador 2 GHz   |    NULL         |    NULL          |
| Procesador 2.4 GHz |   35.00         |    45.00         |
| Procesador 1.7 GHz |   205.00        |   215.00         |
| Procesador 3 GHz   |   560.00        |   570.00         |
|     RAM 128MB      |   10.00         |    20.00         |
|     RAM 256MB      |   35.00         |    45.00         |
| Disco Duro 80 GB   |   60.00         |    70.00         |
| Disco Duro 120 GB  |   78.00         |    88.00         |
| Disco Duro 200 GB  |   110.00        |   120.00         |
| Disco Duro 40 GB   |    NULL         |    NULL          |
| Monitor 1024x876   |   80.00         |    90.00         |
| Monitor 1024x876   |   67.00         |    77.00         |
+--------------------+-----------------+------------------+
12 rows in set (0.00 sec)
```

Algunos ejemplos de las funciones más usadas:

Operadores lógicos

Comparación. Aparte de los estándares =, !=, <, >, IS NULL, IS NOT NULL, BETWEN, IN, destacan COALESCE, INTERVAL, LEAST, GREATEST para trabajar con listas de valores.

Control del flujo de ejecución

- CASE .. WHEN .. THEN .. ELSE .. END: Similar a la estructura que crearíamos mediante cualquier lenguaje de programación:

```
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
+-----------------------------------------+
| CASE WHEN 1>0 THEN 'true' ELSE 'false' END |
+-----------------------------------------+
| true                                    |
+-----------------------------------------+
1 row in set (0.00 sec)
```

- IF(expr1,expr2,expr3): Típica estructura condicional, si la expr1 es cierta, devuelve la expr2, en caso contrario, la expr3:

```
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
+---------------------------------------+
| IF(STRCMP('test','test1'),'no','yes') |
+---------------------------------------+
| no                                    |
+---------------------------------------+
1 row in set (0.00 sec)
```

Funciones para trabajar con cadenas de caracteres (sólo algunos ejemplos)

- CONCAT, INSTR (encontrar en una cadena), SUBSTRING, LCASE/RCASE, LENGTH, REPLACE, TRIM, entre otras, son funciones similares a las que podemos encontrar en lenguajes de programación para manipular cadenas de caracteres.

- QUOTE: delimita una cadena de texto correctamente para evitar problemas al usarla en sentencias SQL. La cadena resultante estará delimitada por comillas simples. Las comillas, el valor ASCII NUL y otros potencialmente conflictivos serán devueltos precedidos del carácter '\'.

- ENCODE/DECODE, CRYPT, COMPRESS/UNCOMPRESS, MD5, etc. son funciones que nos pueden ayudar mucho en el almacenamiento de datos sensibles como contraseñas, etc.

Funciones numéricas

- Los operadores aritméticos clásicos para realizar todo tipo de operaciones, suma, resta, división, producto, división entera, etc.

- Funciones matemáticas de todo tipo, trigonométricas, logarítmicas, etc.

Funciones para trabajar con fechas y horas

- Obtención de fechas en cualquier formato: DATE_FORMAT, DATE, NOW, CURRDATE, etc.

- Manipulación y cálculos con fechas: ADDDATE, ADDTIME, CONVERT_TZ, DATE_DIFF, etc.

4.2.2. La sentencia EXPLAIN

MySQL nos ofrece también facilidades a la hora de evaluar las sentencias SQL, gracias a la sentencia EXPLAIN.

Presentamos primero la ejecución de una sentencia SQL más o menos compleja:

```
mysql> select productos.clave, concat(parte,' ',tipo,' ', especificación) as producto, proveedores.em-
presa , precio , pago from productos natural join precios natural join proveedores;
+-------+----------------------------+----------+--------+------------------+
| clave | producto                   | empresa  | precio | pago             |
+-------+----------------------------+----------+--------+------------------+
|  003  | Procesador 1.7 GHz 64 bits | Tecno-k  | 198.34 | crédito          |
|  005  | RAM 128MB 333 MHz          | Tecno-k  | 9.27   | crédito          |
|  006  | RAM 256MB 400 MHz          | Tecno-k  | 34.85  | crédito          |
|  007  | Disco Duro 80 GB 7200 rpm  | Tecno-k  | 59.95  | crédito          |
|  010  | Disco Duro 40 GB 4200 rpm  | Tecno-k  | 61.22  | crédito          |
|  012  | Monitor 1024x876 60 Hz     | Tecno-k  | 62.29  | crédito          |
|  001  | Procesador 2 GHz 32 bits   | Patito   | 30.40  | efectivo         |
|  002  | Procesador 2.4 GHz 32 bits | Patito   | 33.63  | efectivo         |
|  003  | Procesador 1.7 GHz 64 bits | Patito   | 195.59 | efectivo         |
|  005  | RAM 128MB 333 MHz          | Patito   | 9.78   | efectivo         |
|  006  | RAM 256MB 400 MHz          | Patito   | 32.44  | efectivo         |
|  007  | Disco Duro 80 GB 7200 rpm  | Patito   | 59.99  | efectivo         |
|  010  | Disco Duro 40 GB 4200 rpm  | Patito   | 62.02  | efectivo         |
|  001  | Procesador 2 GHz 32 bits   | Nacional | 30.82  | crédito,efectivo |
|  002  | Procesador 2.4 GHz 32 bits | Nacional | 32.73  | crédito,efectivo |
|  003  | Procesador 1.7 GHz 64 bits | Nacional | 202.25 | crédito,efectivo |
|  005  | RAM 128MB 333 MHz          | Nacional | 9.76   | crédito,efectivo |
|  006  | RAM 256MB 400 MHz          | Nacional | 31.52  | crédito,efectivo |
|  007  | Disco Duro 80 GB 7200 rpm  | Nacional | 58.41  | crédito,efectivo |
|  010  | Disco Duro 40 GB 4200 rpm  | Nacional | 64.38  | crédito,efectivo |
+-------+----------------------------+----------+--------+------------------+
20 rows in set (0.00 sec)
```

Ahora utilizamos la sentencia EXPLAIN para que MySQL nos explique cómo ha realizado esta consulta:

mysql> explain select productos.clave, concat(parte,' ',tipo,' ', especificación) as producto, proveedores.empresa , precio , pago from productos natural join precios natural join proveedores;

```
+-------------+--------+---------------+---------+---------+---------------+------+------------+
| table       | type   | possible_keys | key     | key_len | ref           | rows | Extra      |
+-------------+--------+---------------+---------+---------+---------------+------+------------+
| precios     | ALL    | NULL          | NULL    |  NULL   | NULL          | 20   |            |
| productos   | eq_ref | PRIMARY       | PRIMARY |    4    | precios.clave|  1   |            |
| proveedores | ALL    | PRIMARY       | NULL    |  NULL   | NULL          | 3    | where used |
+-------------+--------+---------------+---------+---------+---------------+------+------------+
3 rows in set (0.00 sec)
```

En cada fila del resultado, nos explica cómo ha utilizado los índices de cada tabla involucrada en la consulta. La columna 'type' nos indica el tipo de “join” que ha podido hacer. En nuestro caso, 'eq_ref', 'ref' o 'ref_or_null' indica que se ha consultado una fila de esta tabla para cada combinación de filas de las otras. Es una buena señal, se están utilizando los índices, tal como indican el resto de columnas (en concreto el atributo 'clave' que es su clave primaria).

Vemos que en las otras dos tablas, el tipo de 'join' es ALL, esto indica que el gestor ha tenido que leer toda la tabla para comprobar las condiciones que le hemos exigido en la consulta. En el caso de la tabla proveedores, habría podido utilizar la clave primaria ('possible_keys'), pero no lo ha hecho.

Vamos a intentar mejorar esta consulta. Vemos que en la tabla precios no se ha definido ningún índice, lo que facilitaría la labor al SGBD:

```
mysql> alter table precios add index empresa_idx (empresa);
Query OK, 20 rows affected (0.00 sec)
Records: 20 Duplicates: 0 Warnings: 0

mysql> alter table precios add index clave_idx (clave);
Query OK, 20 rows affected (0.00 sec)
Records: 20 Duplicates: 0 Warnings: 0

mysql> explain select productos.clave, concat(parte,' ',tipo,' ', especificación) as producto,
proveedores.empresa , precio , pago from productos natural join precios natural join proveedores;
+------------+--------+----------------------+------------+---------+-------------+-----+-----------+
| table      | type   | possible_keys        | key        | key_len | ref         |rows | Extra     |
+------------+--------+----------------------+------------+---------+-------------+-----+-----------+
| proveedores | ALL    | PRIMARY              | NULL       | NULL    | NULL        | 3   |           |
| precios    | ref    | empresa_idx,clave_idex| empresa_idx | 20     | productos.emp| 7   |           |
| productos  | eq_ref | PRIMARY              | PRIMARY    | 4       | precios.clave| 1   |           |
+------------+--------+----------------------+------------+---------+-------------+-----+-----------+
3 rows in set (0.00 sec)
```

Las cosas han cambiado sustancialmente. El gestor ha pasado de leer 24 filas de datos, a leer 11. También ha cambiado el orden de lectura de las tablas, haciendo primero una lectura total de la tabla proveedores (que es inevitable ya que no hemos puesto ninguna condición en el SELECT) y, después, ha aprovechado los índices definidos en 'precios' y en 'productos'.

Veremos más sobre los índices en el subapartado 5.5 "Análisis y optimización" de esta unidad didáctica.

4.3. Manipulación de filas

Para la manipulación de filas disponemos de las sentencias SQL INSERT, UPDATE y DELETE, su uso y sintaxis ya se ha visto en el módulo 3 de este curso. En algunos casos, MySQL nos proporciona extensiones o modificadores que nos pueden ayudar mucho en determinadas situaciones.

- INSERT [DELAYED]. Cuando la sentencia INSERT puede tardar mucho en devolver el resultado (tablas muy grandes o con muchos índices que deben recalcularse al insertar una nueva fila) puede ser interesante añadir la palabra clave DELAYED para que MySQL nos devuelva el control y realice la inserción en segundo plano.

- INSERT [[LOW_PRIORITY] | [HIGH_PRIORITY]]. En tablas muy ocupadas, donde muchos clientes realizan consultas constantemente, una inserción lenta puede bloquear al resto de clientes durante un tiempo. Mediante estos modificadores podemos variar este comportamiento.

- INSERT [IGNORE]. Este modificador convierte los errores de inserción en avisos. Por ejemplo, si intentamos insertar una fila que duplica una clave primaria existente, el SGBD nos devolverá un aviso (y no insertará la nueva fila), pero nuestro programa cliente podrá continuar con su cometido si el resultado de la inserción no era importante para su correcta ejecución.

- UPDATE [LOW_PRIORITY] [IGNORE]. Se comportan de igual modo que en la sentencia INSERT.

- DELETE [QUICK]. Borra el/los registros sin actualizar los índices.

- TRUNCATE. Es una forma muy rápida de borrar todos los registros de una tabla, si no necesitamos saber el número de registros que ha borrado. DELETE FROM <tabla> realiza el mismo cometido, pero devuelve el número de registros borrados.

- LAST_INSERT_ID(). Devuelve el último identificador asignado a una columna de tipo AUTO_INCREMENT después de una sentencia INSERT.

5. Administración de MySQL

Las tareas administrativas como la instalación, gestión de usuarios, copias de seguridad, restauraciones, entre otras, son tareas ineludibles en cualquier organización. Las políticas, los recursos y preferencias de los administradores generan una gran variedad de estilos y mecanismos para llevar a cabo estas tareas, por lo que no es posible hablar de métodos completamente estandarizados en estas áreas.

En este apartado se contemplan las opciones de uso común para la administración de un servidor MySQL. Existen tantas alternativas que no es posible incluirlas todas en un curso. Por tal motivo, en este capítulo se tratan algunos temas de importancia para el administrador, desde una perspectiva general, que permiten obtener una visión global de las posibilidades prácticas de las herramientas administrativas.

En este sentido, el manual de MySQL es la referencia principal para encontrar posibilidades y resolver dudas. En especial se recomienda leer los siguientes capítulos:

- Capítulo 2. Instalación de MySQL.
- Capítulo 4. Administración bases de datos.
- Capítulo 5. Optimización.

La información contenida en ellos es muy amplia y clara, y representa una excelente guía para resolver dudas. Asimismo, se deben tener en cuenta las listas de correo incluidas en el sitio oficial www.mysql.com.

Este capítulo se inicia con una breve reseña del proceso de instalación de MySQL. En la actualidad es posible realizar la instalación a partir de binarios empaquetados que facilitan enormemente el proceso. La administración de usuarios se trata con algo más de detalle, incluyendo una breve descripción de las tablas del directorio de datos. Para los temas de copias de seguridad y restauración se muestran los comandos y utilidades de mayor uso en la práctica omitiendo algunos detalles técnicos poco usuales. La optimización se trata de manera muy general, exponiendo los temas básicos que en la práctica son pasados por alto.

Finalmente, se describe brevemente cómo realizar la replicación de datos en un servidor esclavo.

5.1. Instalación de MySQL

La instalación de MySQL no representa mayores problemas, ya que muchas distribuciones incluyen paquetes con los que realizar la instalación y configuración básica. Sin embargo, aquí veremos la instalación de MySQL utilizando el código fuente que se puede obtener en www.mysql.com. Cabe destacar que el uso de una versión de MySQL compilada tiene la ventaja de que, probablemente,

se adaptará mucho mejor al entorno del servidor donde se ejecutará, proporcionando así un mejor rendimiento. Por contra, implicará más trabajo en caso de que surjan errores en la versión y tengamos que actualizarla. Las instrucciones que se describen en este apartado se basan en la documentación incluida en la distribución.

En primer lugar, debemos asegurarnos de que contamos con las librerías y utilidades necesarias para compilar los ficheros fuente. Principalmente la lista de verificación debe incluir los ficheros siguientes:

- Compilador gcc
- Librerías libgc

El proceso de instalación incluye los siguientes pasos:

- Descomprimir los archivos fuente

```
cd /usr/local/src
tar xzvf mysql-VERSION.tar.gz
cd mysql-VERSION
```

- Configurar la versión de MySQL que vamos a obtener. El script 'configure' admite muchos parámetros que deberemos examinar mediante la opción '--help'. Según los esquemas de tabla que necesitemos o extensiones muy concretas que debamos utilizar, deberemos examinar con cuidado sus opciones. En su versión más simple lo ejecutaríamos de la siguiente manera:

```
./configure –prefix=/usr/local/mysql
```

- Compilar. Procederemos a compilar si no ha habido problemas con la configuración. El parámetro -prefix especifica la ruta del sistema de ficheros donde será instalado.

```
make
```

- Instalar el sistema el servidor ya compilado, mediante la siguiente instrucción:

```
make install
```

- Crear la base de datos inicial del servidor, la que almacenará los usuarios y privilegios. Esta base de datos es imprescindible para que los usuarios se puedan conectar al servidor.

```
scripts/mysql_istall_db
```

- Crear un nuevo usuario y su grupo, para que el servicio se ejecute en un entorno de privilegios restringido en el sistema operativo. En ningún caso se recomienda que el usuario que ejecute el servicio mysqld sea root.

```
groupadd mysql
useradd -g mysql mysql
```

- Todos los archivos deben ser propiedad de root (mysql no debe poder modificarse a sí mismo) y del grupo mysql. El directorio de datos será del usuario mysql para que pueda trabajar con las bases de datos, ficheros de registro, etc.

```
chown -R root /usr/local/mysql
chgrp -R mysql /usr/local/mysql
chown -R mysql /usr/local/mysql/var
```

- Crear el archivo de configuración. La distribución incluye varios archivos de configuración que sirven como plantilla para adaptarlo a nuestras necesidades. En este caso, utilizamos la configuración media como plantilla. Opcionalmente podemos editar el archivo /etc/my.cnf

```
cp support-files/my-medium.cnf /etc/my.cnf
```

- Lanzar el servidor

```
/usr/local/mysql/bin/mysql_safe &
```

- En este estado, el servidor no puede servir aún de SGBD. Por defecto, tendremos creado un usuario '*root*' sin contraseña que podrá acceder tanto desde el equipo local como remotamente. El siguiente paso será asignar una contraseña a este usuario y repasar los usuarios y privilegios definidos. Para asignar la contraseña, deberemos hacer lo siguiente:

```
mysqladmin -u root password "nuevapasswd"
mysqladmin -u root -h host_name password "nuevapasswd"
```

Podemos probar el funcionamiento del SGBD conectando con el cliente 'mysql':

```
mysql -u root -p
```

Veamos ahora algunas características del servidor que acabamos de instalar:

- **mysqld**. El primer método es lanzarlo directamente, se le pueden especificar las opciones que el administrador desee.
- **mysqld_safe**. Es un *script* que ejecuta *mysqld* garantizando una configuración segura. Es mucho más recomendable que ejecutar *mysqld* directamente.
- **mysql_server**. Es un guión que realiza dos tareas: iniciar y detener el servidor *mysqld* con los parámetros start y stop respectivamente. Utiliza *mysqld_safe* para lanzar el servidor *mysqld*. No es común encontrarlo con ese nombre, ya que generalmente se copia como el archivo /etc/init.d/mysql
- **mysql_multi**. Permite la ejecución de múltiples servidores de forma simultanea.

Para detener el servidor básicamente tenemos dos métodos:

- **/etc/init.d/mysql stop**. Es el mecanismo estándar en los sistemas tipo UNIX. Aunque los directorios pueden cambiar.
- **$ mysqladmin -u root -p shutdown**. Es la utilidad para realizar tareas administrativas en un servidor MySQL, en este caso le pasamos el parámetro 'shutdown' para detener el servicio.

Para que los mensajes del servidor aparezcan en español, se debe ejecutar con el parámetro *-language:*

```
$ mysqld --language=spanish
```

Otra opción es agregar en el archivo /etc/my.cnf una línea en la sección [mysqld]

```
[mysqld]
language = /usr/share/mysql/spanish
```

5.2. Usuarios y privilegios

El acceso al servidor MySQL está controlado por usuarios y privilegios. Los usuarios del servidor MySQL no tienen ninguna correspondencia con los

usuarios del sistema operativo. Aunque en la práctica es común que algún administrador de MySQL asigne los mismos nombres que los usuarios tienen en el sistema, son mecanismos totalmente independientes y suele ser aconsejable en general.

El usuario administrador del sistema MySQL se llama *root*. Igual que el superusuario de los sistemas tipo UNIX.

Además del usuario *root*, las instalaciones nuevas de MySQL incluyen el usuario anónimo, que tiene permisos sobre la base de datos test. Si queremos, también podemos restringirlo asignándole una contraseña. El usuario anónimo de MySQL se representa por una cadena vacía. Vemos otra forma de asignar contraseñas a un usuario, desde el cliente de mysql y como usuario *root*:

```
mysql> set password for ''@'localhost' = password('nuevapasswd');
```

La administración de privilegios y usuarios en MySQL se realiza a través de las sentencias:

- **GRANT**. Otorga privilegios a un usuario, en caso de no existir, se creará el usuario.
- **REVOKE**. Elimina los privilegios de un usuario existente.
- **SET PASSWORD**. Asigna una contraseña.
- **DROP USER**. Elimina un usuario.

5.2.1. La sentencia GRANT

La sintaxis simplificada de **grant** consta de tres secciones. No puede omitirse ninguna, y es importante el orden de las mismas:

- **grant** *lista de privilegios*
- **on** *base de datos.tabla*
- **to** *usuario*

Ejemplo

Creación de un nuevo usuario al que se otorga algunos privilegios

```
mysql> grant update, insert, select
-> on demo.precios
-> to visitante@localhost ;
```

En la primera línea se especifican los privilegios que serán otorgados, en este caso se permite actualizar (**update**), insertar (**in**sert) y consultar (**select**). La segunda línea especifica que los privilegios se aplican a la tabla precios de la base de datos *demo*. En la última línea se encuentra el nombre del usuario y el equipo desde el que se va a permitir la conexión.

El comando **grant** crea la cuenta si no existe y, si existe, agrega los privilegios especificados. Es posible asignar una contraseña a la cuenta al mismo tiempo que se crea y se le otorgan privilegios:

```
mysql> grant update, insert, select
-> on demo.precios
-> to visitante@localhost identified by 'nuevapasswd';
```

En la misma sentencia es posible también otorgar permisos a más de un usuario y asignarles, o no, contraseña:

```
mysql> grant update, insert, select
-> on demo.precios
-> to visitante@localhost,
-> yo@localhost identified by 'nuevapasswd',
-> tu@equipo.remoto.com;
```

5.2.2. Especificación de lugares origen de la conexión

MySQL proporciona mecanismos para permitir que el usuario realice su conexión desde diferentes equipos dentro de una red específica, sólo desde un equipo, o únicamente desde el propio servidor.

```
mysql> grant update, insert, select
-> on demo.precios
-> to visitante@'%.empresa.com';
```

El carácter % se utiliza de la misma forma que en el comando **like**: sustituye a cualquier cadena de caracteres. En este caso, se permitiría el acceso del usuario 'visitante' (con contraseña, si la tuviese definida) desde cualquier equipo del dominio 'empresa.com'. Obsérvese que es necesario entrecomillar el nombre del equipo origen con el fin de que sea aceptado por MySQL. Al igual que en **like**, puede utilizarse el carácter '_'.

Entonces, para permitir la entrada desde cualquier equipo en Internet, escribiríamos:

```
-> to visitante@'%'
```

Obtendríamos el mismo resultado omitiendo el nombre del equipo origen y escribiendo simplemente el nombre del usuario:

```
-> to visitante
```

Los anfitriones válidos también se pueden especificar con sus direcciones IP.

```
to visitante@192.168.128.10
to visitante@'192.168.128.%'
```

Los caracteres ' %' y '_' no se permiten en los nombres de los usuarios. 

5.2.3. Especificación de bases de datos y tablas

Después de analizar las opciones referentes a los lugares de conexión permitidos, veamos ahora cómo podemos limitar los privilegios a bases de datos, tablas y columnas.

En el siguiente ejemplo otorgamos privilegios sobre todas las tablas de la base de datos *demo*.

```
mysql> grant all
-> on demo.*
-> to 'visitante'@'localhost';
```

Podemos obtener el mismo resultado de esta forma:

```
mysql> use demo;
mysql> grant all
-> on *
-> to 'visitante'@'localhost';
```

De igual modo, al especificar sólo el nombre de una tabla se interpretará que pertenece a la base de datos en uso:

```
mysql> use demo;
mysql> grant all
-> on precios
-> to 'visitante'@'localhost';
```

Opciones para la clausula on del comando grant

Opción	Significado
.	Todas las bases de datos y todas las tablas
base.*	Todas las tablas de la base de datos especificada
tabla	Tabla especificada de la base de datos en uso
*	Todas las tablas de la base de datos en uso

5.2.4. Especificación de columnas

A continuación presentamos un ejemplo donde se especifican las columnas sobre las que se otorgan privilegios con el comando **grant**:

```
mysql> grant update(precio,empresa)
-> on demo.precios
-> to visitante@localhost;
```

Podemos especificar privilegios diferentes para cada columna o grupos de columnas:

```
mysql> grant update(precio), select (precio, empresa)
-> on demo.precios
-> to visitante@localhost;
```

5.2.5. Tipos de privilegios

MySQL proporciona una gran variedad de tipos de privilegios.

- Privilegios relacionados con tablas: **alter**, **create**, **delete**, **drop**, **index**, **insert**, **select**, **update**
- Algunos privilegios administrativos: file, proccess, super reload, replication client, grant option, shutdown
- Algunos privilegios para fines diversos: **lock tables**, **show databases**, **create temporary tables**.

El privilegio **all** otorga todos los privilegios exceptuando el privilegio **grant option**. Y el privilegio **usage** no otorga ninguno, lo cual es útil cuando se desea, por ejemplo, simplemente cambiar la contraseña:

```
grant usage
on *.*
to visitante@localhost identified by 'secreto';
```

Tipos de privilegios en MySQL

Tipo de privilegio	Operación que permite
all [privileges]	Otorga todos los privilegios excepto **grant option**
usage	No otorga ningún privilegio
alter	Privilegio para alterar la estructura de una tabla
create	Permite el uso de **create table**
delete	Permite el uso de **delete**
drop	Permite el uso de **drop table**
index	Permite el uso de **index** y **drop index**
insert	Permite el uso de **insert**
select	Permite el uso de **select**
update	Permite el uso de **update**
file	Permite le uso de **select . . . into outfile** y **load data infile**
process	Permite el uso de **show full procces list**
super	Permite la ejecución de comandos de supervisión
reload	Permite el uso de **flush**
replication client	Permite preguntar la localización de maestro y esclavo
replication slave	Permite leer los *binlog* del maestro
grant option	Permite el uso de **grant** y **revoke**
shutdown	Permite dar de baja al servidor
lock tables	Permite el uso de **lock tables**
show tables	Permite el uso de **show tables**
create temporary tables	Permite el uso de **create temporary table**

En entornos grandes, es frecuente encontrarse en la necesidad de delegar el trabajo de administrar un servidor de bases de datos para que otros usuarios, además del administrador, puedan responsabilizarse de otorgar privilegios sobre una base de datos particular. Esto se puede hacer en MySQL con el privilegio **grant option**:

```
mysql> grant all, grant option
-> on demo.*
-> to operador@localhost;
```

El mismo resultado se puede obtener con la siguiente sintaxis alternativa:

```
mysql> grant all
-> on demo.*
-> to operador@localhost
-> with grant option;
```

De este modo el usuario *operador* podrá disponer de todos los privilegios sobre la base de datos *demo*, incluido el de controlar el acceso a otros usuarios.

5.2.6. Opciones de encriptación

MySQL puede establecer conexiones seguras encriptándolas mediante el protocolo SSL*; de esta manera, los datos que se transmiten (tanto la consulta, en un sentido, como el resultado, en el otro) entre el cliente y el servidor estarán protegidos contra intrusos. Para especificar que un usuario debe conectarse obligatoriamente con este protocolo, se utiliza la cláusula **require**:

** Secure Sockets Layer*

```
mysql> grant all
-> on *.*
-> to visitante@localhost
-> require ssl;
```

Las conexiones encriptadas ofrecen protección contra el robo de información, pero suponen una carga adicional para el servicio, que debe desencriptar la petición del cliente y encriptar la respuesta (además de un proceso más largo de negociación al conectar), por ello, merman el rendimiento del SGBD.

5.2.7. Limites de uso

Los recursos físicos del servidor siempre son limitados: si se conectan muchos usuarios al mismo tiempo al servidor y realizan consultas o manipulaciones de datos complejas, es probable que pueda decaer el rendimiento notablemente. Una posible solución a este problema es limitar a los usuarios el trabajo que pueden pedir al servidor con tres parámetros:

- Máximo número de conexiones por hora.
- Máximo número de consultas por hora.
- Máximo número de actualizaciones por hora.

La sintaxis de estas limitaciones es como se muestra a continuación:

```
mysql> grant all
-> on *.*
-> to
-> with MAX_CONECTIONS_PER_HOUR 3
-> MAX_QUERIES_PER_HOUR 300
-> MAX_UPDATES_PER_HOUR 30;
```

5.2.8. Eliminar privilegios

El comando **revoke** permite eliminar privilegios otorgados con **grant** a los usuarios. Veamos un ejemplo representativo:

```
revoke all
on *.*
from visitante@localhost;
```

Al ejecutar este comando se le retiran al usuario *visitante* todos sus privilegios sobre todas las bases de datos, cuando se conecta desde *localhost*.

El comando anterior no retira todos los privilegios del usuario *visitante*, sólo se los retira cuando se conecta desde *localhost*. Si el usuario se conecta desde otra localidad (y tenía permiso para hacerlo) sus privilegios permanecen intactos.

5.2.9. Eliminar usuarios

Antes de proceder a la eliminación de un usuario, es necesario asegurarse de que se le han quitado primero todos sus privilegios. Una vez asegurado este detalle, se procede a eliminarlo mediante el comando **drop user**:

```
mysql> drop user visitante;
```

5.2.10. La base de datos de privilegios: mysql

MySQL almacena la información sobre los usuarios y sus privilegios en una base de datos como cualquier otra, cuyo nombre es *mysql*. Si exploramos su estructura, entenderemos la manera como MySQL almacena la información de sus usuarios y privilegios:

```
mysql -u root -p
mysql> use mysql;
mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
```

```
mysql> show columns from user;
+-----------------+------------------+------+-----+---------+-------+
| Field           | Type             | Null | Key | Default | Extra |
+-----------------+------------------+------+-----+---------+-------+
| Host            | char(60) binary  | PRI  |     |         |       |
| User            | char(16) binary  | PRI  |     |         |       |
| Password        | char(16) binary  |      |     |         |       |
| Select_priv     | enum('N','Y')    | N    |     |         |       |
| Insert_priv     | enum('N','Y')    |      | N   |         |       |
| Update_priv     | enum('N','Y')    | N    |     |         |       |
| Delete_priv     | enum('N','Y')    |      | N   |         |       |
| Create_priv     | enum('N','Y')    | N    |     |         |       |
| Drop_priv       | enum('N','Y')    | N    |     |         |       |
| Reload_priv     | enum('N','Y')    |      | N   |         |       |
| Shutdown_priv   | enum('N','Y')    | N    |     |         |       |
| Process_priv    | enum('N','Y')    | N    |     |         |       |
| File_priv       | enum('N','Y')    |      | N   |         |       |
| Grant_priv      | enum('N','Y')    |      | N   |         |       |
| References_priv | enum('N','Y')    |      | N   |         |       |
| Index_priv      | enum('N','Y')    |      | N   |         |       |
| Alter_priv      | enum('N','Y')    |      |N    |         |       |
+-----------------+------------------+------+-----+---------+-------+
17 rows in set (0.04 sec)

mysql> show columns from db;
+-----------------+------------------+------+-----+---------+-------+
| Field           | Type             | Null | Key | Default | Extra |
+-----------------+------------------+------+-----+---------+-------+
| Host            | char(60) binary  | PRI  |     |         |       |
| Db              | char(64) binary  | PRI  |     |         |       |
| User            | char(16) binary  | PRI  |     |         |       |
| Select_priv     | enum('N','Y')    |      | N   |         |       |
| Insert_priv     | enum('N','Y')    |      | N   |         |       |
| Update_priv     | enum('N','Y')    |      | N   |         |       |
| Delete_priv     | enum('N','Y')    |      | N   |         |       |
| Create_priv     | enum('N','Y')    |      | N   |         |       |
| Drop_priv       | enum('N','Y')    |      | N   |         |       |
| Grant_priv      | enum('N','Y')    |      | N   |         |       |
| References_priv | enum('N','Y')    |      | N   | N       |       |
| Index_priv      | enum('N','Y')    |      | N   |         |       |
| Alter_priv      | enum('N','Y')    |      | N   |         |       |
+-----------------+------------------+------+-----+---------+-------+
13 rows in set (0.00 sec)

mysql> show columns from tables_priv;
+-------------+-----------------------------------------------------------------------------------------------+----+---+-------+-----+
| Field       | Type                                                                                          |Null|Key|Default|Extra|
+-------------+-----------------------------------------------------------------------------------------------+----+---+-------+-----+
| Host        | char(60) binary                                                                               |PRI |   |       |     |
| Db          | char(64) binary                                                                               |PRI |   |       |     |
| User        | char(16) binary                                                                               |PRI |   |       |     |
| Table_name  | char(60) binary                                                                               |    | N |       |     |
| Grantor     | char(77)                                                                                      |    | N |       |     |
| Timestamp   | timestamp(14)                                                                                 |    | N |       |     |
| Table_priv  | set('Select','Insert','Update','Delete','Create','Drop','Grant','References','Index','Alter')|    | N |       |     |
| Column_priv | set('Select','Insert','Update','References')                                                  |N   |   |       |     |
+-------------+-----------------------------------------------------------------------------------------------+----+---+-------+-----+
8 rows in set (0.00 sec)
```

Es posible realizar modificaciones directamente sobre estas tablas y obtener los mismos resultados que si utilizáramos los comandos **grant**, **revoke**, **set password** o **drop user**:

```
mysql> update user
-> set Password = password('nuevapasswd')
-> where User ='visitante' and Host = 'localhost';
mysql> flush privileges;
```

El comando **flush privileges** solicita a MySQL que vuelva a leer las tablas de privilegios. En el momento de ejecutarse, el servidor lee la información de estas tablas sobre privilegios. Pero si se han alterado las tablas manualmente, no se enterará de los cambios hasta que utilicemos el comando **flush privileges**.

Tablas de la base de datos *mysql*

Tabla	Contenido
user	Cuentas de usuario y sus privilegios globales
db	Privilegios sobres bases de datos
tables_priv	Privilegios sobre tablas
columns_priv	Privilegios sobre columnas
host	Privilegios de otros equipos anfitriones sobre bases de datos

El acceso directo a las tablas de privilegios es útil en varios casos; por ejemplo, para borrar un usuario del sistema en las versiones de MySQL anteriores a la 4.1.1:

```
mysql> delete from user
-> where User = 'visitante' and Host = 'localhost';
mysql> flush privileges;
```

No es posible eliminar mediante un solo comando **revoke** todos los privilegios de un usuario.

Ejemplo

Se otorgan derechos a un usuario con dos comandos **grant**.

Observando el contenido de la base de datos de privilegios, podemos entender el comportamiento de los comandos **grant** y **revoke**. Primero asignamos privilegios para usar el comando **select** al usuario visitante con dos comandos **grant**: el primero de ellos le permite el ingreso desde el servidor *nuestra-ong.org* y el segundo le otorga el mismo tipo de privilegio, pero desde cualquier equipo en Internet.

```
mysql> grant select
-> on *.*
-> to visitante@nuestra-ong.org;
Query OK, 0 rows affected (0.01 sec)
mysql> grant select
-> on *.*
-> to visitante@'%';
Query OK, 0 rows affected (0.00 sec)
```

Consultando la tabla *user* de la base de datos de privilegios, podemos observar los valores 'Y' en la columna del *privilegio select*.

```
mysql> select user,host,select_priv from user
-> where user = 'visitante';
```

```
+-----------+-----------------+-------------+
|   user    |      host       | select_priv |
+-----------+-----------------+-------------+
| visitante | nuestra-ong.org |      Y      |
| visitante |        %        |      Y      |
+-----------+-----------------+-------------+
2 rows in set (0.00 sec)
```

Ahora solicitamos eliminar el *privilegio select* de todas las bases de datos y de todos los equipos en Internet.

```
mysql> revoke all
-> on *.*
-> from visitante@'%';
Query OK, 0 rows affected (0.00 sec)
mysql> select user,host,select_priv from user
-> where user = 'visitante';
```

```
+-----------+-----------------+-------------+
|   user    |      host       | select_priv |
+-----------+-----------------+-------------+
| visitante | nuestra-ong.org |      Y      |
| visitante |        %        |      N      |
+-----------+-----------------+-------------+
2 rows in set (0.01 sec)
```

En la tabla *user* observamos que, efectivamente, se ha eliminado el privilegio para visitante@'%' pero no para 'visitante@nuestra-ong.org'. MySQL considera que son direcciones diferentes y respeta los privilegios otorgados a uno cuando se modifica otro.

5.3. Copias de seguridad

Ningún sistema es perfecto ni está a salvo de errores humanos, cortes en el suministro de la corriente eléctrica, desperfectos en el *hardware* o errores de *software*; así que una labor más que recomendable del administrador del servidor de bases de datos es realizar copias de seguridad y diseñar un plan de contingencia. Se deben hacer ensayos del plan para asegurar su buen funcionamiento y, si se descubren anomalías, realizar los ajustes necesarios.

No existe una receta universal que nos indique cómo llevar nuestras copias de seguridad de datos. Cada administrador debe diseñar el de su sistema de acuerdo a sus necesidades, recursos, riesgos y el valor de la información.

MySQL ofrece varias alternativas de copia de seguridad de la información. La primera que podemos mencionar consiste simplemente en copiar los archivos de datos. Efectivamente, es una opción válida y sencilla.

En primera instancia son necesarios dos requisitos para llevarla a cabo:

- Conocer la ubicación y estructura del directorio de datos.
- Parar el servicio MySQL mientras se realiza la copia.

En cuanto a la ubicación y estructura del directorio, recordemos que la distribución de MySQL ubica el directorio de datos en /usr/local/mysql/var, las distribuciones GNU/Linux basadas en paquetes como DEB o RPM ubican, por lo general, los datos en /var/lib/mysql.

Si por algún motivo no encontramos el directorio de datos, podemos consultarlo a MySQL. El comando **show variables** nos muestra todas las variables disponibles, basta realizar un filtro con la clausula **like**:

```
mysql> show variables like 'datadir';
+---------------+-----------------+
| Variable_name |      Value      |
+---------------+-----------------+
|    datadir    | /var/lib/mysql/ |
+---------------+-----------------+
1 row in set (0.00 sec)
```

Una vez ubicados los archivos, detenemos la ejecución del servidor: un modo sencillo de asegurarnos de que la base de datos no será modificada mientras terminamos la copia:

```
$ mysqladmin -u root -p shutdown
```

Finalmente, copiamos el directorio completo con todas las bases de datos:

```
$ cp -r /var/lib/mysql/ /algun_dir/
```

Por supuesto podemos elegir otras formas de copiarlo o comprimirlo, de acuerdo a nuestras preferencias y necesidades.

```
$ tar czf mysql-backup.tar.gz /var/lib/mysql
```

Si queremos copiar sólo una base de datos, copiamos el directorio con el mismo nombre de la base de datos:

```
$ cp -r /var/lib/mysql/demo/ /algun_dir/respaldo_demo/
```

También es posible hacer copia de seguridad de una sola tabla.

```
$ cp -r /var/lib/mysql/demo/productos.* / algun_dir/backup_demo/
```

Como podemos observar, la organización de la base de datos en MySQL es muy simple:

- Todas las bases de datos se almacenan en un directorio, llamado el directorio de datos(*datadir*).

- Cada base de datos se almacena como un subdirectorio del directorio de datos.
- Cada tabla se almacena en un archivo, acompañada de otros archivos auxiliares con el mismo nombre y diferente extensión.

El problema de este mecanismo es que debemos detener el servicio de bases de datos mientras realizamos el respaldo.

5.3.1. mysqlhotcopy

Un mecanismo que permite realizar la copia de los archivos del servidor sin necesidad de detener el servicio es el *script* '*mysqlhotcopy*'. El *script* está escrito en Perl y bloquea las tablas mientras realiza el respaldo para evitar su modificación. Se usa de la siguiente manera:

```
$ mysqlhotcopy demo /algun_directorio
```

En este caso, creará un directorio /algun_directorio/demo con todos los archivos de la base de datos.

El comando *mysqlhotcopy* puede recibir sólo el nombre de una base de datos como parámetro:

```
$ mysqlhotcopy demo
```

En este caso, creará un directorio /var/lib/mysql/demo_copy.

Este método no funciona para tablas con el mecanismo de almacenamiento tipo InnoDB.

5.3.2. mysqldump

Las dos opciones anteriores representan copias binarias de la base de datos. El comando *mysqldump*, en cambio, realiza un volcado de las bases de datos pero traduciéndolas a SQL; es decir, entrega un archivo de texto con todos los comandos necesarios para volver a reconstruir las bases de datos, sus tablas y sus datos. Es el método más útil para copiar o distribuir una base de datos que deberá almacenarse en otros servidores.

```
$ mysqldump demo > demo.sql
```

El comando *mysqldump* ofrece multitud de parámetros para modificar su comportamiento o el tipo de volcado generado: por defecto, genera sentencias SQL, pero puede generar ficheros de datos tipo CSV u otros formatos. También podemos especificarle que haga el volcado de todas las bases de datos o que sólo vuelque los datos y no la creación de las tablas, etc.

Las primeras líneas del archivo *demo.sql* según el ejemplo anterior tendrían el siguiente aspecto:

```
~$ mysqldump demo | head -25
-- MySQL dump 8.21
--
-- Host: localhost Database: demo
---------------------------------------------------------
-- Server version 3.23.49-log
--
-- Table structure for table 'ganancia'
--
DROP TABLE IF EXISTS ganancia;
CREATE TABLE ganancia (
venta enum('Por mayor','Por menor') default NULL,
factor decimal(4,2) default NULL
) TYPE=MyISAM;
--
Dumping data for table 'ganancia'
--
INSERT INTO ganancia VALUES ('Por mayor',1.05);
INSERT INTO ganancia VALUES ('Por menor',1.12);--
```

La ventaja de utilizar *mysqldump* es que permite que los archivos puedan ser leídos (y modificados) en un simple editor de textos, y pueden ser utilizados para migrar la información a otro SGBD que soporte SQL. Además soporta todos los tipos de tablas. La desventaja es que su procesamiento es lento y los archivos que se obtienen son muy grandes.

5.3.3. Restaurar a partir de respaldos

En algún momento, sea por el motivo que sea, necesitaremos realizar la restauración de nuestras bases de datos.

Si tenemos una copia binaria del directorio de datos, bastará con copiarla al directorio original y reiniciar el servidor:

```
# mysqladmin -u root -p shutdown
# cp /algun_dir/respaldo-mysql/* /var/lib/mysql
# chown -R mysql.mysql /var/lib/mysql
# mysql_safe
```

Es importante restaurar también el dueño y el grupo de los archivos de datos, para tener los accesos correctamente establecidos. En este ejemplo se adopta el supuesto que el usuario mysql es el que ejecuta el servidor *mysqld*.

La restauración de un archivo SQL obtenido con *mysqldump*, se realiza desde el cliente *mysql*, la base de datos debe existir, ya que el archivo *demo.sql* no la crea por defecto.

```
$ mysql demo -u root -p < demo.sql
```

5.4. Reparación de tablas

En determinadas circunstancias de uso muy frecuente, como la inserción y borrado masivos de datos, coincidiendo con bloqueos del sistema o llenado del espacio en disco u otras circunstancias, es posible que una tabla o algunos de sus índices se corrompan.

Podemos consultar el estado de integridad de una tabla con el comando **check table**, que realiza algunas verificaciones sobre la tabla en busca de errores y nos entrega un informe con las siguientes columnas de información:

```
mysql> check table precios;
+--------------+-------+----------+----------+
|    Table     |  Op   | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| demo.precios | check |  status  |    OK    |
+--------------+-------+----------+----------+
1 row in set (0.03 sec)
```

La columna *Op* describe la operación que se realiza sobre la tabla. Para el comando **check table** esta columna siempre tiene el valor *check* porque ésa es la operación que se realiza. La columna *Msg_type* pude contener uno de los valores *status*, *error*, info, o *warning*. Y la columna *Msg_text* es el texto que reporta de alguna situación encontrada en la tabla.

Es posible que la información entregada incluya varias filas con diversos mensajes, pero el último mensaje siempre debe ser el mensaje *OK* de tipo *status*.

En otras ocasiones **check table** no realizará la verificación de tabla, en su lugar entregará como resultado el mensaje *Table is already up to date*, que significa que el gestor de la tabla indica que no hay necesidad de revisarla.

MySQL no permite realizar consultas sobre una tabla dañada y enviará un mensaje de error sin desplegar resultados parciales:

```
mysql> select * from precios;
ERROR 1016: No puedo abrir archivo: 'precios.MYD'. (Error: 145)
```

Para obtener información del significado del error 145, usaremos la utilidad en línea de comandos *perror*:

```
$ perror 145
145 = Table was marked as crashed and should be repaired
```

Después de un mensaje como el anterior, es el momento de realizar una verificación de la integridad de la tabla para obtener el reporte. 

```
mysql> check table precios extended;
+-------------+------+----------+---------------------------------------+
|    Table    |  Op  | Msg_type |              Msg_text                 |
+-------------+------+----------+---------------------------------------+
|demo.precios |check |  error   |Size of datafile is:450 Should be:452 |
|demo.precios |check |  error   |               Corrupt                 |
+-------------+------+----------+---------------------------------------+
2 rows in set (0.00 sec)
```

En este caso localizamos dos errores en la tabla. La opción **extended** es uno de los cinco niveles de comprobación que se pueden solicitar para verificar una tabla.

Tipos de verificación

Tipo	Significado
quick	No revisa las filas en busca de referencias incorrectas.
fast	Solamente verifica las tablas que no fueron cerradas adecuadamente.
changed	Verifica sólo las tablas modificadas desde la última verificación o que no se han cerrado apropiadamente.
medium	Revisa las filas para verificar que los ligados borrados son correctos, verifica las sumas de comprobación de las filas.
extended	Realiza una búsqueda completa en todas las claves de cada columna. Garantiza el 100% de la integridad de la tabla.

La sentencia **repair table** realiza la reparación de tablas tipo MyISAM corruptas:

```
mysql> repair table precios;
+-------------+------+---------+---------------------------------------+
|    Table    |  Op  | Msg_type|               Msg_text                |
+-------------+------+---------+---------------------------------------+
|demo.precios |repair|   info  | Wrong bytesec: 0-17-1 at 168;Skipped |
|demo.precios |repair| warning | Number of rows changed from 20 to 7  |
|demo.precios |repair|  status |                  OK                   |
+-------------+------+--------+---------------------------------------+
3 rows in set (0.04 sec)
```

El segundo mensaje informa de la pérdida de 13 filas durante el proceso de reparación. Esto significa, como es natural, que el comando **repair table** es útil sólo en casos de extrema necesidad, ya que no garantiza la recuperación total de la información. En la práctica, siempre es mejor realizar la restauración de la información utilizando las copias de seguridad. En caso de desastre, se debe conocer el motivo que origina la corrupción de las tablas y tomar las medidas adecuadas para evitarlo. En lo que respecta a la estabilidad de MySQL, se puede confiar en que muy probablemente nunca será necesario utilizar el comando **repair table**.

El comando **optimize table** puede también realizar algunas correcciones sobre una tabla.

5.4.1. myisamchk

El programa **myisamchk** es una utilidad en línea de comandos que se incluye con la distribución de MySQL y sirve para reparar tablas tipo MyISAM. Para utilizarlo con seguridad el servidor no debe estar ejecutándose y se recomienda realizar un respaldo del directorio de datos antes de su utilización.

Recibe como parámetro principal los archivos .MYI correspondientes a las tablas que hay que revisar; es decir, **myisamchk** no conoce la ubicación del directorio de datos. Por ejemplo, si el directorio de datos está ubicado en /val/lib/mysql, las siguientes serían dos maneras de realizar una comprobación de los archivos de la base de datos *demo*:

```
# myisamchk /var/lib/mysql/demo/*.MYI
# cd /var/lib/lib/mysql/demo
# myisamchk *.MYI
```

Se pueden revisar todas las bases de datos utilizando '*' para denominar el directorio de la base de datos:

```
# myisamchk /var/lib/mysql/*/*.MYI
```

Para realizar una comprobación rápida, el manual sugiere utilizar el siguiente comando:

```
# myisamchk --silent --fast *.MYI
```

Y para realizar la corrección de las tablas corruptas, el manual sugiere la sintaxis siguiente:

```
# myisamchk --silent --force --update-state -O key_buffer=64M \
-O sort_buffer=64M -O read_buffer=1M -O write_buffer=1M *.MYI
```

Las opciones dadas por -O se refieren al uso de memoria, que permiten acelerar de forma notoria el proceso de reparación.

`--force` reinicia myisamchk con el parámetro `--recover` cuando encuentra algún error.

`--updatestate` almacena información sobre el resultado del análisis en la tabla MYI.

Nota

En la práctica con estas opciones se logran corregir los errores más comunes. Para conocer otras opciones de recuperación con **myisamchk**, podéis consultar el manual que acompaña a la distribución de MySQL.

5.5. Análisis y optimización

El diseño de MySQL le permite funcionar con un rendimiento notable, sin embargo, se pueden cometer fácilmente errores que disminuyan la capacidad de respuesta del servidor. También se pueden realizar algunos ajustes a la configuración de MySQL que incrementan su rendimiento.

5.5.1. Indexación

La indexación es la principal herramienta para optimizar el rendimiento general de cualquier base de datos. Es también la más conocida por los usuarios de servidores MySQL y, paradójicamente, su no utilización es una de las principales causas de bajo rendimiento en servidores de bases de datos.

> Muchos administradores y diseñadores simplemente parecen olvidar usar índices para optimizar los accesos a las bases de datos. Por otro lado, algunas personas tienden a indexar todo, esperando que de esta manera el servidor acelere cualquier tipo de consulta que se le solicite. En realidad, esta práctica puede causar una disminución en el rendimiento, sobre todo en lo que respecta a inserciones y modificaciones.

Para ver las ventajas de utilizar índices, analizaremos en primer término una simple búsqueda en una tabla sin índice alguno:

- El constante acceso de escritura de una tabla la mantiene desordenada.
- La ordenación de una tabla es una operación costosa: el servidor tendría que detenerse un tiempo considerable para ordenar sus tablas.
- Muchas tablas tienen más de un criterio de ordenación: ordenar según una columna implica desordenar otra.
- La inserción y eliminación de datos sin alterar el orden en una tabla es costosa: la inserción de un registro en una tabla grande implicaría una larga espera en la actualización de la misma.
- Si se opta por mantener la tabla desordenada (que es la opción más viable), una búsqueda implicaría forzosamente un recorrido secuencial (también denominado *full scan*), registro por registro.

El uso de índices en la ordenación de las bases de datos ofrece las ventajas siguientes:

- Permite ordenar las tablas por varios criterios simultáneamente.
- Es menos costoso ordenar un archivo índice, porque incluye sólo referencias a la información y no la información en sí.
- El coste de inserción y eliminación es menor.
- Con los registros siempre ordenados se utilizaran algoritmos mucho más eficientes que el simple recorrido secuencial en las consultas.

El uso de índices también comporta alguna desventaja:

- Los índices ocupan espacio en disco.

- Aún teniendo registros pequeños, el mantener en orden un índice disminuye la velocidad de las operaciones de escritura sobre la tabla.

A pesar de estos inconvenientes, la utilización de índices ofrece mayores ventajas que desventajas, sobre todo en la consulta de múltiples tablas, y el aumento de rendimiento es mayor cuanto mayor es la tabla.

Consideremos por ejemplo una consulta sobre las tablas A, B, y C, independientemente del contenido de la cláusula **where**, las tres tablas se deben de combinar para hacer posible posteriormente el filtrado según las condiciones dadas:

```
select *
from A,B,C
where A.a = B.b
and B.b = C.c;
```

Consideremos que no son tablas grandes, que no sobrepasan los 1.000 registros. Si A tiene 500 registros, B tiene 600 y C 700, la tabla resultante de la consulta anterior tendrá 210 millones de registros. MySQL haría el producto cartesiano de las tres tablas y, posteriormente, se recorrería la relación resultante para buscar los registros que satisfacen las condiciones dadas, aunque al final el resultado incluya solamente 1.000 registros.

Si utilizamos índices MySQL los utilizaría de una forma parecida a la siguiente:

- Tomaría cada uno de los registros de A.

- Por cada registro de A, buscaría los registros en B que cumpliesen con la condición A.a = B.b. Como B está indexado por el atributo 'b', no necesitaría hacer el recorrido de todos los registros, simplemente accedería directamente al registro que cumpliese la condición.

- Por cada registro de A y B encontrado en el paso anterior, buscaría los registros de C que cumpliesen la condición B.b = C.c. Es el mismo caso que en el paso anterior.

Comparando las dos alternativas de búsqueda, la segunda ocuparía cerca del 0,000005% del tiempo original. Por supuesto que sólo se trata de una aproximación teórica, pero adecuada para comprender el efecto de los índices en las consultas sobre bases de datos.

5.5.2. Equilibrio

El índice ideal debería tener las siguientes características:

- Los registros deberían ser lo más pequeños posible.
- Sólo se debe indexar valores únicos.

Analicemos cada recomendación:

- Cuanto más pequeños sean los registros, más rápidamente se podrán cambiar de lugar (al insertar, modificar o borrar filas), además, en un momento dado, el índice puede permanecer en memoria. Consideremos las dos definiciones posibles:

```
create table Empresa(
nombre char(30),
teléfono char(20),
index (nombre)
);
```

En esta tabla el índice se realiza sobre *nombre*, que es un campo de 30 caracteres, y se utiliza como clave para hacer los '*joins*' con otras tablas.

Ahora considérese la siguiente alternativa:

```
create table Empresa(
id int ,
nombre char(30),
teléfono char(20),
index (id)
);
```

Se agrega una columna que servirá como identificador de la empresa. Desde el punto de vista de rendimiento implica una mejora, ya que el índice se realiza sobre números enteros, por lo tanto, ocupará menos espacio y funcionará más rápido.

Cuanto más pequeña sea la columna indexada mayor velocidad se tendrá en el acceso a la tabla.

- Consideremos el índice siguiente, creado para disminuir la necesidad de efectuar accesos a la tabla:

```
create table Empresa(
nombre char(30),
crédito enum{'SI','NO'},
index(crédito)
);
```

Si consideramos que un índice se crea para evitar la necesidad de recorrer la tabla, veremos que el índice creado es prácticamente inútil, ya que alguno de los valores ocurre el 50% o más de las veces: para encontrar todos los resultados hay que recorrer gran parte de la tabla. MySQL no utiliza los índices que implican un 30% de ocurrencias en una tabla.

Aun así, y exceptuando casos exagerados como este último, puede ser interesante indexar una tabla por algún atributo que no sea único, si ese atributo se utiliza para ordenar los resultados. También puede ser conveniente crear un índice por varios atributos simultáneamente si se usan todos en alguna consulta en la cláusula ORDER BY.

Cuanto menor sea la repetición de valores en una columna indexada, menor será la necesidad de acceder a la tabla y más eficiente será el índice.

5.5.3. La *cache* de consultas de MySQL

El servidor MySQL incluye la posibilidad de utilizar una *cache** con los resultados de las últimas consultas para acelerar la velocidad de respuesta. Esta solución es útil cuando las tablas tienen relativamente pocos cambios y se realizan los mismos tipos de consultas. El funcionamiento de la *cache* se basa en las premisas siguientes:

* Memoria intermedia de acceso rápido.

- La primera vez que se recibe una consulta se almacena en la *cache*.
- Las siguientes veces la consulta se realiza primero en la *cache*; si tiene éxito, el resultado se envía inmediatamente.

La *cache* tiene las siguientes características:

- El servidor compara el texto de la consulta; aunque técnicamente sea igual si difiere en uso de mayúsculas-minúsculas o cualquier otro cambio, no se considera la solicitud idéntica y no será tratada por la *cache*.
- Si alguna tabla incluida en alguna consulta cambia, el contenido de la consulta es eliminado de la *cache*.

La configuración de la *cache* se realiza a través de variables globales:

- **query_cache_limit**. No almacena resultados que sobrepasen dicho tamaño. Por omisión es de 1M.
- **query_cache_size**. Tamaño de la memoria cache expresada en bytes. Por omisión es 0; es decir, no hay cache.
- **query_cache_type**. Puede tener tres valores: ON , OFF o DEMAND.

Tipos de *cache*

Valor	Tipo	Significado
0	OFF	*Cache* desactivado
1	ON	*Cache* activado
2	DEMAND	Sólo bajo solicitud explicita

Cuando la *cache* del servidor esta en modo DEMAND, se debe solicitar explícitamente que la consulta utilice o no la *cache*:

```
select sql_cache
select sql_no_cache
```

5.6. Replicación

La replicación es la copia sincronizada entre dos servidores de bases de datos de forma que cualquiera de los dos puede entregar los mismos resultados a sus clientes.

MySQL incluye la posibilidad de replicación con las siguientes características:

- Funciona con el esquema *maestro-esclavo*: existe un servidor maestro que lleva el control central y uno o varios servidores esclavos que se mantienen sincronizados con el servidor maestro.

- La réplica se realiza mediante un registro de los cambios realizados en la base de datos: no se realizan las copias de las bases de datos para mantenerlas sincronizadas, en su lugar se informa de las operaciones realizadas en el servidor maestro (insert, delete , update ...) para que las realicen a su vez los servidores esclavos.

- No es posible realizar cambios en los servidores esclavos, son exclusivamente para consultas.

Este sencillo esquema permite la creación de replicas sin mayores complicaciones obteniendo los siguientes beneficios:

- Se distribuye la carga de trabajo.

- El sistema es redundante, por lo que en caso de desastre hay menos probabilidades de perder los datos.

- Es posible realizar los respaldos de un esclavo sin interrumpir el trabajo del servidor maestro.

5.6.1. Preparación previa

El equipo maestro debe tener acceso por red. Antes de realizar la configuración de los servidores maestro y esclavo es necesario realizar las siguientes tareas:

- Asegurarse de que en ambos está instalada la misma versión de MySQL.
- Asegurarse de que ninguno de los servidores atenderá peticiones durante el proceso de configuración.
- Asegurarse de que las bases de datos del servidor maestro han sido copiadas manualmente en el servidor esclavo, de manera que en ambos se encuentre exactamente la misma información.
- Asegurarse de que ambos atienden conexiones vía TCP/IP. Por seguridad, esta opción está desactivada por omisión. Para activarla se debe comentar la línea *skip_networking* en el archivo de configuración /etc/my.cnf

5.6.2. Configuración del servidor maestro

En el servidor maestro creamos una cuenta de usuario con permisos de replicación para autorizar, en el servidor maestro, al nuevo usuario para realizar réplicas:

```
mysql> grant replication slave
-> on *.*
-> to replicador@esclavo.empresa.com identified by 'secreto';
```

Replicador es el nombre del nuevo usuario.

Esclavo.empresa.com es la dirección del servidor esclavo.

'Secreto' es la contraseña.

El servidor maestro llevará un archivo de registro '*binlog*' donde se registrarán todas las solicitudes de actualización que se realicen en las bases de datos. Para activar la creación de este archivo debemos editar el archivo /etc/my.cnf y agregar las siguientes líneas en la sección [mysqld]:

```
[mysqld]
log-bin
server-id = 1
```

El servidor maestro debe identificarse con un id, en este caso será el número 1. a continuación, reiniciamos el servidor:

```
/etc/init.d/mysql restart
```

Finalmente, consultamos el nombre del archivo '*binlog*' y la posición de compensación (estos datos son necesarios para configurar el esclavo):

```
mysql> show master status;
+-----------------+----------+--------------+------------------+
|      File       | Position | Binlog_do_db | Binlog_ignore_db |
+-----------------+----------+--------------+------------------+
| maestro-bin.001 |    76    |              |                  |
+-----------------+----------+--------------+------------------+
```

5.6.3. Configuración del servidor esclavo

En el servidor esclavo, editamos el archivo /etc/my.cnf y agregamos, al igual que en el maestro, la activación del archivo '*binlog*' y un identificador del servidor (que debe ser distinto del identificador del servidor maestro):

```
[mysqld]
log-bin
server-id = 2
```

Reiniciamos el servidor esclavo:

```
# /etc/init.d/mysql restart
```

Configuramos los datos del maestro en el servidor esclavo:.

```
mysql> change master to
-> master_host = 'maestro.empresa.com',
-> master_user = 'replicador',
-> master_password = 'secreto',
-> master_log_file = 'maestro-log.001',
-> master_log_pos = 76;
```

El último paso es iniciar el servidor esclavo:

```
mysql> start slave;
```

Y ya tendremos el servidor esclavo funcionando.

5.7. Importación y exportación de datos

En muchas ocasiones es necesario mover datos de una aplicación a otra, para ello son necesarios formatos estándares que puedan ser escritos por la aplicación origen y leídos por la aplicación destino. El más simple de esos formatos es el texto plano, donde cada archivo es una tabla, cada fila es un registro y los valores de los campos se separan por tabuladores.

MySQL puede leer este tipo de archivos, incluyendo valores nulos representados por '\N'(N mayúscula).s

Utilizando el cliente *mysql*, podemos introducir los datos del archivo local *proveedores.txt* en la tabla *proveedores*:

```
mysql> load data local infile 'proveedores.txt'
-> into table proveedores;
```

Si se omite la palabra **local**, MySQL buscará el archivo en el servidor y no en el cliente.

En un archivo se pueden entrecomillar los campos, utilizar comas para separarlos y terminar las líneas con los caracteres '\r\n' (como en los archivos Windows). El comando **load data** tiene dos clausulas opcionales, **fields**, en el que se especifican estos parámetros.

```
mysql> load data local infile 'prooveedores.txt'
-> fields terminated by ','
-> enclosed by '"'
-> lines terminated by '\r\n';
```

La opción **enclosed by** puede tener la forma **optionaly enclosed by**, en caso de que los campos numéricos no sean delimitados.

Además pueden omitirse las primeras lineas del archivo si contienen información de encabezados:

```
mysql> load data local infile 'proveedores.txt'
-> ignore 1 lines;
```

5.7.1. mysqlimport

La utilidad **mysqlimport** que se incluye en la distribución puede realizar el mismo trabajo que **load data**. Estos son algunos de sus parámetros:

```
mysqlimport basededatos archivo.txt
```

Estos son algunos de los argumentos de **mysqlimport** para realizar las tareas equivalentes a la sentencia **load data**:

```
--fields-terminated-by=
--fields-enclosed-by=
--fields-optionally-enclosed-by=
--fields-escaped-by=
--lines-terminated-by=
```

La forma más simple para exportar datos es redireccionando la salida del cliente *mysql*. El parámetro -e permite ejecutar un comando en modo de procesamiento por lotes. MySQL detecta si la salida es en pantalla o está redireccionada a un archivo y elige la presentación adecuada: con encabezados y líneas de separación para la salida en pantalla, y sin encabezados y con tabuladores para un archivo:

```
$ mysql demo -e "select * from proveedores" > proveedores.txt
```

La sentencia **select** también cuenta con una opción para realizar la tarea inversa de la sentencia **load data**:

```
mysql> select *
-> into outfile "/tmp/proveedores.txt"
-> fields termitated by ','
-> optionaly enclosed by '"'
-> lines termitates by '\n'
-> from proveedores;
```

5.7.2. mysqldump

La utilidad **mysqldump** realiza el volcado de bases de datos y puede utilizarse para transportar datos de una base a otra que también entienda SQL. Sin embargo, el archivo debe ser editado antes de utilizarse, ya que algunas opciones

son exclusivas de MySQL. Por lo general, basta con eliminar el tipo de tabla que se especifica al final de un comando **create table**.

El siguiente comando realiza el vaciado completo de la base de datos *demo*:

```
$ mysqldump demo > demo.sql
```

En algunos casos, los comandos **insert** son suficientes y no necesitamos las definiciones de las tablas.

El siguiente comando realiza un vaciado de la tabla *proveedores* de la base de datos *demo* filtrando la salida con el comando *grep* de UNIX que selecciona sólo las líneas que contienen la palabra INSERT. De este modo, el archivo *proveedores-insert.txt* contiene exclusivamente comandos **insert**:

```
$ mysqldump demo proveedores | grep INSERT
```

6. Clientes gráficos

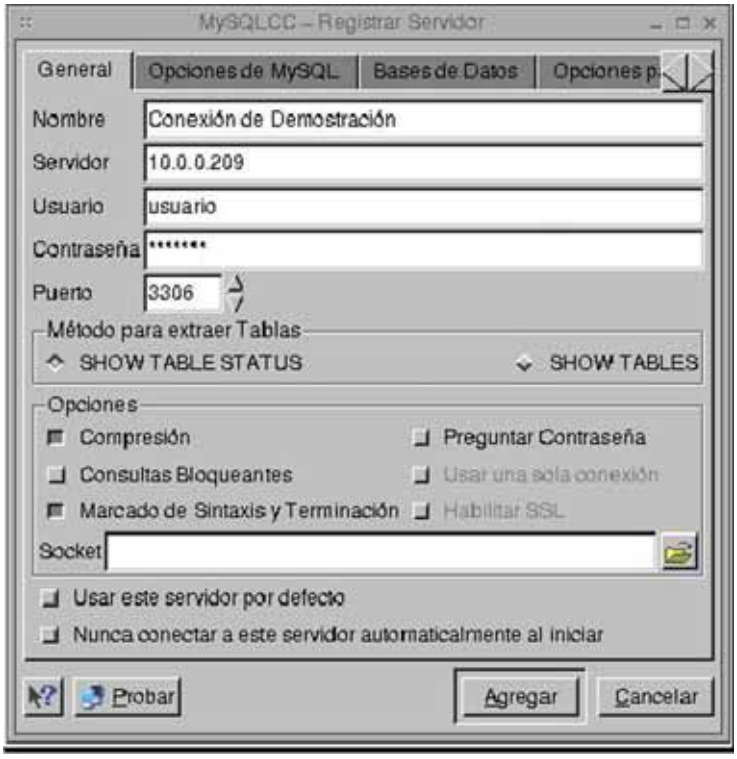
Existen múltiples clientes de entorno gráfico que permiten la interacción con un servidor MySQL. Analizaremos brevemente los que distribuye la empresa MySQL AB (*mysqlcc, mysql-query-browser y mysql-administrator*) y que se pueden descargar del sitio ofical www.mysql.com.

Nota

Actualmente, *mysqlcc* se ha dado por obsoleto en favor de los otros dos.

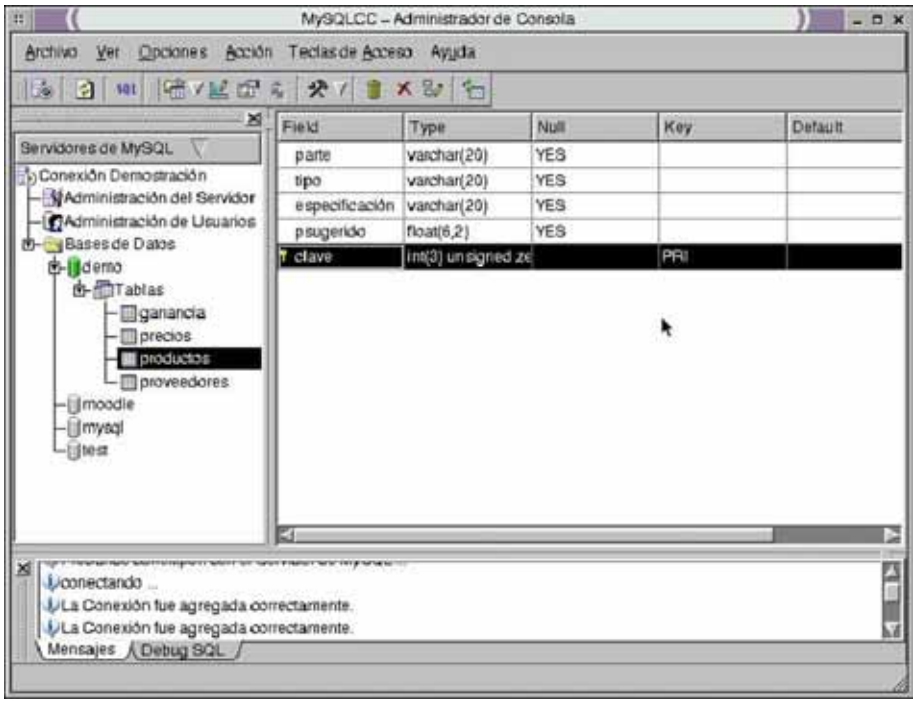
6.1. mysqlcc

Al ejecutarse por primera vez abrirá el diálogo que permite realizar el registro de un nuevo servidor MySQL:



En la ventana principal se pueden apreciar los servidores registrados, que en este caso es solamente uno.

Con el botón derecho del ratón sobre "Conexión de Demostración", se puede activar la conexión. Después de eso, *mysqlcc* muestra las propiedades de los elementos de la base de datos.



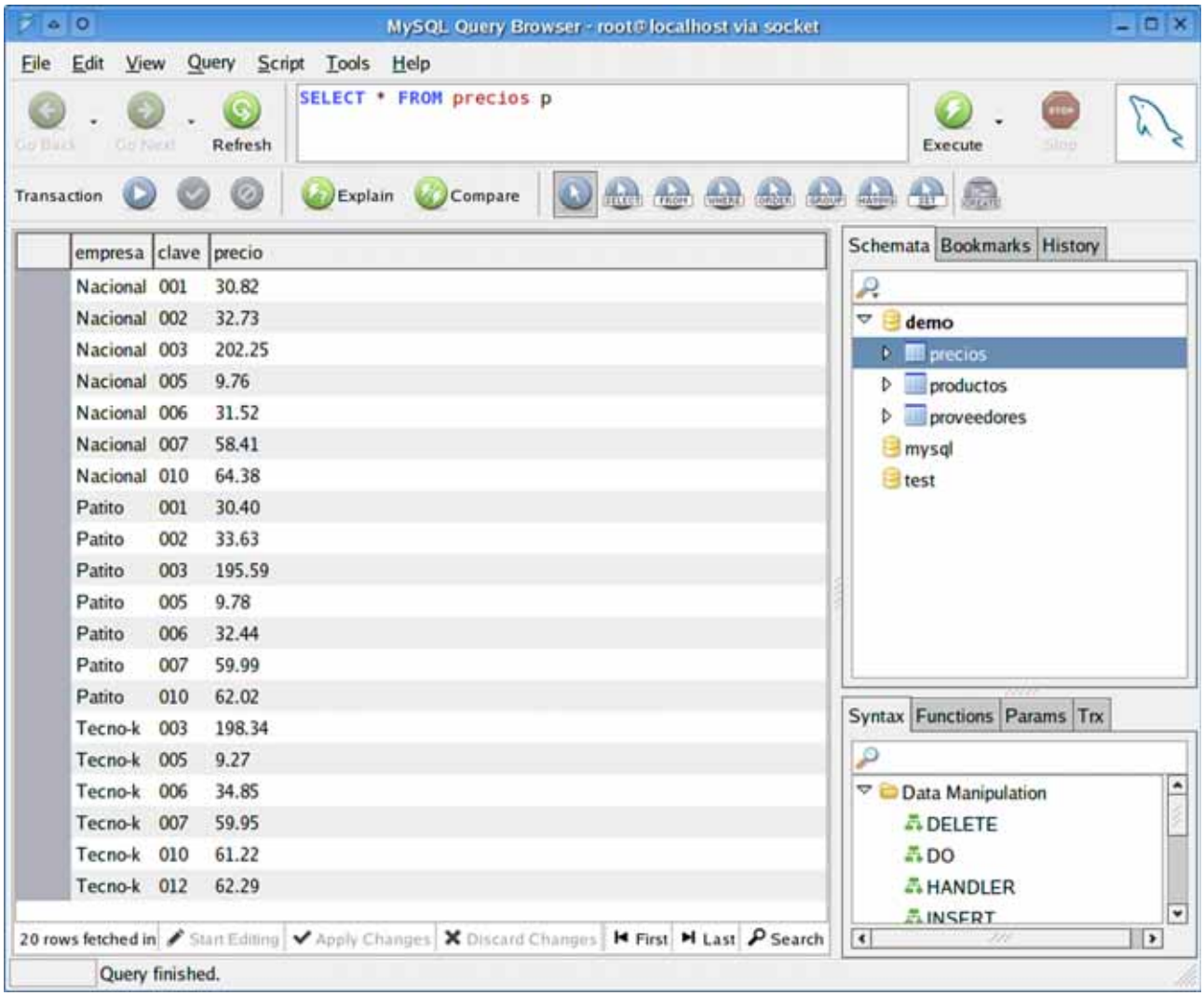
Ahora ya estamos en disposición de realizar consultas SQL con Ctrl-Q (o haciendo click sobre el icono 'SQL'). Se abrirá una nueva ventana en la que podremos escribir la consulta que, una vez escrita, se ejecutará al teclear Ctrl-E. Los resultados se mostrarán en forma de tabla como en la captura de pantalla anterior.

mysqlcc ofrece múltiples opciones para realizar inserciones, eliminaciones, configurar teclas de acceso rápido y una serie de características de uso muy intuitivo. También ofrece prestaciones para exportar el resultado de una consulta a un fichero de texto.

6.2. mysql-query-browser

Tanto *mysql-query-browser* como *mysql-administrator* comparten la información relativa a las conexiones almacenadas. La pantalla inicial nos permitirá seleccionar una existente o configurar una nueva:

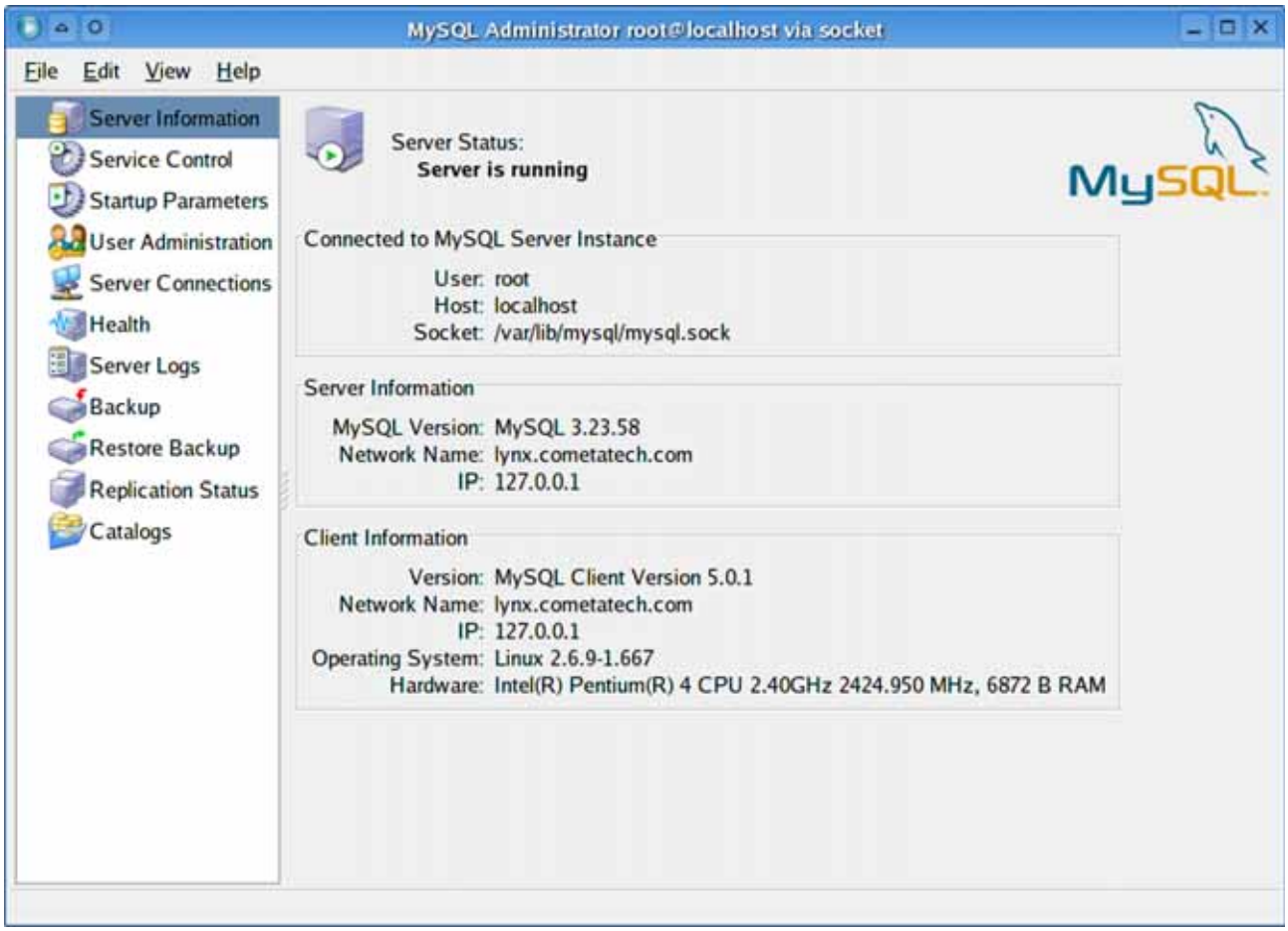
El aspecto de este programa es mejor que el de *mysqlcc* y ofrece prestaciones de ayuda en la generación de consultas, favoritos, marcadores, accesos rápidos a *EXPLAIN*, etc.



Asimismo, ofrece varios formatos de exportación de los resultados de una consulta y más facilidades a la hora de navegar por los resultados.

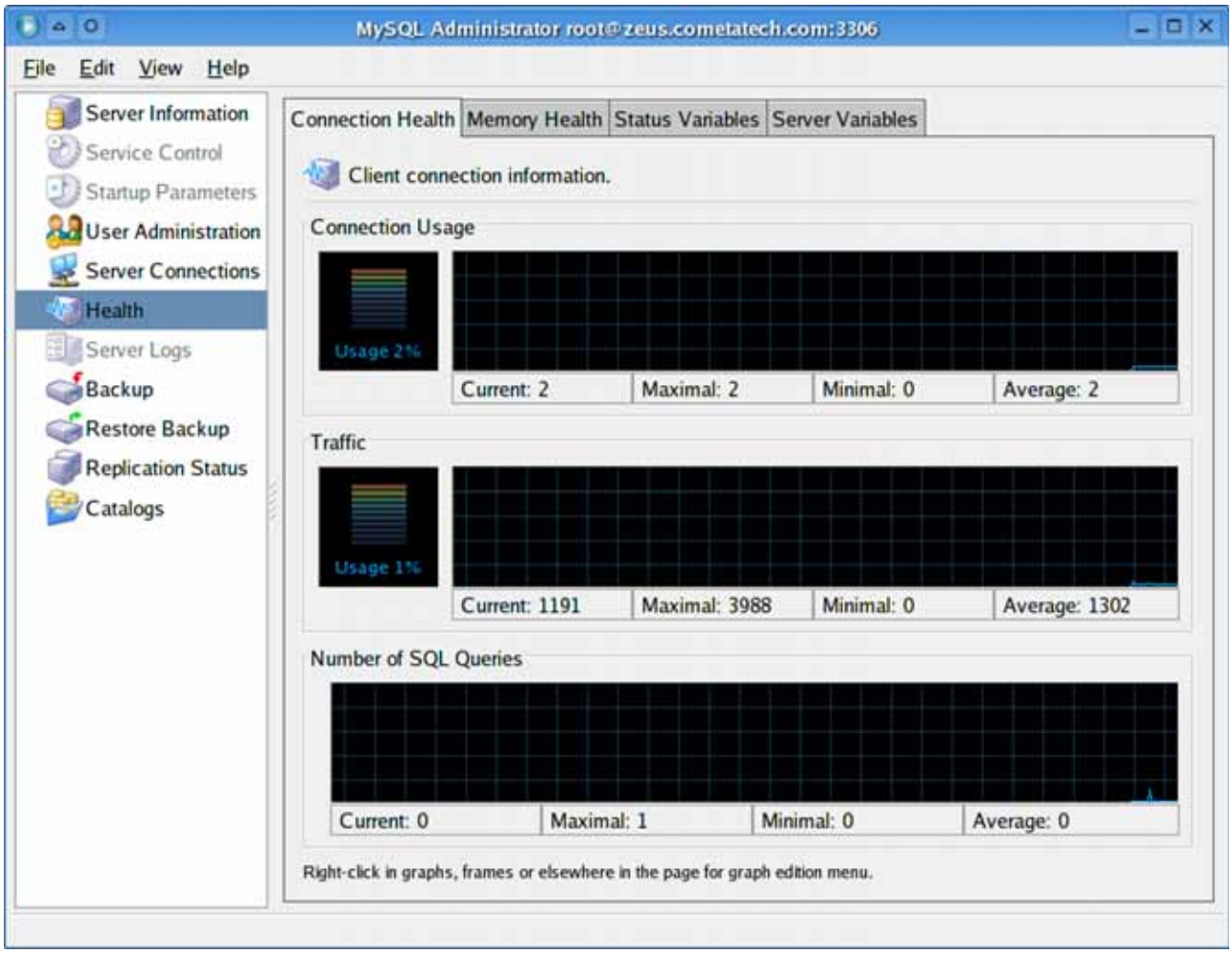
6.3. mysql-administrator

Esta novedosa herramienta es extremadamente potente y completa en cuanto a tareas de administración se refiere.



mysql-administrator permite, entre otras cosas:

- Encender y parar el SGBD.
- Gestionar el fichero de configuración */etc/my.cnf* de forma gráfica.
- Gestionar usuarios y privilegios.
- Monitorizar el uso del gestor que se está haciendo del mismo, el número de conexiones, consultas simultáneas y todo tipo de información estadística.
- Consultar los ficheros de registro (*log*) del servidor.
- Gestionar copias de seguridad.
- Gestionar la replicación de bases de datos.
- Crear y borrar bases de datos (SCHEMA).


MySQL Administrator root@zeus.cometatech.com:3306
File Edit View Help
Server Information
Service Control
Startup Parameters
User Administration
Server Connections
Health
Server Logs
Backup
Restore Backup
Replication Status
Catalogs
Connection Health
Memory Health
Status Variables
Server Variables
Client connection information.
Connection Usage
Usage 2%
Current: 2
Maximal: 2
Minimal: 0
Average: 2
Traffic
Usage 1%
Current: 1191
Maximal: 3988
Minimal: 0
Average: 1302
Number of SQL Queries
Current: 0
Maximal: 1
Minimal: 0
Average: 0
Right-click in graphs, frames or elsewhere in the page for graph edition menu.

Resumen

MySQL es un SGBD relacional de fácil uso y alto rendimiento, dos características muy valiosas para un desarrollador de sistemas: su facilidad de uso permite la creación de bases de datos con rapidez y sin muchas complicaciones, y su alto rendimiento lo hace sumamente atractivo para aplicaciones comerciales importantes o portales web de mucho tráfico. Si a ello le añadimos la disponibilidad de código y su licencia dual, se comprende que MySQL sea atractivo y accesible para todo el mundo.

Estos atributos tienen sus costes: mantenerlo con un alto rendimiento hace algo más lento su desarrollo, por lo que no es el más avanzado en cuanto a prestaciones y compatibilidad con estándares. MySQL carece de características que muchos otros SGBD poseen. Pero no se debe olvidar que está en continuo desarrollo, por lo que futuras versiones incluirán nuevas características. Por supuesto, para MySQL es más importante la eficiencia que incluir prestaciones sólo por competir o satisfacer a algunos usuarios.

Bibliografía

DuBois, P. (2003). *MySQL Second Edition: The definitive guide to using, programming, and administering MySQL 4 databases* Indianapolis: Developer's Library.

Manual de MySQL de la distribución (accesible en: http://dev.mysql.com/doc/).

Silberschatz, A.; Korth, H.; Sudarshan, S. (2002) *Fundamentos de Bases de Datos* (4.ª ed.). Madrid: McGraw Hill.