3. ESTUDIO TEÓRICO DE MYSQL

3.1. INTRODUCCIÓN

MySQL es un sistema de administración de bases de datos (DBMS) relacional *Open Source* y de licencia pública. Fue desarrollado por TcX, una firma consultora sueca. Tomando algunos conceptos de otro sistema de administración de bases de datos llamado mSQL, crearon un producto rápido, confiable y muy flexible. Inicialmente, los principales usuarios de MySQL eran universidades, proveedores de servicios de Internet y organizaciones no lucrativas, pero hoy en día su uso se ha extendido mucho más allá.

La razón de esta popularidad fue el gran crecimiento del movimiento *Open Source*, que puso a disposición de los administradores de bases de datos una herramienta de bajo coste para la gestión de sus bases de datos, que ofrece unas prestaciones realmente buenas. Su funcionamiento es simple, los usuarios se conectan al servidor de bases de datos a través de una aplicación cliente, el servidor de la base de datos consulta con su base de datos y devuelve una respuesta al usuario.

MySQL es un sistema gestor de bases de datos multiusuario, es decir, permite el acceso de varios usuarios al mismo tiempo. En concreto, MySQL puede tener hasta 4000 usuarios simultáneos, dependiendo de la máquina en la que corra el servidor. Por lo general, la mayoría de los sistemas de bases de datos permiten menos usuarios a la vez. Esto implica que el servidor MySQL es un servidor multiprocesos, es decir, que cada vez que alguien establece una conexión con el servidor, el programa servidor crea un subproceso para manejar la solicitud del cliente. Esto conlleva que el servidor sea extremadamente rápido.

Uno de los requerimientos que se le exige a un buen sistema de gestión de bases de datos es la seguridad. Esto se estudiará más adelante, pero se puede avanzar que MySQL posee excelentes características en cuanto a seguridad se refiere.

MySQL no es soportado sólo por Linux, sino que además se puede utilizar en otras plataformas (p.ej. Windows), lo que hace que sea un sistema fácil de migrar. Además, cuenta con bastantes interfaces a nivel de aplicación en diferentes lenguajes, como C, Perl, ODBC, JDBC, etc. MySQL es muy estable en estas plataformas, y como prueba de ello se tiene que actualmente es uno de los sistemas gestores de bases de datos más utilizados.

Por último, sólo faltaría destacar que posee una buena ayuda en línea, hecho que es muy del agrado de los desarrolladores y administradores, sobre todo en un entorno que, como se verá, está basado en línea de comandos.

Hasta aquí se ha visto una pequeña aproximación a MySQL y a algunas de sus características. A continuación, en los siguientes apartados, se podrá ver de forma más exhaustiva estas características y muchas otras, para así hacer una valoración final de este sistema gestor de bases de datos, con objeto de compararlo con el otro sistema a estudio.

3.2. INSTALACIÓN Y PUESTA EN MARCHA.

3.2.1. Descarga.

Como se comentó antes, MySQL es una licencia *Open Source*, con lo que se puede acceder libremente al código para verlo y modificarlo como se desee. Además, a partir de la versión 3.23.19 la licencia de MySQL es del tipo GPL, lo que permite usar MySQL en cualquier situación, sin importar si se trata de una aplicación comercial o personal. MySQL está disponible para muchos sistemas, que se listan a continuación:

- AIX 4.x, 5.x
- Amiga
- BSDI
- Digital Unix 4.x
- FreeBSD
- HP-UX
- Linux 2.0
- Mac OS
- XNetBSD
- Novell NetWare
- OpenBSD
- OS/2 Warp
- SCO OpenServer
- SCO UnixWare
- SGI Iris
- Solaris
- SunOS
- Tru64 Unix
- Windows 9x, Me, NT, 2000, XP y 2003.

Sabiendo esto, desde el sitio Web de MySQL se puede descargar este sistema gestor de bases de datos para instalarlo libremente en cualquiera de los sistemas soportados. No obstante, en la mayoría de distribuciones Linux viene como paquete RPM, con lo cual se puede instalar a la par que se instala el Sistema Operativo Linux o, si éste ya está instalado, se podrá instalar MySQL a partir de los CDs de instalación de Linux. Esta es la manera más fácil de hacerlo.

3.2.2. Instalación de la aplicación.

Es recomendable que se inicie la sesión en Linux como usuario *root*, para así evitar cualquier posible problema con los permisos. Una vez que se tiene el archivo con extensión .tar.gz se procede a desempaquetar este archivo y a instalar la aplicación. Es recomendable que se desempaque este archivo en el directorio /usr/local, ya que todos los valores predeterminados apuntan a esta ruta.

Una vez hecho esto, ya se está en disposición de poner en marcha MySQL en el equipo. Cabe destacar que si se hubiera hecho la instalación desde alguna distribución de Linux, estas habrían copiado los archivos necesarios en /usr/bin, con lo cual se pueden ejecutar los comandos directamente en la línea de comandos, sin necesidad de estar en ese directorio.

A continuación, para completar la instalación de MySQL, lo que se debe hacer es crear las tablas de permiso de acceso. Para ello, se debe ejecutar el siguiente comando desde el terminal de Linux:

```
mysql_install_db
```

Se verá pasar mucha información por pantalla. Lo que hace esta aplicación es crear la tabla de permisos de acceso para MySQL. Esta tabla determina quienes pueden conectarse a la base de datos.

Todas las bases de datos que se creen estarán en el directorio /var/lib/mysql. Si se observa este directorio, se puede ver que hay dos bases de datos ya creadas, que son *mysql* y *test*. Estas bases de datos sirven para la gestión de usuarios y realización de pruebas, y se estudiarán más adelante.

En toda esa información que se mostraba por pantalla al ejecutar *mysql_install_db*, se recuerda que es un buen momento para establecer la contraseña principal de la base de datos *mysql*. Para hacer esto se debe teclear lo siguiente desde el *shell* de Linux:

mysqladmin -u root password *nuevopassword*

Nótese que los privilegios de la base de datos y los del sistema de archivo son dos cosas diferentes, es decir, que un usuario de Linux no tiene por qué ser un usuario de MySQL, a no ser que se cree explícitamente.

Como puede verse, el procedimiento de instalación es relativamente sencillo. Además, el fabricante proporciona manuales bastante detallados para la instalación en cada una de las plataformas soportadas.

3.2.3. Inicio y detención del servidor.

MySQL se ejecuta como servicio o *daemon*. Un servicio o *daemon* es un programa que se ejecuta continuamente en segundo plano. El programa servidor será *mysqld*. Su propósito es esperar a que alguien se conecte a él y le envíe una solicitud, para que este responda enseguida a esta. El comando para poner en marcha el servidor es el siguiente:

mysqld_safe &

Este comando arranca el servidor. El carácter “&” fuerza al programa a ejecutarse en segundo plano. Para comprobar que el servidor está en marcha se puede teclear el siguiente comando:

mysqladmin –p ping

Se pedirá la clave de usuario, se introduce, y si el servidor esta en marcha, se obtendrá como respuesta el mensaje “*mysqld is alive*”. Esto también se podría haber hecho viendo los procesos del sistema con el monitor del sistema.

Para detener el servidor se emplea también el comando *mysqladmin* de la siguiente forma:

mysqladmin –p shutdown

De nuevo se solicitará la clave de usuario. Si todo es correcto, se detiene la ejecución del servidor.

3.3. USO BÁSICO DE MySQL.

Como antes se dijo, MySQL es un sistema de gestión de bases de datos basado en el modelo cliente/servidor. Siempre que se quiera trabajar con MySQL tendrá que estar activo el servidor o *daemon* MySQL. En este apartado se verán aspectos básicos como la creación de bases de datos, creación de tablas, edición de bases de datos, etc. Aquí es donde se observará uno de los puntos débiles de MySQL, pues todas estas tareas se realizan desde una línea de comandos, denominada "monitor *mysql*". Esto puede ser un claro inconveniente para aquellos que estén acostumbrados a trabajar con otras bases de datos que posean una clara e intuitiva interfaz gráfica de usuario (GUI). No obstante, más adelante se verán algunos métodos y herramientas útiles para paliar esta desventaja.

3.3.1. El monitor *mysql*.

Una vez que el servidor MySQL está ejecutándose, se esta en disposición de trabajar con la bases de datos. Para ello, MySQL proporciona el monitor *mysql*, una herramienta en línea de comandos que permite administrar bases de datos. Para arrancar el monitor, el comando empleado es el siguiente:

mysql –p

Se pedirá la contraseña, y tras introducirla se verá lo siguiente:

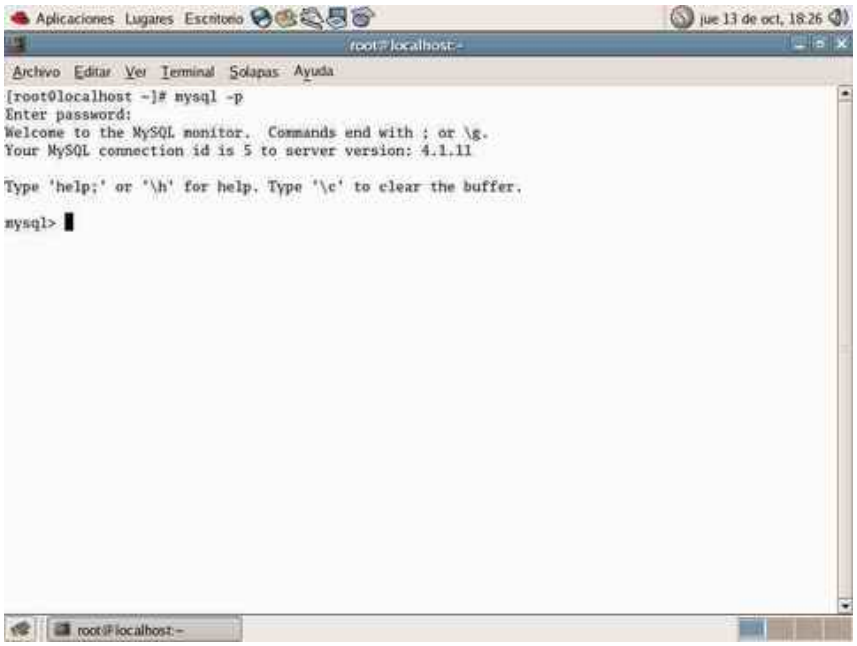

Figura 3.1. *El monitor MySQL*

Para conectar como un usuario concreto se deberá indicar explícitamente. De igual forma, si se desea realizar la conexión a la base de datos desde otra máquina que no sea la que ejecuta el servidor MySQL (*localhost*) se deberá indicar también el nombre de host. Así, de una forma más genérica, se tiene que la instrucción para la conexión al servidor MySQL es:

mysql –h *nombre_host* –u *nombre_usuario_mysql* –p

Esta interfaz basada en línea de comandos no es sólo exclusiva para MySQL bajo Linux, sino que en las demás plataformas también se adopta esta forma. Para un usuario acostumbrado a trabajar en entornos gráficos claros e intuitivos, esta interfaz en línea de comandos le puede resultar incómoda, y puede suponerle un esfuerzo adicional de aprendizaje.

Desde la línea de comandos del monitor, se puede utilizar cualquier comando del lenguaje SQL (Lenguaje de Consultas Estructurado). En este aspecto, se podría entablar una discusión acerca de la línea de comandos. Si bien a priori puede resultar una debilidad el uso de la línea de comandos, algunos podrían argumentar que esto es una ventaja para administrar de manera remota bases de datos rápida y eficientemente, sobre todo en líneas de baja velocidad. Pero es cierto que hoy en día, prácticamente en cualquier entorno empresarial, e incluso en entornos domésticos se poseen líneas de datos con ancho de banda relativamente alto.

Una vez aclaradas las ventajas y desventajas que pueda tener el uso de la línea de comandos, lo siguiente será ver algunos comandos básicos que permitan operar con bases de datos. Cabe destacar que en la línea de comandos, todas las instrucciones deben terminar con un punto y coma (;) o con /g. Si no se terminan los comandos con estos caracteres, se pasará a la siguiente línea. Esta característica puede resultar útil para facilitar la legibilidad en instrucciones largas. Otra característica interesante es la tecla de repetir (flecha arriba). MySQL utiliza un histórico diferente al del sistema operativo, de tal forma que las únicas líneas rescatadas son las tecleadas desde la línea de comandos de MySQL.

3.3.2. Comandos básicos.

A continuación se verán una serie de comandos básicos para manejar bases de datos. Antes de nada hay que comentar que los comandos no son "*case-sensitive*", es decir, que no distinguen entre mayúsculas y minúsculas. No obstante, es buena costumbre escribir los comandos en mayúscula por cuestiones de legibilidad. Sin embargo, los nombres de los elementos que se traten (bases de datos, tablas, columnas, etc.) si son sensibles al uso de mayúsculas y minúsculas. A continuación se muestran los comandos más importantes.

Help
Muestra ayuda sobre los comandos disponibles. Si se teclea algún comando después de la palabra "*help*" se mostrará ayuda sobre ese comando en concreto.

SHOW DATABASES;
Permite ver las bases de datos existentes en el servidor

USE *nombre_bd*;
Sirve para seleccionar una base de datos específica, cuyo nombre viene dado por *nombre_bd*.

SHOW TABLES FROM *nombre_bd*;
Se utiliza para ver el esquema o estructura de una base de datos. Proporciona un listado de todas las tablas de la base de datos dada. Si no se especifica ningún nombre de base de datos, mostrará las tablas de la base de datos que este en uso.

SHOW COLUMNS FROM *nombre_tabla*;
Muestra en detalle todos los campos de una tabla dada. La base de datos que contiene dicha tabla debe estar en uso (ver comando USE). Se despliega la descripción de los campos de la tabla de la base de datos seleccionada.

CREATE DATABASE *nombre_bd*;
Crea una base de datos cuyo nombre se especifica en *nombre_bd*. Hay que comentar que crear una base de datos no la selecciona para su uso.

DROP DATABASE *nombre_bd*;
Esta instrucción borra la base de datos especificada en *nombre_bd* y todo su contenido. Hay que tener mucho cuidado con este comando, pues no pide confirmación para borrar la base de datos en cuestión, sino que la borra directamente.

CREATE TABLE;
La creación de tablas en MySQL es relativamente fácil. Hay varias formas de crear una tabla. Una de ellas es introduciendo la siguiente sintaxis (hay que tener en cuenta que se debe estar usando la base de datos en la cual se quiere crear la tabla):

```
CREATE TABLE nombre_tabla (
        Nombre_campo1       TIPO_DATO OPCIONES,
        Nombre_campo2       TIPO_DATO OPCIONES,
        …
        Nombre_campoN       TIPO_DATO OPCIONES
);
```

Donde *nombre_tabla* es el nombre de la tabla que se quiere crear, TIPO_DATO es el tipo de dato que va a almacenar ese campo de la tabla, y OPCIONES son los modificadores de columna adicionales para ese campo. Los tipos de datos y modificadores de columna se verán en un apartado más adelante.

A este comando, se le pueden añadir unas opciones que pueden resultar bastante útiles. Por ejemplo, se puede usar CREATE TEMPORARY TABLE para crear tablas temporales. Estas tablas temporales sólo existen durante la sesión en la que son creadas y desaparecen cuando se cierra la conexión. Sólo pueden ser vistas por la conexión que las ha creado. Otra opción que se puede utilizar es CREATE TABLE IF NOT EXIST, que como su propio nombre indica, sólo creará la tabla en el caso de que no exista. De otra forma no hará nada.

En las últimas versiones de MySQL se introdujo una característica que puede resultar bastante interesante, que es la posibilidad de crear tablas a partir de resultados de consultas. La sintaxis para hacer esto sería:

CREATE TABLE *nombre_tabla* SELECT *consulta* FROM *nombre_tabla2*

Esta orden crearía una tabla denominada *nombre_tabla* con los registros de la tabla *nombre_tabla2* que cumplieran la condición especificada en "*consulta*".

ALTER TABLE;

Esta instrucción permite hacer modificaciones sobre las tablas, tales como cambiar el nombre de una columna, cambiar el tipo de una columna, cambiar el nombre de una tabla, borrar una columna de una tabla y agregar una columna a una tabla. La sintaxis para las instrucciones anteriores es la siguiente:

- Cambiar nombre a una columna:

  ALTER TABLE *nombre_tabla*
  CHANGE *nombre_ant nombre_nuevo* TIPO_DE_DATO;

  Nótese que debe volver a especificar el tipo de datos de nuevo, o se obtendrá un error.

- Cambiar el tipo de una columna:

  ALTER TABLE *nombre_tabla*
  CHANGE *nombre_campo nombre_campo* TIPO_DE_DATO_NUEVO;

- Cambiar el nombre de una tabla:

  ALTER TABLE *nombre_tabla_ant* RENAME *nombre_tabla_nuevo*;

- Borrar una columna de una tabla:

  ALTER TABLE *nombre_tabla* DROP *nombre_columna*;

- Agregar una columna a una tabla:

  ALTER TABLE *nombre_tabla* ADD *nombre_columna* TIPO_DATO;

DROP TABLE *nombre_tabla*;
Este comando permite eliminar una tabla de una base de datos. Para ello, la base de datos que contiene la tabla a eliminar debe estar en uso. Al igual que con las demás órdenes de eliminación, hay que ser precavido al emplearla, puesto que, como se comentó antes, el monitor MySQL no da avisos de alerta.

SELECT.
Las bases de datos se pueden consultar con una selección relacional normal. Para ello se usa la declaración SQL SELECT. La declaración se divide en una lista destino (la parte que lista los atributos que han de ser devueltos) y una cualificación (la parte que especifica cualquier restricción). La sintaxis completa es la siguiente:

SELECT
[ALL | DISTINCT | DISTINCTROW]
[HIGH_PRIORITY]
[STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
select_expr, ...
[INTO OUTFILE '*nombre_fich*' *export_options* | INTO DUMPFILE ' *nombre_fich*' ']

[FROM *referencias_de_tablas*
[WHERE *condicion*]
[GROUP BY {*nombre_col* | *expr* | *posicion*}[ASC | DESC], ... [WITH ROLLUP]]
[HAVING *definicion*]
[ORDER BY { *nombre_col* | *expr* | *posicion* }[ASC | DESC] , ...]
[LIMIT {[offset,] *row_count* | *row_count* OFFSET *offset*}]
[PROCEDURE *nombre_procedimiento*(*lista_argumentos*)]
[FOR UPDATE | LOCK IN SHARE MODE]]

Una de las desventajas de MySQL con respecto a otros sistemas gestores de bases de datos esta en que MySQL no tiene soporte completo para subconsultas. Mediante el uso de subconsultas se puede seleccionar un conjunto de resultados que han sido generados a partir de otra consulta, todo ello con una sola instrucción SQL.

3.3.3. Modo *batch*.

En la sección anterior, se vio el uso de algunas instrucciones básicas en la línea de comandos. Existe otro modo de ejecutar estas instrucciones. Este modo se denomina modo *batch*, en el cual, se almacenan en un archivo de texto todas las instrucciones que se desean ejecutar, con la misma sintaxis con la que se escribirían en la línea de comandos del monitor MySQL. Una vez que se tiene este archivo de texto, desde el terminal de Linux, se podrá ejecutar el siguiente comando:

mysql –p *fichero_batch*

El fichero *fichero_batch* se puede editar con cualquier editor de texto. Esto puede resultar bastante cómodo, por ejemplo, a la hora de crear nuevas base de datos, puesto que se puede tener una plantilla para hacer esto de una manera más cómoda que introduciendo en el monitor uno a uno los comandos.

3.3.4. Índices.

Los índices son elementos de la base de datos que facilitan el acceso a lo datos. Colocar un índice en la columna adecuada puede acelerar notablemente la velocidad de una consulta. Los índices funcionan como separadores alfabéticos en un campo. Pero hay que tener cuidado también con su uso, puesto que si se emplean demasiados índices se podría obtener un efecto adverso ralentizando la velocidad en las consultas. Otro efecto negativo es que agregar una fila dentro de una columna indexada es más lento. En MySQL se recomienda indexar las columnas que se utilizan mas a menudo, pero no se recomienda el uso de muchos índices. Por tanto, esto se puede considerar como una desventaja de MySQL, El número máximo de índices que se pueden utilizar en una misma tabla de MySQL es de 16. También cabe comentar que los índices trabajan bien en columnas que contienen datos únicos.

Cuando se declara una clave como primaria, MySQL crea automáticamente un índice. La sintaxis para crear un índice en una columna es la siguiente:

CREATE INDEX *nombre_indice* ON *nombre_tabla*(*lista_columnas*);

Cada índice tendrá un nombre, que se establecerá en el campo *nombre_indice*. Los demás parámetros hablan por si solos. En el campo *lista_columnas* se introducirán las columnas a partir de las cuales se va a crear el índice (recuérdese que un índice puede estar formado por varias columnas). Otra forma de crear un índice es con la instrucción ALTER TABLE. La forma de hacerlo sería la siguiente:

ALTER TABLE *nombre_tabla* ADD INDEX (*lista_columnas*);

La misma sintaxis puede usarse para añadir una clave primaria a una tabla que no la tiene, sin más que sustituir “ADD INDEX” por “ADD PRIMARY KEY”. Recuerde que cada tabla sólo puede tener una clave primaria.

Para borrar índices se dispone de dos instrucciones igual de sencillas que las anteriores. Estas instrucciones son:

DROP INDEX (*nombre_indice*) ON *nombre_tabla* ;
ALTER TABLE *nombre_tabla* DROP INDEX (*nombre_indice*) ;

De igual manera que al crearlo, también se puede borrar la clave principal de una tabla de la siguiente forma:

ALTER TABLE *nombre_tabla* DROP PRIMARY KEY;

La mayoría de los índices en MySQL emplean el método de acceso B-Tree, con la excepción de los índices espaciales, que emplean índices R-Tree, y de las tablas almacenadas con el motor de almacenamiento MEMORY (HEAP), que aparte del método B-Tree también soportan el método de acceso Hash.

3.4. USUARIOS.

Como se comentó anteriormente, MySQL es una base de datos multiusuario, es decir, que permite que varios usuarios se conecten a la base de datos para realizar consultas u otro tipo de operaciones. El número máximo de usuarios simultáneos que permite MySQL es de hasta 4000. Pero esto no quiere decir que MySQL sólo pueda tener 4000 usuarios. Puede tener más, pero sólo puede haber un máximo de 4000 usuarios conectados a la vez a la base de datos (esta cifra máxima depende de la máquina en la que se ejecute el servidor). El sistema de usuarios MySQL es independiente del sistema de usuarios de la plataforma Linux. MySQL permite asignar a cada usuario privilegios distintos, que determinan las tareas que podrá realizar cada uno: acceder, modificar, consultar datos y realizar tareas administrativas y de control en la base de datos. Esto se verá con más detalle más adelante cuando se estudien la administración y la seguridad en bases de datos MySQL.

Siempre que se quiera utilizar MySQL se deberá hacer *login* como usuario en el servidor. Al instalar MySQL, por defecto se crea el usuario *root*, sin ninguna contraseña inicialmente. Este el usuario más importante, ya que posee total control sobre las bases de datos.

Para administrar los usuarios, MySQL contiene una base de datos llamada *mysql* que contiene los permisos para todas las bases de datos de MySQL. Esta base de datos está

formada por varias tablas. Con un uso adecuado de estas tablas se puede tener un gran control y variedad de formas de acceso a las bases de datos. Se pueden dar a usuarios permisos totales sobre parte de las bases de datos, o permisos parciales para todas las bases de datos, o descender a detalles como permisos sobre tablas o incluso sobre columnas. En la versión que se ha empleado en este proyecto, la base de datos mysql contiene un total de 15 tablas. De todas estas tablas, las más importantes son tres:

- user. Esta tabla contiene todos los nombres, contraseñas, *hosts* y privilegios de los usuarios de MySQL.
- db. Esta tabla contiene todos los usuarios, bases de datos y nombres de *hosts* de MySQL.
- host. Esta tabla contiene todos los nombres de *hosts*, bases de datos y privilegios de MySQL.

Así, de una manera resumida, para que alguien pueda utilizar una base de datos en MySQL, el nombre de la máquina donde está conectado debe existir en la tabla *host*, el usuario debe existir en la tabla *user* y la base de datos debe existir en la tabla *db*. Más adelante, cuando se vean los aspectos de seguridad y administración de bases de datos MySQL se estudiará con más detalle este aspecto.

3.5. TIPOS DE DATOS.

MySQL tiene varios tipos de datos que soportan diferentes funciones. Un tipo de datos define las características de los datos que la columna en cuestión almacenará. Básicamente existen cuatro grupos de formatos de datos:

- Formato numérico.
- Formato de carácter o cadena.
- Formato de fecha y tiempo.
- Datos espaciales.

MySQL posee, además de los tipos de datos, unos elementos denominados modificadores de columna. Los modificadores de columna ayudan a definir atributos adicionales de las columnas.

3.5.1. Numérico.

Los tipos numéricos almacenan sólo números. Los tipos numéricos pueden clasificarse en diferentes rangos. Los diferentes tipos numéricos usan una cantidad de memoria diferente. La razón es porque cada tipo tiene un rango diferente. Los nombres de los diferentes tipos numéricos y sus rangos se muestran en la tabla 3.1.

Nombre del tipo	Rango de valores (con signo)	Rango de valores (sin signo)
TINYINT[(M)]	-128 a 127	0 - 255
BIT	Sinónimo a TINYINT(1)	
BOOL	Sinónimo a TINYINT(1). FALSO (0) / VERDADERO (valor distinto de 0)	
SMALLINT[(M)]	-32768 a 32767	0 – 65535
MEDIUMINT[(M)]	-88388608 a 8388607	0 – 16777215
INT[(M)]	-2147483648 a 2147483647	0 – 4294967295

Nombre del tipo	Rango de valores (con signo)	Rango de valores (sin signo)
BIGINT[(M)]	-9223372036854775808 a 9223372036854775807	0 – 18446744073709551615
FLOAT[(M,D)]	-3.402823466E+38 a -1.175494351E-38, 0, y 1.175494351E-38 a 3.402823466E+38	
DOUBLE[(M,D)]	-1.7976931348623157E+308 a -2.2250738585072014E-308, 0, y 2.2250738585072014E-308 a 1.7976931348623157E+308	
DECIMAL[(M[,D])]	El mismo que para DOUBLE	

Tabla 3.1. *Tipos de datos numéricos en MySQL*

En los tipos de datos anteriores se ha usado la siguiente notación:

- M: Indica la máxima anchura para la representación. La máxima es 255. En el caso de FLOAT y DOUBLE es el número máximo de dígitos de la parte entera, y en el caso de DECIMAL es el número total de dígitos.
- D: Aplicado a los tipos en punto fijo y punto flotante. Indica el número de dígitos que siguen al punto decimal. El máximo valor posible es 30, pero se recomienda que no sea mayor que M-2.
- []: Estos corchetes indican que los parámetros incluidos dentro de ellos son opcionales.

Los valores M y D anteriores pueden servir para limitar el espacio que ocupan los datos en memoria.

Cabe decir que todos los tipos anteriores se pueden definir como UNSIGNED, es decir, con un rango de valores positivos únicamente. No obstante, en los tipos FLOAT; DOUBLE y DECIMAL, si se declaran sin signo, el límite superior queda intacto, es decir, que es el mismo que si se declararan con signo.

3.5.2. Fecha y hora.

MySQL tiene algunos tipos de datos para el manejo de la información de fecha y hora. En general, MySQL aceptará varios formatos cuando se proporcionen datos para colocar en un campo. Sin embargo, la salida de la fecha y de la hora siempre estará estandarizada y aparecerá en un formato predecible. Todos los tipos de datos de fecha y hora tienen un intervalo de valores válidos, y un valor “cero”, con el que será establecido un campo si se intenta colocar un valor ilegal en él.

Los formatos de hora tienen un orden intuitivo. Las fechas, sin embargo, siempre son generadas con el año a la izquierda, el mes y después el día. Este formato de fecha no es el que se suele emplear en nuestro país (del tipo DD/MM/AAAA).

Al dar salida a la fecha y hora, se tiene la opción de pedir que MySQL proporcione los datos ya sea como datos de cadena o numéricos, aún cuando se trate de la misma información. El formato empleado dependerá del contexto en el que sea usado en SQL.

MySQL es bastante flexible al aceptar datos de fecha y hora: por ejemplo, para MySQL la fecha 2005-10-19 significa lo mismo que 2005/10/19, 5+10+19 o incluso 20051019.

MySQL realiza una verificación parcial de la información de fecha y hora. Por ejemplo, espera que los días se encuentren entre 1 y 31, y los meses entre 1 y 12. Sin embargo, un punto en contra es que no verifica estrictamente si una fecha específica puede existir realmente. Por ejemplo, no rechazará el 30 de Febrero. No obstante se ha optado por este método porque lo hace más eficiente en cuanto a la introducción de datos, pero deja a la aplicación la responsabilidad de asegurar que las fechas que se introduzcan son válidas.

MySQL tiene una variedad de tipos de datos relacionados con la fecha para facilitar el manejo de información de fecha y hora. Los tipos de formato se muestran en la tabla 3.2:

Tipo de datos	Formato estándar	Valor de cero
DATETIME	AAAA-MM-DD HH:MM:SS	0000-00-00 00:00:00
DATE	AAAA-MM-DD	0000-00-00
TIME	HH:MM:SS	00:00:00
YEAR	AAAA	0000
TIMESTAMP	variable	00000000000000 (en su mayor longitud)

Tabla 3.2. *Tipos de datos fecha/hora en MySQL.*

El tipo de datos TIMESTAMP es un útil formato de campo mediante el cual una columna se establecerá con la fecha y hora actuales siempre que la fila sea actualizada o insertada en una tabla. Por tanto, este tipo de dato proporciona una marca de “última utilización” sin tener que establecerlo cada vez que se cambie algún dato en la tabla. TIMESTAMP sólo puede manejar fechas en el intervalo de 1970 a 2037, almacenando los datos con una resolución máxima de un segundo. Aún cuando un campo TIMESTAMP siempre se mantiene internamente en la misma forma, puede tener varios formatos externos. Los formatos que se ofrecen se muestran en la tabla 3.3:

Tipo de datos	Formato desplegado
TIMESTAMP(14)	AAAAMMDDHHMMSS
TIMESTAMP(12)	AAMMDDHHMMSS
TIMESTAMP(10)	AAMMDDHHMM
TIMESTAMP(8)	AAAAMMDD
TIMESTAMP(6)	AAMMDD
TIMESTAMP(4)	AAMM
TIMESTAMP(2)	AA

Tabla 3.3. *Formatos TIMESTAMP.*

No obstante, en este tipo de datos se puede escribir nuevos datos con sólo especificarlos de manera explícita. Los valores no válidos serán rechazados y reemplazados con el valor de “cero”.

MySQL permite transferir datos de un tipo a otro. Por ejemplo, se puede tener un campo DATETIME que ha registrado una hora exacta. Después de un tiempo, tal vez se desee transferir estos datos a otra tabla pero solamente con la fecha. Si se tienen datos de una columna DATETIME o TIMESTAMP y se desean asignarlos a una columna DATE, la información relativa a la hora simplemente se perderán porque el tipo DATE no tiene capacidad para almacenar la información de la hora. Por el contrario, si se tiene un valor DATE y se desea transferirlo a una columna DATETIME o TIMESTAMP, MySQL insertará 00:00:00 en la parte de la hora, porque esos datos no existían anteriormente.

3.5.3. Carácter o cadena.

Otro gran grupo de tipos de datos es el tipo de carácter o cadena. En un tipo de datos cadena se puede almacenar cualquier valor. El tamaño es un factor determinante para el tipo de cadena a emplear. El tamaño máximo se lista en la tabla 3.4. El espacio de almacenamiento para cada tipo está determinado por la longitud de la cadena.

Nombre del tipo	Tamaño máximo
CHAR(X)	255 bytes
VARCHAR(X)	255 bytes
BINARY(X)	255 bytes
VARBINARY(X)	255 bytes
TINYTEXT	255 bytes
TINYBLOB	255 bytes
TEXT	65535 bytes
BLOB	65535 bytes
MEDIUMTEXT	1.6 MB
MEDIUMBLOB	1.6 MB
LONGTEXT	4.2 GB
LONGBLOB	4.2 GB

Tabla 3.4. *Tipos carácter/cadena en MySQL.*

Además de los anteriores, existen dos tipos especiales: ENUM y SET. El tipo ENUM es una lista numerada. Esto significa que una columna con este tipo de datos sólo puede almacenar uno de los valores declarados en una lista dada. La columna puede contener solamente uno de esos valores o NULL. Si se trata de insertar un valor que no esta en la lista, se insertará un espacio. En la lista numerada puede haber hasta 65535 elementos. El tipo SET es muy similar al tipo ENUM. Al igual que ENUM, el tipo SET almacena una lista de valores. La diferencia es que con el tipo SET se puede escoger más de un valor para almacenar en el mismo registro. Un tipo SET puede contener hasta 64 elementos. Los tipos ENUM y SET, aunque se ven y actúan como cadenas, se almacenan realmente como números. Por ejemplo, si se tiene la lista siguiente: SET('Televisión','Radio','Internet') y se quiere almacenar en un registro el valor 'Radio', lo que realmente se almacenará será el valor 2 (su posición en la lista). Por esta razón se procesan de manera más eficiente que una cadena cualquiera.

3.5.4. Espacio en memoria.

Un factor importante para la elección de uno u otro tipo de dato es el espacio que ocupa éste en memoria. En la tabla 3.5 se muestra el tamaño que ocupan los tipos de datos antes comentados. Hay que recordar que la elección de los tipos de datos adecuados asegura el correcto funcionamiento y mantenimiento de una base de datos.

Tipo de dato	Espacio de memoria requerido
TINYINT[(M)]	1 byte
BIT	1 byte
BOOL	1 byte
SMALLINT[(M)]	2 bytes
MEDIUMINT[(M)]	3 bytes
INT[(M)]	4 bytes

Tipo de dato	Espacio de memoria requerido
BIGINT[(M)]	8 bytes
FLOAT[(M,D)]	4 bytes
DOUBLE[(M,D)]	8 bytes
DECIMAL[(M[,D])]	M+2 bytes, si D>0 M+1 bytes, si D=0 D+2 bytes. Si M<D
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 bytes
CHAR(X)	X bytes, 0 <= X <= 255
VARCHAR(X)	L+1 bytes, donde L<=X y 0 <= X <= 255
BINARY(X)	X bytes, 0 <= X <= 255
VARBINARY(X)	L+1 bytes, donde L <= X y 0 <= X <= 255
TINYTEXT	L+1 bytes, donde L < 2^8
TINYBLOB	L+1 bytes, donde L < 2^8
TEXT	L+2 bytes, donde L < 2^{16}
BLOB	L+2 bytes, donde L < 2^{16}
MEDIUMTEXT	L+3 bytes, donde L < 2^{24}
MEDIUMBLOB	L+3 bytes, donde L < 2^{24}
LONGTEXT	L+4 bytes, donde L < 2^{32}
LONGBLOB	L+4 bytes, donde L < 2^{32}
ENUM('valor1','valor2',…)	1 o 2 bytes, dependiendo del número de valores de la lista (65535 valores como máximo)
SET('valor1','valor2',…)	1, 2, 3, 4, o 8 bytes, dependiendo del número de valores de la lista (64 valores como máximo)

Tabla 3.5. *Tamaño ocupado por los diferentes tipos de datos en MySQL.*

En la tabla anterior, "L" indica la longitud de la cadena TEXT o BLOB.

3.5.5. Modificadores de columna adicionales.

MySQL tiene muchas palabras clave que modifican la forma en que se comporta una columna. Estas palabras clave se denominan "modificadores de columna". Los modificadores de columna disponibles en MySQL y los tipos de datos sobre los que se puede aplicar se muestran en la tabla 3.6:

Nombre del modificador	Tipos aplicables
AUTO_INCREMENT	Todos los tipos INT
BINARY	CHAR, VARCHAR
DEFAULT	Todos, excepto BLOB y TEXT
NOT NULL	Todos los tipos
NULL	Todos los tipos
PRIMARY KEY	Todos los tipos
UNIQUE	Todos los tipos
UNSIGNED	Tipos numéricos
ZEROFILL	Tipos numéricos

Tabla 3.6. *Modificadores de columna en MySQL.*

AUTO_INCREMENT
Este modificador de columna incrementa el valor de una columna añadiendo 1 al máximo valor actual. El valor de una fila recién insertada en una columna

AUTO_INCREMENT inicia en 1 y se incrementa de 1 en 1 por cada registro insertado. Hay que tener en cuenta que si se borra el máximo valor, este no se reutilizará. Para sacar ventaja del modificador de columna AUTO_INCREMENT, se deberá introducir un NULL, 0 o espacio en blanco en el campo AUTO_INCREMENT cuando se inserte una nueva fila. También se puede incluir un número. Si el número ya existe, se obtendrá un error. Si no, será insertado. Si el valor es el mas alto en esa columna, el siguiente será uno mas que el valor actual. Por último, hay que comentar que el modificador AUTO_INCREMENT no se reinicia. Si se alcanza el valor máximo para el tipo de datos que se escogió, el sistema retornará un error.

ZEROFILL
El modificador de columna ZEROFILL se usa para desplegar los ceros a la izquierda de un número de acuerdo con el ancho de la columna.

BINARY
Este modificador hace que los valores almacenados en estos tipos sean tratados como cadenas binarias, haciéndolos sensibles a las mayúsculas y minúsculas.

DEFAULT
El modificador DEFAULT permite especificar el valor de una columna, si no existe alguno. El valor predeterminado de MySQL es NULL para todos los tipos, excepto para los tipos ENUM y SET. Para el tipo ENUM, MySQL utiliza como valor predeterminado el primer valor de la lista numerada, y para el tipo SET utiliza como valor predeterminado una cadena vacía.

NULL / NOT NULL
Estos modificadores especifican si una columna debe tener o no valores.

PRIMARY KEY
PRIMARY KEY es en realidad un índice que debe contener valores únicos. No puede ser NULL. Cada tabla debe tener una clave. MySQL permite crear fácilmente un índice declarando esta clave como clave primaria.

UNIQUE
Este modificador refuerza la regla de que todos los datos dentro de una columna declarada deben ser únicos. Si se trata de insertar un valor que ya existe, MySQL generará un error.

UNSIGNED
El modificador de columna UNSIGNED se emplea con los tipos numéricos. Con este modificador se fuerza a que los valores del campo sean positivos, incrementando por tanto el rango de los valores positivos.

3.5.6. Extensión espacial.

A partir de la versión 4.1, MySQL incluye un paquete de extensiones espaciales, que permiten la generación, almacenamiento y análisis de características geográficas. MySQL implementa las extensiones espaciales siguiendo la especificación del consorcio OpenGIS. Siguiendo esta especificación, MySQL implementa un subconjunto de los tipos de datos geométricos definidos en el estándar SQL. Esta serie de tipos

geométricos se basa en el modelo geométrico de OpenGIS. En este modelo, cada objeto geométrico tiene las siguientes propiedades generales:

- Está asociado con un sistema espacial de referencia, que describe las coordenadas espaciales del objeto determinado.
- Pertenece a alguna clase geométrica.

La jerarquía de objetos geométricos es como sigue:

- Geometry (No instanciable)
  - Point (Instanciable)
  - Curve (No instanciable)
    - LineString (Instanciable)
      - Line
      - LinearRing
  - Surface (No instanciable)
    - Polygon (Instanciable)
  - GeometryCollection (Instanciable)
    - MultiPoint (Instanciable)
    - MultiCurve (No instanciable)
      - MultiLineString (Instanciable)
    - MultiSurface (No instanciable)
      - MultiPolygon (Instanciable)

Sólo es posible crear objetos de las clases instanciables. Todas estas clases tienen una serie de propiedades y una serie de reglas que definen los valores válidos para cada una de ellas. La clase *Point* representa objetos de dimensión cero, *Curve* representa objetos de dimensión 1 y *Surface* representa objetos bidimensionales. La clase *GeometryCollection* contiene una serie de clases de dimensiones 0, 1 y 2.

Para representar los objetos geométricos en las consultas, MySQL proporciona dos formatos de datos:

- Well-Known Text (WKT). Formato en modo texto. Por ejemplo POINT (15 20), LINESTRING(0 0, 10 10, 20 25, 50 60), etc.
- Well-Known Binary (WKB). Formato en modo binario con una estructura predefinida.

Los tipos de datos espaciales soportados por MySQL se muestran en la tabla 3.7.

Tipo Dato	Descripción
GEOMETRY	Almacena valores geométricos de cualquier tipo.
POINT	Almacena un punto.
LINESTRING	Almacena una línea.
POLYGON	Almacena un polígono.
MULTIPOINT	Colección de puntos.
MULTILINESTRING	Colección de líneas.
MULTIPOLYGON	Colección de polígonos.
GEOMETRYCOLLECTION	Colección de elementos geométricos de cualquier tipo.

Tabla 3.7. *Tipos de datos espaciales en MySQL.*

Estos tipos de datos se pueden emplear en cualquier columna de una tabla, teniendo en cuenta que actualmente los tipos de columnas espaciales sólo están disponibles para tablas que empleen el motor de almacenamiento MyISAM. Para popular una tabla con estos tipos de datos, estos deben estar en un formato geométrico interno. Para pasar a este formato a partir de las representaciones WKT y WKB existen una serie de funciones, que se comentarán en un apartado posterior.

MySQL puede también crear índices espaciales usando una sintaxis similar a la empleada para crear índices convencionales, sin más que añadir la palabra clave SPATIAL.

3.6. FUNCIONES INTRÍNSECAS

Casi todas las bases de datos proporcionan al usuario una serie de funciones comunes que permiten llevar a cabo tareas complejas dentro de una consulta. Algunas de estas tareas pueden ser la manipulación de cadenas ó ecuaciones matemáticas complejas. MySQL provee muchas de estas funciones intrínsecas. Además de las funciones incluidas en la distribución de MySQL, se pueden crear funciones definidas por el usuario.

Las funciones básicas de MySQL pueden dividirse en cinco tipos: numéricas o matemáticas, lógicas, de manipulación de cadenas, de fecha/hora y geométricas.

3.6.1. Funciones numéricas.

Las funciones numéricas están compuestas de varias operaciones matemáticas. La forma de una función es muy simple. Tiene un nombre de función seguido de su argumento encerrado entre paréntesis. No hay espacios entre el nombre de la función y el paréntesis. Una función puede tener uno, varios o ningún argumento. También hay que reseñar que las funciones numéricas sólo pueden trabajar con tipos de datos numéricos. A continuación se mostrará una breve descripción de estas funciones.

3.6.1.1.Operaciones aritméticas

Los operadores aritméticos usuales (+, -, * y /) están disponibles. Cabe anotar que en el caso de ‘-’, ‘+’, y ‘*’, el resultado se calcula con precisión BIGINT (64 bits).

3.6.1.2.Funciones matemáticas.

Todas las funciones matemáticas retornan NULL en caso de error. Estas funciones se muestran en la tabla 3.11.

Función	Descripción
Menos unario (-)	Cambia el signo del argumento.
ABS(X)	Retorna el valor absoluto de X.
SIGN(X)	Retorna el signo del argumento como –1, 0 o 1, dependiendo de si X es negativo, cero, o positivo.
MOD(N,M)	Módulo aritmético. Retorna el resto de N dividido entre M.
FLOOR(X)	Retorna el mayor valor entero no mayor que X.
CEILING(X)	Retorna el menor entero no inferior a X.

Función	Descripción
ROUND(X)	Retorna el argumento X, redondeado al entero más cercano.
ROUND(X,D)	Retorna el argumento X, redondeado a un número con D decimales. Si D es 0, el resultado no tendrá punto decimal o parte fraccionaria.
EXP(X)	Retorna el valor de e (la base de los logaritmos naturales) elevado a la potencia de X.
LN(X) / LOG(X)	Retorna el logaritmo natural de X.
LOG(B,X)	Esta función retorna el logaritmo de X para una base arbitraria B.
LOG2(X)	Retorna el logaritmo en base 2 de X.
LOG10(X)	Retorna el logaritmo en base 10 de X.
POW(X,Y)	Retorna el valor de X elevado a la Y-ésima potencia.
PI()	Retorna el valor de PI. El número de decimales visualizado por defecto es 5, pero MySQL utiliza internamente la doble precisión de PI.
COS(X)	Retorna el coseno de X, donde X se da en radianes.
SIN(X)	Retorna el seno de X, donde X se da en radianes.
TAN(X)	Retorna la tangente de X, donde X se da en radianes.
ACOS(X)	Retorna el arco del coseno de X. Retorna NULL si X no se halla en el rango de –1 a 1.
ASIN(X)	Retorna el arco del seno de X. Retorna NULL si X no se halla en el rango de –1 a 1.
ATAN(X)	Retorna el arco de la tangente de X.
ATAN(Y,X) / ATAN2(Y,X)	Retorna el arco tangente de las dos variables X e Y. Esto es similar a calcular el arco tangente de Y/X, excepto en que los signos de los dos argumentos se utilizan para determinar el cuadrante del resultado.
COT(X)	Retorna la cotangente de X.
RAND() / RAND(N)	Retorna un valor de coma flotante aleatorio dentro del rango 0 a 1.0. Si se especifica un valor entero N, se utiliza como semilla (una semilla se utiliza para generar el número aleatorio. La misma semilla devolverá la misma serie de números aleatorios).
LEAST(X,Y,...)	Con dos o más argumentos, retorna el argumento más pequeño.
GREATEST(X,Y,...)	Retorna el argumento de mayor valor.
DEGREES(X)	Retorna el argumento X, convertido de radianes a grados.
RADIANS(X)	Retorna el argumento X, convertido de grados a radianes.
TRUNCATE(X,D)	Retorna el número X, truncado a D decimales. Si D es 0, el resultado no tendrá punto decimal o parte fraccionaria.

Tabla 3.11. *Funciones matemáticas en MySQL.*

3.6.2. Funciones lógicas y condicionales.

A continuación se verá una breve descripción de las funciones que ofrece MySQL para realizar operaciones lógicas y comparaciones condicionales.

3.6.2.1.Operadores lógicos.

Todos los operadores lógicos evalúan a 1 (TRUE), 0 (FALSE) o NULL (Desconocido, lo que en la mayoría de casos equivale a FALSE). Los operadores lógicos disponibles en MySQL se muestran en la tabla 3.12.

Operador	Función
NOT, !	No lógico. Devuelve 1 si el operando al que precede es 0, y en cualquier otro caso devuelve 0. Excepción: NOT NULL devuelve NULL.
OR, ||	O lógico. Devuelve NULL si cualquier operando es NULL.
AND, &&	Y lógico. Devuelve NULL si cualquier operando es NULL.
XOR	Operador O-exclusivo. Para operandos no nulos, devuelve 1 si sólo uno de los operadores es diferente de cero. Devuelve NULL si cualquier operando es NULL.

Tabla 3.12. *Operadores lógicos en MySQL.*

3.6.2.2.Funciones de control de flujo

Las funciones para el control de flujo se muestran en la tabla 3.13.

Expresión	Descripción
IFNULL(*expr1*, *expr2*)	Si *expr1* no es NULL, IFNULL() devuelve *expr2*, y en caso contrario devuelve *expr1*. IFNULL() retorna un valor de cadena o número, dependiendo del contexto en el que se utiliza.
NULLIF(*expr1*,*expr2*)	Si *expr1=expr2* es cierto, retorna NULL, y si no, retorna *expr1*.
IF(*expr1*,*expr2*,*expr3*)	Si *expr1* es TRUE (distinto de cero y no NULL) entonces IF() retorna *expr2*, si no retorna *expr3*. IF() retorna un valor numérico o de cadena, dependiendo del contexto en el que se utilice.
CASE *value* WHEN [*compare-value*] THEN *result* [WHEN [*compare-value*] THEN *result*...][ELSE *result*] END	Retorna *result* cuando *value=compare-value*. Si no se cumple ninguna de las condiciones, devuelve el resultado correspondiente al ELSE.
CASE WHEN [*condition*] THEN *result* [WHEN [*condition*] THEN *result*...] [ELSE *result*] END	Retorna el resultado de la primera condición que es cierta. Si no hubiera ningún resultado coincidente, se retorna el resultado que hay después de ELSE. Si no hay ninguna parte ELSE, se retorna NULL. El tipo de valor de retorno (INTEGER, DOUBLE o STRING) es el mismo que el tipo del primer valor retornado (la expresión después del primer THEN).

Tabla 3.13. *Funciones para el control de flujo.*

3.6.3. Funciones de cadenas y caracteres.

MySQL proporciona una gran variedad de funciones para el tratamiento de cadenas y caracteres. A continuación se listan estas funciones, con una breve descripción de las mismas.

3.6.3.1.Funciones de cadena

Antes de ver el listado de funciones, hay que comentar que las funciones de valor de cadena retornan NULL si la longitud del resultado superara el valor del parámetro de servidor *max_allowed_packet*. Una vez sabido esto, las funciones se detallan en la tabla 3.14.

Función	Descripción
ASCII(*str*)	Retorna el valor del código ASCII del carácter más a la izquierda de la cadena *str*. Retorna 0 si *str* es una cadena vacía. Retorna NULL si *str* es NULL.
ORD(*str*)	Si el carácter más a la izquierda de la cadena *str* es un carácter multi-byte, retorna el código para este carácter, calculado como el valor en código ASCII para sus caracteres constituyentes utilizando la fórmula: ((primer código ASCII)*256+(segundo código ASCII)) [*256+(primer código ASCII)*256]

	Si el carácter más a la izquierda no es un carácter multi-byte, retorna el mismo valor que la función ASCII.
CONV(N,*from_base*,*to_base*)	Convierte números entre diferentes bases
BIN(N)	Retorna una representación de cadena con el valor binario de N, donde N es un entero largo (BIGINT).
OCT(N)	Retorna una representación de cadena con el valor octal de N, donde N es un entero largo (BIGINT).
HEX(X)	Si X es número, retorna una representación de cadena del valor hexadecimal de N, dónde N es un entero largo (BIGINT). Si X es una cadena, retorna cadena hexadecimal de X donde cada carácter en X se convierte a 2 dígitos hexadecimales.
CHAR(N,...)	CHAR() interpreta los argumentos como enteros y retorna una cadena consistente en los caracteres dados por el código ASCII de estos enteros.
CONCAT(*str1*,*str2*,...)	Retorna la cadena que resulta de concatenar los argumentos. Retorna NULL si cualquier argumento es NULL.
CONCAT_WS(*separator*,*str1*,*str2*,...)	Concatenación con separador.
LENGTH(*str*) / OCTET_LENGTH(*str*) / CHAR_LENGTH(*str*) / CHARACTER_LENGTH(*str*)	Retornan la longitud de la cadena *str*.
BIT_LENGTH(*str*)	Retorna la cadena de la cadena *str* en bits.
LOCATE(*substr*,*str*) / POSITION(*substr* IN *str*)	Retorna la posición de la primera ocurrencia de la subcadena *substr* en la cadena *str*. Retorna 0 si *substr* no se halla en *str*
LOCATE(*substr*,*str*,*pos*)	Retorna la posición de la primera ocurrencia de la subcadena *substr* en la cadena *str*, empezando en la posición *pos*. Retorna 0 si *substr* no se halla en *str*.
INSTR(*str*,*substr*)	Retorna la posición de la primera ocurrencia de la subcadena *substr* en la cadena *str*.
LPAD(*str*,*len*,*padstr*)	Retorna la cadena *str*, rellenada por la izquierda con la cadena *padstr* hasta que *str* tenga *len* caracteres de largo. Si *str* es más largo que *len* entonces se recortará para que tenga *len* caracteres.
RPAD(*str*,*len*,*padstr*)	Retorna la cadena *str*, rellenada por la derecha con la cadena *padstr* hasta que *str* tiene *len* caracteres de largo. Si *str* es más larga que *len* entonces será recortada a *len* caracteres.
LEFT(*str*,*len*)	Retorna los *len* caracteres más a la izquierda de la cadena *str*.
RIGHT(*str*,*len*)	Retorna los *len* caracteres más a la derecha de la cadena *str*.
SUBSTRING(*str*,*pos*,*len*) / SUBSTRING(*str* FROM *pos* FOR *len*) / MID(*str*,*pos*,*len*)	Retorna la subcadena de *len* caracteres de largo desde la cadena *str*, empezando desde la posición *pos*.
SUBSTRING(*str*,*pos*) / SUBSTRING(*str* FROM *pos*)	Retorna una subcadena desde la cadena *str* empezando desde la posición *pos*.
SUBSTRING_INDEX(*str*,*delim*,*count*)	Retorna la subcadena desde la cadena *str* antes de *count* ocurrencias del delimitador *delim*.
LTRIM(*str*)	Retorna la cadena *str* con los espacios precedentes eliminados.
RTRIM(*str*)	Retorna la cadena *str* con los espacios finales eliminados.
TRIM([[BOTH|LEADING|TRAILING] [*remstr*] FROM] *str*)	Retorna la cadena *str* con todos los prefijos *remstr* y/o sufijos eliminados. Los especificadores TRAILING, LEADING y BOTH harán que se eliminen los sufijos, los prefijos o ambos respectivamente.
SOUNDEX(*str*)	Retorna una cadena *soundex* de *str*. Dos cadenas que pueden sonar casi igual deberían tener cadenas *soundex* idénticas.
SPACE(N)	Retorna una cadena consistente en N caracteres de espacio.
REPLACE(*str*,*from_str*,*to_str*)	Retorna la cadena *str* con todas las ocurrencias de la cadena *from_str* reemplazada por la cadena *to_str*.
REPEAT(*str*,*count*)	Retorna una cadena consistente en la cadena *str* repetida

	count veces. Si *count* <=0, retorna una cadena vacía. Retorna NULL si *str* o *count* son NULL.
REVERSE(*str*)	Retorna la cadena *str* con el orden de caracteres invertido.
INSERT(*str*,*pos*,*len*,*newstr*)	Retorna la cadena *str*, con *len* caracteres remplazados por *newstr* a partir de la posición *pos*.
ELT(N,*str1*,*str2*,*str3*,...)	Retorna *str* si N=1, *str2* si N=2, y así hasta el final. Retorna NULL si N es menor que 1 o mayor que el número de argumentos. ELT() es el complemento de FIELD().
FIELD(*str*,*str1*,*str2*,*str3*,...)	Retorna el índice de *str* en la lista *str1*, *str2*, *str3*,... Retorna 0 si *str* no se encuentra. FIELD() es el complemento de ELT().
FIND_IN_SET(*str*,*strlist*)	Retorna un valor entre 1 y N si la cadena *str* está en la lista *strlist* consistente en N subcadenas.
MAKE_SET(*bits*,*str1*,*str2*,...)	Retorna un conjunto [SET] consistente en las cadenas que se corresponden con el conjunto de bits. *str1* corresponde al bit 0, *str2* al bit 1, etc. Las cadenas NULL de *str1*, *str2*,... no se añaden al resultado.
EXPORT_SET(*bits*, *on*,*off*, [*separator*,[*number*_of_bits]])	Retorna una cadena donde cada bit indicado en '*bit*' da una cadena '*on*' y por cada bit a cero consigues una cadena '*off*'. Cada cadena se separa con '*separator*' (por defecto ',') y sólo '*number_of_bits*' (64 por defecto) de bits usados.
LCASE(*str*) / LOWER(*str*)	Retorna la cadena *str* con todos los caracteres cambiados a minúsculas de acuerdo con el actual mapeo del juego de caracteres (por defecto ISO-8859-1 Latin1).
UCASE(*str*) / UPPER(*str*)	Retorna la cadena *str* con todos los caracteres cambiados a mayúscula de acuerdo con el actual mapeo de juego de caracteres (por defecto ISO-8859-1 Latin1).
LOAD_FILE(*file_name*)	Lee un archivo y retorna su contenido como cadena.

Tabla 3.14. *Funciones de cadena en MySQL.*

3.6.3.2.Funciones de comparación de cadenas.

En cuanto a las funciones de comparación de cadenas, hay que comentar que si cualquier expresión en una comparación de cadenas es *case-sensitive*, la comparación se realiza en modo *case-sensitive*. Las funciones disponibles para este propósito se muestran en la tabla 3.15.

Función	Descripción
expr LIKE *pat* [ESCAPE '*escape-char* ']	Busca el patrón *pat* en la expresión *exp*. Retorna 1 (TRUE) si localiza el patrón o 0 (FALSE) en caso contrario.
expr NOT LIKE [EXCAPE '*escape-char* ']	Equivalente a NOT (*expr* LIKE *pat* [ESCAPE '*escape-char* ']).
expr REGEXP *pat* / *expr* RLIKE *pat*	Realiza una búsqueda de un patrón de expresión cadena *expr* contra un patrón *pat*. El patrón puede ser una expresión regular extendida. Retorna 1 si *expr* se ajusta a *pat*. En caso contrario, retorna 0.
expr NOT REGEXP *pat* / *expr* NOT RLIKE *pat*	Equivalente a NOT(*expr* REGEXP *pat*).
STRCMP(*expr1*,*expr2*)	STRCMP() retorna 0 si las cadenas son iguales, -1 si el primer argumento es menor que el segundo de acuerdo con el número de orden, y 1 en el resto de casos.
MATCH(*col1*,*col2*,...) AGAINST (*expr*) / MATCH(*col1*,*col2*,...) AGAINST (*expr* IN BOOLEAN MODE)	MATCH...AGAINST() se utiliza para búsqueda de texto completo y retorna la relevancia (medida de similitud entre el texto en las columnas (*col1*,*col2*,...) y la consulta *exp*.).
BINARY	El operador BINARY trata la cadena que le sigue

Función	Descripción
	como una cadena binaria.

Tabla 3.15. *Funciones de comparación de cadenas.*

3.6.4. Funciones de fecha y hora.

Las funciones para el tratamiento de fechas y hora que ofrece MySQL se muestran en la tabla 3.16. Los rangos de valores para cada tipo de dato ya se vieron en el apartado referente a los tipos de datos en MySQL.

Función	Descripción
DAYOFWEEK(*date*)	Retorna el índice del día de semana para la fecha dada (1 = domingo, 2 = lunes, ... 7= sábado).
WEEKDAY(*date*)	Retorna el índice del día de la semana para la fecha (0=lunes, 1=martes,..., 6 = domingo).
DAYOFMONTH(*date*)	Retorna el día del mes para la fecha, dentro del rango 1 a 31.
DAYOFYEAR(*date*)	Retorna el día del año para la fecha, en el rango de 1 a 366.
MONTH(*date*)	Retorna el mes de la fecha, en el rango de 1 a 12.
DAYNAME(*date*)	Retorna el nombre del día de la semana de la fecha.
MONTHNAME(*date*)	Retorna el nombre del mes correspondiente a la fecha dada.
QUARTER(*date*)	Retorna el trimestre de la fecha, en el rango de 1 a 4.
WEEK(*date*) / WEEK(*date*,*first*)	Con un solo argumento, retorna la semana de la fecha, en el rango de 0 a 53. La forma en dos argumentos de WEEK() permite especificar si la semana empieza en domingo o en lunes.
YEAR(*date*)	Retorna el año de la fecha, en el rango de 1000 a 9999.
YEARWEEK(*date*) / YEARWEEK(*date*,*first*)	Retorna el año y la semana de la fecha.
HOUR(*time*)	Retorna la hora para el dato dado, en el rango de 0 a 23.
MINUTE(*time*)	Retorna los minutos del dato dado, en el rango de 0 a 59.
SECOND(*time*)	Retorna los segundos para el dato dado, en el rango de 0 a 59.
PERIOD_ADD(P,N)	Añade N meses al período P (en el formato YYMM o YYYYMM).
PERIOD_DIFF(P1,P2)	Retorna el número de meses entre los períodos P1 y P2.
DATE_ADD(*date*,INTERVAL *expr type*) DATE_SUB(*date*,INTERVAL *expr type*) ADDDATE(*date*,INTERVAL *expr type*) SUBDATE(*date*,INTERVAL *expr type*)	Estas funciones realizan la aritmética de fechas.
EXTRACT(*type* FROM *date*)	Extrae partes de la fecha.
TO_DAYS(*date*)	Dada una fecha *date*, retorna el número de día (el número de día desde el año 0).
DATE_FORMAT(*date*,*format*)	Da formato a la fecha *date* de acuerdo con la cadena *format*.
TIME_FORMAT(*time*,*format*)	Se utiliza como DATE_FORMAT(), pero la cadena *format* sólo puede contener aquellos especificadores de formato que manejen horas, minutos y segundos.
CURDATE() / CURRENT_DATE	Retorna la fecha de hoy.
CURTIME() / CURRENT_TIME	Retorna la hora actual.
NOW() / SYSDATE() / CURRENT_TIMESTAMP	Retorna la fecha y hora actual.
UNIX_TIMESTAMP() / UNIX_TIMESTAMP(date)	Si se llama sin argumentos, retorna un timestamp en formato UNIX (segundos desde '1970-01-01 00:00:00 GMT) como entero sin signo. Si UNIX_TIMESTAMP() se llama con el argumento *date*, retorna el valor del

Función	Descripción
	argumento como segundos desde '1970-01-01 00:00:00' hasta la fecha indicada en *date* GMT.
FROM_UNIXTIME(*unix_timestamp*)	Retorna una representación del argumento *unix_timestamp* como un valor en formato 'YYYYMMDD HH:MM:SS' o YYYYMMDDHHMMSS, dependiendo del contexto en el que se utilice esta función.
FROM_UNIXTIME(*unix_timestamp*,*format*)	Retorna una representación del argumento unix_timestamp con un formato determinado por la cadena *format*.
SEC_TO_TIME(*seconds*)	Retorna el argumento *seconds*, convertido a horas, minutos y segundos.
TIME_TO_SEC(*time*)	Retorna el argumento *time*, convertido en segundos.

Tabla 3.16. *Funciones de fecha y hora en MySQL.*

3.6.5. Funciones geométricas.

MySQL proporciona también un conjunto de funciones para trabajar con los tipos de datos geométricos. Estas funciones se listan en la tabla 3.17.

Función	Descripción
GeomCollFromText(wkt[,srid])	Construye un valor GEOMETRYCOLLECTION usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
GeomFromText(wkt[,srid])	Construye un valor geométrico de cualquier tipo usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
LineFromText(wkt[,srid])	Construye un valor LINESTRING usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
MLineFromText(wkt[,srid])	Construye un valor MULTILINESTRING usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
MpointFromText(wkt[,srid])	Construye un valor MULTIPOINT usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
MpolyFromText(wkt[,srid])	Construye un valor MULTIPOLYGON usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
PointFromText(wkt[,srid])	Construye un valor POINT usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
PolyFromText(wkt[,srid])	Construye un valor POLYGON usando su representación WKT y un identificador de sistema de referencia espacial (opcional).
BdMPolyFromText(wkt,srid)	Construye un valor MULTIPOLYGON a partir de un valor MULTILINESTRING en formato WKT que contiene una colección cerrada de valores LINESTRING.
BdPolyFromText(wkt,srid)	Construye un valor POLYGON a partir de un valor MULTILINESTRING en formato WKT que contiene una colección cerrada de valores LINESTRING.
GeomCollFromWKB(wkb[,srid])	Construye un valor GEOMETRYCOLLECTION usando su representación WKB y un identificador de sistema de referencia espacial (opcional).
GeomFromWKB(wkb[,srid])	Construye un valor geométrico de cualquier tipo usando su representación WKB y un identificador de sistema de referencia espacial (opcional).
LineFromWKB(wkb[,srid])	Construye un valor LINESTRING usando su representación WKB y un identificador de sistema de referencia espacial (opcional).
MLineFromWKB(wkb[,srid])	Construye un valor MULTILINESTRING usando su representación WKB y un identificador de sistema de referencia

Función	Descripción
	espacial (opcional).
MPointFromWKB(wkb[,srid])	Construye un valor MULTIPOINT usando su representación WKB y un identificador de sistema de referencia espacial (opcional).
MPolyFromWKB(wkb[,srid])	Construye un valor MULTIPOLYGON usando su representación WKB y un identificador de sistema de referencia espacial (opcional).
PointFromWKB(wkb[,srid])	Construye un valor POINT usando su representación WKB y un identificador de sistema de referencia espacial (opcional).
PolyFromWKB(wkb[,srid])	Construye un valor POLYGON usando su representación WKB y un identificador de sistema de referencia espacial (opcional).
BdMPolyFromWKB(wkb,srid)	Construye un valor MULTIPOLYGON a partir de un valor MULTILINESTRING en formato WKB que contiene una colección cerrada de valores LINESTRING.
BdPolyFromWKB(wkb,srid)	Construye un valor POLYGON a partir de un valor MULTILINESTRING en formato WKB que contiene una colección cerrada de valores LINESTRING.
GeometryCollection(g1,g2,…)	Construye una representación WKB de un valor GEOMETRYCOLLECTION a partir de las representaciones WKB de varios objetos geométricos.
LineString(pt1,pt2,…)	Construye una representación WKB de un valor LINESTRING a partir de las representaciones WKB de varios objetos POINT.
MultiLineString(ls1,ls2,…)	Construye una representación WKB de un valor MULTILINESTRING a partir de las representaciones WKB de varios objetos LINESTRING.
MultiPoint(pt1,pt2,…)	Construye una representación WKB de un valor MULTIPOINT a partir de las representaciones WKB de varios objetos POINT.
MultiPolygon(poly1,poly2,…)	Construye una representación WKB de un valor MULTIPOLYGON a partir de las representaciones WKB de varios objetos POLYGON.
Point(x,y)	Construye una representación WKB de un punto usando sus coordenadas.
Polygon(ls1,ls2,…)	Construye una representación WKB de un valor POLYGON a partir de las representaciones WKB de varios objetos LINESTRING.
AsText(nombre_col)	Convierte un dato geométrico de su formato interno a una cadena WKT.
AsBinary(nombre_col)	Convierte un dato geométrico de su formato interno a tipo BLOB que contiene el valor WKB.
Dimension(g)	Devuelve la dimensión del valor geométrico g.
Envelope(g)	Devuelve el menor rectángulo que envuelva a g.
GeometryType(g)	Devuelve una cadena con el nombre del tipo geométrico del objeto g.
SRID(g)	Devuelve el indicador del sistema espacial de referencia de g.
X(p)	Devuelve la coordenada X del punto p como un número de precisión doble.
Y(p)	Devuelve la coordenada Y del punto p como un número de precisión doble.
EndPoint(ls)	Devuelve el punto final del valor LINESTRING *ls*.
Glength(ls)	Devuelve la longitud del LINESTRING o MULTILINESTRING *ls* en su sistema de referencia asociado como un número de precisión doble.
NumPoints(ls)	Devuelve el número de puntos del valor LINESTRING *ls*.
PointN(ls,n)	Devuelve el n-ésimo punto del LINESTRING *ls*.
StartPoint(ls)	Devuelve el punto de comienzo del LINESTRING *ls*.
IsClosed(mls)	Devuelve 1 si el valor MULTILINESTRING *mls* es cerrado o 0 en caso contrario.
Area(poly)	Devuelve el área del valor POLYGON o MULTIPOLYGON *poly*

Función	Descripción
	como un número de precisión doble.
ExteriorRing(poly)	Devuelve el aro exterior de un polígono como un valor LINESTRING.
InteriorRingN(poly,n)	Devuelve el n-ésimo aro interior de un polígono como un valor LINESTRING.
NumInteriorRings(poly)	Devuelve el número de aros interiores de *poly*
GeometryN(gc,n)	Devuelve el n-ésimo valor geométrico de un valor GEOMETRYCOLLECTION *gc*.
NumGeometries(gc)	Devuelve el número de objetos geométricos que contiene el valor GEOMETRYCOLLECTION *gc*.
MBRContains(g1,g2)	Devuelve 1 si el menor rectángulo que envuelva a g1 contiene al menor rectángulo que envuelve a g2. Devuelve 0 en caso contrario.
MBRDisjoint(g1,g2)	Devuelve 1 si el menor rectángulo que envuelva a g1 no interfecta con el menor rectángulo que envuelve a g2. Devuelve 0 en caso contrario.
MBREqual(g1,g2)	Devuelve 1 si el menor rectángulo que envuelva a es igual al menor rectángulo que envuelve a g2. Devuelve 0 en caso contrario.
MBRIntersects(g1,g2)	Devuelve 1 si el menor rectángulo que envuelva a g1 intersecta con el menor rectángulo que envuelve a g2. Devuelve 0 en caso contrario.
MBROverlaps(g1,g2)	Devuelve 1 si el menor rectángulo que envuelva a g1 se superpone con el menor rectángulo que envuelve a g2. Devuelve 0 en caso contrario.
MBRTouches(g1,g2)	Devuelve 1 si el menor rectángulo que envuelva a g1 toca al menor rectángulo que envuelve a g2. Devuelve 0 en caso contrario.
MBRWithin(g1,g2)	Devuelve 1 si el menor rectángulo que envuelva a g1 esta dentro del menor rectángulo que envuelve a g2. Devuelve 0 en caso contrario.

Tabla 3.17. *Funciones geométricas en MySQL.*

3.6.6. Funciones definidas por el usuario.

Existen dos métodos para añadir nuevas funciones a MySQL. El primero es a través de la interfaz de funciones de usuario (UDF). Estas funciones de usuario se compilan como ficheros objeto y posteriormente añadidas y eliminadas dinámicamente del servidor empleando las sentencias CREATE FUNCTION y DROP FUNCTION. El otro método, más complejo, no supone en realidad una característica de MySQL, pues consiste en modificar el código fuente de este para añadir funciones nativas.

3.7. BLOQUEO DE TABLAS Y TIPOS DE CLAVES.

En este apartado se van a estudiar dos aspectos fundamentales en el diseño de una base de datos, como son el bloqueo de tablas y los tipos de claves. Se verá como trabaja MySQL con ambos aspectos. Los bloqueos son importantes para proteger la integridad de los datos en un entorno multiproceso, mientras que las claves son importantes para el diseño de la arquitectura de una base de datos.

3.7.1. Bloqueos.

Los bloqueos constituyen una parte importante en el diseño de una base de datos consistente. Imagine una situación en la que una base de datos esta muy ocupada, de tal manera que los accesos se hacen casi simultáneamente. Mientras algunos subprocesos

tratan de leer datos, otos necesitan leerlos, calcularlos y después escribirlos. En estas situaciones, se necesita un cierto control sobre el acceso a las tablas de la base de datos, pues se pueden dar casos de lecturas erróneas o indeseadas. Bloquear permite a los subprocesos tener acceso exclusivo a un cierto número de tablas, asegurando que pueden hacer las tareas requeridas sin la interferencia de otros subprocesos.
MySQL proporciona para este fin la función LOCK TABLES. La sintaxis de esta instrucción es la siguiente:

LOCK TABLES *nombre_tabla1* (AS *alias*) {READ | [LOW_PRIORITY] WRITE} [*nombre_tabla2* {READ | [LOW_PRIORITY] WRITE}...];

Como se puede observar, este comando permite el bloqueo de varias tablas simultáneamente, algunas para bloquearse en lectura (READ) y otras para escritura (WRITE).
En MySQL existen unas reglas simples para el manejo de bloqueos de lectura o escritura:

- Si un subproceso hace un bloqueo de lectura en algunas tablas, ese subproceso, y todos los demás, sólo podrán leer de esas tablas.
- Si un subproceso hace un bloqueo de escritura, se convierte en el único subproceso con cualquier tipo de acceso a esas tablas. Puede, por tanto, leer y escribir en ellas, y ningún otro subproceso puede acceder a las tablas hasta que el subproceso que las bloqueó las libere.

El hecho de enviar el comando LOCK TABLES al servidor no implica que instantáneamente se obtenga el bloqueo, sino que hay un proceso bastante importante de cola de espera primero. En esta cola de espera, un subproceso que solicita un bloqueo tiene que esperar hasta que otros bloqueos de las tablas sean liberados. Hay una cola para los bloqueos de escritura y otra para los de lectura, las cuales funcionan de manera sensiblemente diferente.

El orden de prioridades para un bloqueo WRITE es el siguiente:

- Si no hay bloqueos de cualquier tipo en ese momento en la tabla, se otorga el bloqueo WRITE sin colocarlo en la cola de espera.
- Si hay bloqueos de algún tipo, el nuevo bloqueo se coloca en una cola de bloqueos WRITE.

El orden de prioridades para un bloqueo de tipo READ es el siguiente:

- Si la tabla no tiene bloqueos WRITE, se otorga el bloqueo READ sin colocarlo en la cola de espera.
- Si la tabla tiene bloqueos WRITE la solicitud se coloca en la cola de bloqueos READ.

Cuando se libera un bloqueo, los subprocesos de la cola de procesos WRITE tienen prioridad sobre los de la cola READ. De esta forma, al dar prioridad a los subprocesos para desempeñar una operación WRITE, MySQL asegura que cualquier actualización a la base de datos se procese tan rápido como sea posible. No obstante, hay una forma de contrarrestar este comportamiento en el caso de que se necesite otorgar bloqueos READ

más urgentemente que los bloqueos WRITE. Para ello se puede usar el modificador LOW_PRIORITY en la sentencia LOCK TABLES. Esto hace que el sistema de colas comentado anteriormente se comporte de manera diferente (para obtener un bloqueo LOW_PRIORITY WRITE se tendrá que esperar a que todos los procesos READ hayan liberado sus colas). Otra forma de influir en la política de listas de esperas es utilizar un SELECT HIGH_PRIORITY. Si se envía esta instrucción, se permite a SELECT leer de la tabla, incluso si hay un bloqueo WRITE en la cola.

Para desbloquear las tablas que hayan sido bloqueadas por un subproceso se debe emplear el comando UNLOCK TABLES. Las tablas también serán desbloqueadas si el mismo subproceso envía otro comando LOCK, o si está cerrada la conexión al servidor. Los bloqueos no serán liberados por ningún compás de espera.

Una de las desventajas más importantes de MySQL es que no todos los motores de almacenamiento disponibles (modos en que se almacenan los datos) poseen control transaccional. Para la mayoría de los motores de almacenamiento disponibles, MySQL carece de la habilidad para administrar varias transacciones dentro del sistema de administración de bases de datos. Con el control transaccional, los cambios a las bases de datos no afectan de inmediato a las tablas de destino, aunque parezca que si lo hacen. En vez de eso, los datos modificados se almacenan en un buffer temporal hasta que se envíe un comando COMMIT para conformar los nuevos datos para la tabla. Para paliar esta carencia, se pueden utilizar los bloqueos en MySQL mediante una técnica muy simple. Esta técnica consiste en que una vez que se hayan bloqueado las tablas requeridas, se hagan pruebas para cualquier tipo de condiciones adversas que hicieran pensar en deshacer las modificaciones en las tablas. Después, si todo esta bien, se harán las modificaciones y se desbloquearan las tablas.

3.7.2. Claves.

Una clave en una tabla de base de datos proporciona los medios para localizar rápidamente información específica. Es un campo en la tabla que ayuda a localizar, de manera muy eficiente, las entradas de una tabla.

Una clave existe como una tabla extra en una base de datos, aunque pertenece a la tabla base. Ocupa espacio físico, y puede ser tan grande como la tabla principal y, en teoría, hasta más grande. Cada clave se relaciona a una o más columnas en una tabla específica. El uso adecuado de claves puede mejorar notablemente el funcionamiento de una base de datos. En las bases de datos modernas, entre las que se incluye MySQL, las claves están diseñadas para minimizar el acceso a disco cuando se está leyendo de dichas bases de datos. Incluso el método de rastrear por medio de la clave y determinar si los datos coinciden con lo que se está buscando involucra algoritmos de coincidencia muy complicados.

MySQL soporta los siguientes comandos para crear claves en las tablas existentes:

ALTER TABLE *nombre_tabla* ADD {KEY | INDEX} *nombre_indice* (*nombre_columna1*[,*nombre_columna2*,…]);
ALTER TABLE *nombre_tabla* ADD UNIQUE *nombre_indice* (*nombre_columna1*[,*nombre_columna2*,…]);

ALTER TABLE *nombre_tabla* ADD PRIMARY KEY *nombre_indice* (*nombre_columna1*[,*nombre_columna2*,…]);

Estos comandos ya se vieron anteriormente cuando se hablo de los índices. En MySQL, clave e índice son sinónimos. Por tanto, no se hará mucho más hincapié en este apartado. Lo único que cabe reseñar es que MySQL proporciona diversos tipos de clave, que son:

- Claves de una columna.
- Claves de columnas múltiples (índices compuestos). Sirve para crear claves en más de una columna de la misma tabla.
- Claves parciales. Sirven para indexar sólo los primeros caracteres de una columna de tipo CHAR o VARCHAR. Para ello, detrás del nombre de la columna, entre paréntesis, se pondrá el número de caracteres que se desea indexar.
- Claves parciales compuestas. Permiten crear claves parciales en columnas múltiples.
- Claves únicas. Sirven para asegurar la integridad de los datos. Con este tipo de clave no se permitirá la creación de una segunda fila con un dato duplicado en el campo que marque la clave.
- Claves externas. Actualmente MySQL no soporta las claves externas (excepto para el motor de almacenamiento InnoDB), aunque se incluye algo de sintaxis para completar y facilitar la portabilidad del código a otros sistemas de bases de datos.
- Claves primarias. Son similares a una clave única en el sentido de que sus datos deben ser únicos, pero la clave primaria de una tabla tiene un estado más privilegiado. Sólo puede existir una clave primaria por cada tabla, y los valores de sus campos no pueden ser nulos. Este tipo de claves se usa generalmente para definir relaciones entre tablas.
- Claves primarias de columnas múltiples. Igual que la anterior, pero la clave incluye varias columnas de la misma tabla.

Como se puede observar, MySQL proporciona una gran variedad de tipos de claves para usar en bases de datos. El uso de claves no es algo trivial, pero su buen uso puede hacer que el desempeño de una base de datos sea bastante bueno, y MySQL proporciona las herramientas adecuadas para este fin.

3.8. HERRAMIENTAS DE ADMINISTRACIÓN

MySQL proporciona un gran número de herramientas para administrar de forma eficiente una base de datos. En este apartado se estudiarán algunas de estas herramientas, que van desde útiles para el respaldo de datos hasta herramientas para reparación de bases de datos dañadas. Se comprobará que en este aspecto, MySQL tiene un gran potencial. Si algo se le podría objetar a estas herramientas era que su uso tradicionalmente estaba ligado a la introducción de datos por la línea de comandos. Pero desde hace no mucho tiempo MySQL proporciona un cliente gráfico para tareas de administración de una base de datos denominado MySQL Administrador.

3.8.1. Salvaguardar datos

El respaldo de los datos constituye una tarea fundamental en la administración de cualquier sistema de bases de datos. Además, tan importante como el respaldo de los archivos es la manera de restaurar los datos en caso de pérdida total o parcial de los mismos, en el sentido de que sea rápida, segura y que devuelva el funcionamiento normal con un mínimo de retrasos.
Para este fin, MySQL provee dos herramientas, que son *mysqldump* para hacer la copia de seguridad y *mysqlimport* para la restauración. Estas dos herramientas se verán más en detalle más adelante.

3.8.2. Registro de cambios.

MySQL ofrece algunas formas útiles para registrar cambios en una base de datos. Se puede indicar a MySQL que mantenga un registro de cada cambio a la base de datos. Para ello, sólo se tendrá que iniciar el servidor MySQL con la opción *--log-update*. Esto crea un registro de actualización en forma de consultas SQL que pueden volverse a ejecutar para reproducir las actualizaciones. Los registros tienen nombres como *nombre_host*.n, donde n es un número que se incrementa cada vez que se inicia un nuevo registro. Un nuevo registro se iniciará cada vez que se ejecute alguna de estas instrucciones:

- “mysqladmin refresh”.
- “mysqladmin flush-logs”.
- “mysqldump –flush-logs” con opción.
- “FLUSH LOGS”.
- Un inicio o reinicio del servidor.

Esta herramienta puede resultar útil para hacer respaldos por incrementos, es decir, en vez de hacer un respaldo completo del sistema, hacer sólo un respaldo de las últimas modificaciones que se hicieron desde que se hizo el último respaldo completo del sistema. De esta forma, si se ejecuta el servidor con esta opción, se podrá hacer un respaldo completo del sistema (lo cual elimina automáticamente los registros con la correspondiente opción) cada cierto tiempo, e ir guardando los registros desde que se hizo el último respaldo completo.

Aparte de los explicados antes, existen otros registros de transacciones en MySQL. Si se inicia el servidor MySQL con la opción *–log*, MySQL generará un registro de su actividad principal. Este registro lo registra prácticamente todo. Este registro se suele almacenar en /usr/local/mysql/data o en /usr/local/var (dependiendo del sistema) con el nombre de *nombre_host.log*, donde *nombre_host* es el nombre de la máquina. Esta opción puede resultar muy útil para diagnosticar errores de una aplicación basada en MySQL.

Si se desea obtener información detallada sobre lo que está haciendo MySQL, también se puede iniciar el servidor MySQL con la opción *–debug*. Esto permitirá depurar problemas que no son fáciles de diagnosticar. Es un modo de depuración que contiene muchas opciones adicionales, y que creará un registro que contendrá todo lo que MySQL está haciendo. Este registro se almacenará en un fichero.

3.8.3. *Scripts* asociados al servidor.

El servidor MySQL, *mysqld*, es el programa principal que hace la mayoría del trabajo en un sistema gestor de base de datos MySQL. Este servidor está acompañado por una serie de *scripts* que facilitan opciones de configuración del servidor. La siguiente lista describe estros *scripts* de manera breve:

- **mysqld.** Es el servidor de MySQL. Ya se vio en un capítulo anterior. Para usar programas clientes, el servidor debe estar ejecutándose.
- **mysqldmax.** Es una versión del servidor que incluye algunas características adicionales, tales como algunos motores de almacenamiento extras.
- **mysqld_safe.** Este *script* intenta iniciar *mysqld-max* en el caso de que este exista. Si no existe, inicia *mysqld*.
- **mysql.server.** Se utiliza en sistemas que utilizan directorios que contienen *scripts* que inician el sistema para niveles de ejecución particulares.
- **mysqld_multi.** Este *script* puede iniciar o detener múltiples servidores instalados en el sistema.
- **mysql_install_db.** Este *script* ya se vio anteriormente. Sólo es necesario ejecutarlo una vez, después de la instalación de MySQL. Creará las tablas de *Grant* con los privilegios por defecto.
- **mysql_fix_privilege_tables.** Este *script* se utiliza únicamente después de una reinstalación de MySQL, para actualizar las tablas de *Grant* con cualquier cambio que se hayan hecho en las versiones más actuales de MySQL.

3.8.4. Funciones administrativas de MySQL.

MySQL tiene algunas funciones que proporcionan acceso a los aspectos administrativos de la base de datos, las cuales pueden llamarse desde dentro de la base de datos MySQL, es decir, desde el monitor o desde cualquier aplicación cliente. Algunos de estos comandos se muestran en la tabla 3.18.

Función	Descripción
DATABASE()	Devuelve el nombre de la base de datos actual a la que el usuario está conectado.
USER(), SYSTEM_USER() y SESSION_USER()	Estas funciones devuelven el nombre del usuario actual de la base de datos, incluyendo el nombre del host. El valor devuelto será de la forma *usuario@nombre_host.*
VERSION()	Devuelve el número de versión que se está usando.
PASSWORD(*cadena*)	Esta función codificará una cadena dada como parámetro y devolverá el resultado codificado, tal como se almacena en la columna *password* de la tabla *Users* de la base de datos *mysql*.
ENCRYPT(*cadena*[,*salt*])	Esta función codificará una cadena dada con una semilla opcional (*salt*).
ENCODE(*cadena*,*contraseña*) / DECODE(*cadena_codificada*,*contraseña*)	La función ENCODE() codifica una cadena dada con una contraseña como contraseña de acceso y devuelve una cadena binaria. DECODE() realiza la función contraria, es decir, toma la cadena codificada binaria y la decodifica con la contraseña.
MD5(*cadena*)	Devuelve una suma de verificación MD5 para una cadena dada, es decir, un número hexadecimal de 32 caracteres.
LAST_INSERT_ID([*expr*])	Esta función devuelve el último valor introducido en una

Función	Descripción
	columna AUTO_INCREMENT después de que ha sido generada automáticamente. Esta se refiere solamente a los valores manejados por la conexión actual a la base de datos.

Tabla 3.18. *Funciones administrativas de MySQL.*

3.8.5. mysqladmin

La herramienta *mysqladmin* se utiliza para realizar una gran variedad de opciones administrativas en una base de datos de MySQL. Normalmente se ejecuta desde la línea de comandos de Linux. Al invocar esta utilidad, se deben especificar una serie de opciones para indicarle al programa cómo ejecutarse y qué hacer. Anteriormente se vieron algunos usos de esta utilidad, como por ejemplo para detener el servidor MySQL. En este apartado se estudiarán más de lleno las características de este programa. La sintaxis básica de *mysqladmin* es la siguiente:

mysqladmin [*opciones*] *comando1* [*opc_com1*] *comando2* [*opc_com2*]

Después de asignar nombres de usuario y privilegios a una base de datos, serán necesarias las opciones –p y/o –u al invocar a *mysqladmin* si el nombre de usuario de MySQL es distinto del nombre de usuario de Linux.
Los comandos que se pueden utilizar con *mysqladmin* se muestran en la tabla 3.19.

Comando	Descripción
create *nombre_bd*	Crea una nueva base de datos con el nombre especificado.
drop *nombre_bd*	Elimina la base de datos con el nombre especificado.
status	Proporciona un pequeño mensaje de estado del servidor.
version	Despliega la versión del servidor.
extended-status	Crea un mensaje de estado extendido del servidor.
variables	Despliega las variables disponibles.
proccesslist	Muestra una lista de todos los subprocesos activos en el servidor. Este comando es muy útil para ver lo ocupado que está el servidor.
flush-host	Vacía todos los host almacenados en la caché.
flush-logs	Vacía todos los registros.
flush-tables	Vacía todas las tablas.
flush-privileges	Carga de nuevo las tablas de permiso de acceso.
kill *id1,id2*	Elimina subprocesos de MySQL.
password *nueva_contraseña*	Cambia la contraseña anterior por una nueva.
ping	Envía una señal al servidor para verificar si está funcionando.
reload	Obliga a MySQL a cargar nuevamente las tablas de permiso de acceso.
refresh	Vacía todas las tablas de concesiones y cierra y abre todos los archivos de registro.
shutdown	Apaga el servidor MySQL

Tabla 3.19. *Comandos disponibles en mysqladmin.*

Además de los comandos anteriores, a *mysqladmin* se le pueden especificar las opciones mostradas en la tabla 3.20.

Opción	Efecto
-#, --debug	Crea un registro de depuración.
-f, --force	Fuerza a continuar el proceso de eliminación de una base de datos, a pesar de la advertencia.
-C, --compress	Utiliza compresión en el protocolo cliente/servidor.
-h, --host=*nombre_host*	Se conecta al host especificado.
-p, --password	Especifica la contraseña que debe utilizarse al conectarse al servidor. Si no se proporciona la contraseña, se solicitará.
-P, --port[*numero_de_puerto*]	Especifica el número de puerto que se va a utilizar en la conexión.
-i, --sleep=*num*	Ocasionará que *mysqladmin* ejecute los comandos una y otra vez, con un tiempo muerto entre ellos de *num* segundos.
-s, --silent	Ocasionará que *mysqladmin* salga silenciosamente si no puede conectarse al servidor.
-S, --socket=*socket*	Especifica el archivo *socket* que se va a utilizar para la conexión.
-t, --timeout=*num*	Especifica el tiempo de final de cesión en segundos, para la conexión al servidor.
-u, --user=*nombre_usuario*	Especifica en nombre de usuario para ingresar si no es el usuario del sistema actual.
-V, --version	Imprime información sobre la versión y sale.
-w, --wait [=*intentos*]	Espera e intenta nuevamente el número especificado de veces si la conexión a la base de datos no está disponible.
-?, --help	Despliega los comandos y opciones y se sale.

Tabla 3.20. *Opciones de mysqladmin.*

Algunos de estos comandos son bastante intuitivos, como por ejemplo los que sirven para crear y eliminar bases de datos. Como puede verse, la lista de comandos es bastante amplia.

3.8.6. Reparaciones en bases de datos.

Un problema muy grave que se le puede presentar a cualquier administrador de bases de datos es que se dañen algunos elementos de dicha base de datos. Un buen sistema gestor de bases de datos debe proporcionar las herramientas adecuadas para solucionar este posible problema y reparar las tablas dañadas. MySQL proporciona dos herramientas para realizar esto: *isamchk* y *myisamchk*. En este apartado se verá cómo funcionan estas dos herramientas, pero antes de ello, es conveniente detenerse un poco en conocer la estructura de datos en MySQL y de qué forma almacena la información.

3.8.6.1.Estructura de datos en MySQL.

Si se explora la estructura de directorios de las bases de datos creadas en MySQL (que habitualmente se encuentran en /var/lib/mysql), se puede ver que para cada base de datos MySQL crea un directorio con su nombre. Si se observa el contenido de esos directorios se puede ver que por cada tabla existen tres archivos con el mismo nombre (el nombre de la tabla) y diferentes extensiones (.frm, .MYD y .MYI).
El archivo con la extensión .frm es el archivo de formato que contiene la estructura de datos. Este archivo no está en un formato legible. El archivo .MYD contiene los datos reales de la tabla, y el archivo .MYI contiene la información sobre las claves y otras referencias cruzadas internas. Es este último el archivo al cual acuden las herramientas *isamchk* y *myisamchk* para realizar cualquier reparación. Estas dos herramientas son equivalentes. La diferencia entre ellas es que *isamckh* trabaja con tablas que se han almacenado según el motor de almacenamiento ISAM (extensiones de tabla .ISM

e .ISD), y *myisamchk* trabaja con tablas que se almacenan según el motor MyISAM (extensiones .MYI y MYD).

3.8.6.2.myisamchk / isamchk

La herramienta *myisamchk* proporciona información acerca de las tablas de una base de datos, y permite chequearlas y repararlas en caso de fallo. Como se comentó antes, *myisamchk* e *isamchk* tienen la misma función, pero se utilizan para métodos de almacenamiento diferentes. De aquí en adelante, se hablará únicamente de *myisamchk*, puesto que MyISAM es el motor de almacenamiento por defecto en MySQL, pero todo lo que se diga aquí es perfectamente aplicable a *isamchk*.

La sintaxis de *myisamchk* es la siguiente:

myisamchk [opciones] *nombre_tabla* …

Esta herramienta posee un gran número de opciones. Estas opciones son las que indican a la utilidad lo que debe hacer. Si no se especifican opciones, *myisamchk* simplemente chequea la tabla que se le indique. Esta es la operación por defecto. El nombre de la tabla sobre la cual se quiere actuar se especifica en el campo *nombre_tabla*. Si no se está en el directorio de dicha base de datos, para especificar el nombre de la tabla se debe especificar la ruta completa. Se puede emplear esta herramienta con varias tablas a la vez, sin más que indicar sus nombres separados con un espacio. También se puede actuar sobre todas las tablas de un directorio usando el patrón *.MYI.
Las opciones que se pueden utilizar con *myisamchk* se muestran en la tabla 3.21.

Opción	Efecto
-a, --analyze	Analiza la distribución de claves en la tabla. Esta opción es útil para acelerar algunas operaciones conjuntas.
-#=*opciones_depuración*, --debug=*opciones_depuración*	Crea un registro para depuración, utilizando frecuentemente la forma 'd:t:o,*nombre_archivo*' para guardar en el archivo especificado en *nombre_archivo*.
-d, --description	Se obtiene información descriptiva sobre el estado de una tabla.
-e, --extended-check	Verifica exhaustivamente un archivo. Normalmente no se emplea mucho, porque *myisamchk* encuentra la mayoría de los errores sin esta opción.
-f, --force	Sobrescribe archivos temporales.
-i, --information	Muestra información estadística de la tabla.
-k=*num*, --keys-used=*num*	Utilizado con la opción –r, indica a *myisamchk* que omita las primeras *num* claves antes de reparar.
-l, --no-symlinks	Indica a *myisamchk* que no siga vínculos de enlaces simbólicos (lo hará en forma predeterminada).
-q, --quick	Se utiliza con –r para una reparación más rápida. Sólo repara los archivos .MYD; no repara archivos de datos a menos que se especifique una segunda –q opcional, en cuyo caso también se repararán dichos archivos.
-r, --recover	Realiza una recuperación. Esto arreglará la mayoría de los problemas con excepción de violaciones de clave única.
-o, --safe_recover	Se utiliza un método más antiguo de recuperación. La recuperación con este método es más lenta que con –r.
-s, --silent	Imprime solamente los errores. Si se especifica –ss se hará en un modo "muy silencioso".
-v, --verbose	Imprime más información. Si se emplea esta opción dos veces (-vv) se imprimirá información aún mas detallada.

Opción	Efecto
-S, --sort-index	Ordena los bloques de índice para alguna mejoría de velocidad en las aplicaciones que realizan muchas operaciones del tipo “leer siguiente”.
-R=índice, --sort-records=índice	Ordena los registros de datos según un índice dado. Esto puede mejorar la velocidad en algunas consultas.
-u, --unpack	Descomprime un archivo comprimido con *pack_myisam*.
-V, --version	Despliega el número de versión y sale.
-W, --wait	Espera antes de procesar si la tabla está bloqueada.
-?, --help	Muestra una lista completa de las opciones.

Tabla 3.21. *Opciones de myisamchk*

Antes de terminar con este apartado, hay que hacer una serie de consideraciones. En primer lugar, cabe comentar que si algún cliente accede a una base de datos a través del servidor en el momento en que se está ejecutando *myisamchk*, éste puede confundirse y pensar que las tablas están dañadas cuando en realidad no es así. Para *myisamchk*, cualquier actualización en proceso puede parecer como errores o daños. Por lo tanto, para evitar esta situación, es conveniente cerrar el servidor antes de ejecutar *myisamchk*. Alternativamente, siempre que no se pueda cerrar el servidor durante un tiempo, se puede ejecutar *mysqladmin flush-tables* antes de ejecutar *myisamchk*, siempre y cuando se esté seguro de que nadie está accediendo a las tablas. En segundo lugar, para un administrador de bases de datos, es buena práctica ejecutar *myisamchk –d* de manera regular en una base de datos que está creciendo, para estar al tanto del espacio libre que se esta dejando en las tablas. Esto es de particular importancia si la aplicación ejecuta muchas instrucciones DELETE, lo que probablemente resultará en espacios entre los datos del disco duro. Este espacio será espacio desperdiciado. Siempre que la cuenta de los bloques eliminados parezca ser muy alta (información que se muestra con la opción –d), o una parte de los registros de datos, se debe ejecutar *myisamchk –r* para eliminar el espacio desperdiciado y poder así reutilizarlo.

3.8.6.3.Reparar problemas con claves.

Aunque las claves (o índices) están pensadas para mejorar el rendimiento de una base de datos, hay ocasiones en que lo empeoran. Esta situación suele darse cuando se utilizan mucho las instrucciones INSERT y DELETE. Si se ejecuta alguna de las herramientas anteriores cuando existen claves dañadas, *myisamchk* supondrá que son los datos los que están dañados, y no las claves, eliminando por tanto alguno de ellos. De esta forma, cuando existan problemas con claves, el procedimiento para reparar la base de datos debe ser el siguiente:

- Eliminar temporalmente la clave.
- Reparar la tabla.
- Restablecer las claves.

Para ello, se utilizará *myisamchk* de la siguiente forma:

```
myisamchk –rqk=0 *.MYI
```

Esta orden establecerá a cero las claves utilizadas (eliminándolas, por tanto) y después realiza una verificación y reparación rápidas. Para reconstruir las claves, se debe ejecutar de nuevo *myisamchk –rq*.

3.8.7. MySQL Administrator.

Todas las herramientas de administración que se han mostrado hasta ahora tienen una característica común, que es el hecho de que se deben ejecutar desde la línea de comandos, con todos los inconvenientes que esto conlleva. Hasta hace algún tiempo se echaba en falta de alguna herramienta que permitiese hacer todas estas tareas desde una misma interfaz gráfica e intuitiva, que permitiese a los administradores realizar todas las tareas anteriores de una forma sencilla y clara. Afortunadamente, MySQL cuenta con una herramienta de este estilo: MySQL Administrator.

MySQL Administrator es un programa para realizar tareas administrativas, tales como configuración del servidor, monitorizar su estado, iniciarlo o pararlo, administrar los usuarios y conexiones, hacer copias de seguridad, etcétera. Como se podrá ver, se podrán realizar una gran variedad de tareas desde una misma interfaz gráfica, con la comodidad que esto supone para cualquier administrador. Las características fundamentales de este programa son las siguientes:

- Una interfaz gráfica intuitiva.
- Proporciona una mejor información de la configuración y del estado del servidor.
- Proporciona indicadores gráficos de rendimiento, que permiten optimizar y determinar de manera más fácil el estado del servidor.

MySQL Administrator está diseñado para trabajar con versiones de servidores MySQL iguales o superiores a la 4.0, y esta disponible para Linux, Windows y Mac OSX. MySQL Administrator está disponible bajo dos licencias, una GPL y otra comercial. Este programa no suele venir con las distribuciones de Linux, por lo que hay que descargarlo desde la Web del fabricante.

3.8.7.1. Instalación y puesta en marcha.

Para que MySQL Administrator funcione bajo una máquina Linux, esta debe tener un gestor de ventanas instalado. Este programa está especialmente diseñado para funcionar bajo el gestor Gnome, pero no debería tener problemas para funcionar con algún otro. MySQL AB ha probado MySQL Administrator para las versiones 2.4 y 2.6 del *kernel* de Linux, pero aseguran que funciona bien con otras versiones.

La instalación bajo Linux es sumamente sencilla. Una vez que se haya descargado el archivo .tar.gz sólo habrá que descomprimirlo en el directorio que se desee, lo que creará una carpeta con el nombre mysql-administrator, que contendrá el ejecutable para iniciar dicha aplicación.

Para empezar con MySQL Administrator, simplemente hay que ejecutar el archivo ejecutable mysql-administrator. Tras hacer esto se mostrará la siguiente ventana:

Figura 3.2. *Pantalla de conexión de MySQL Administrator.*

Esta es la ventana de conexión al servidor. Si el servidor ya está ejecutándose, solo se tendrá que introducir el nombre de host, el usuario y el *password*. El puerto que utiliza MySQL por defecto es el 3306. Una vez introducido esto, se pulsa en "*Connect*" para conectar al servidor.
Si el servidor MySQL no esta ejecutándose, en la ventana de conexión de MySQL Administrator se deberá pulsar la tecla CTRL. Al pulsar esta tecla el botón "*Connect*" cambia por "*Skip*". Al pulsar sobre este último, se entrará en la aplicación. Desde ésta se podrá efectuar la conexión al servidor.

3.8.7.2.Ventana principal.

La ventana principal del programa se muestra en la figura 3.3.

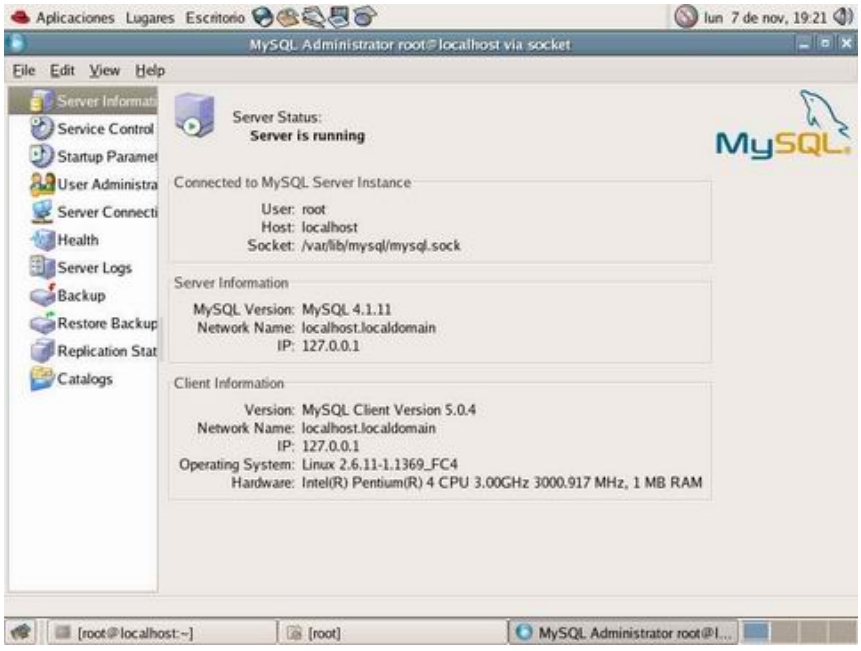

Figura 3.3. *Pantalla principal de MySQL Administrator.*

Como puede verse, la ventana principal contiene una barra lateral dividida en varias secciones, un área de trabajo, en la que se muestra la información pertinente y se hacen las modificaciones oportunas, y una barra de menús, al estilo de cualquier programa de Windows. El funcionamiento del programa es bastante simple. Para hacer alguna modificación en algún parámetro, sólo se tendrá que seleccionar la categoría oportuna en la barra lateral, y en el área de trabajo se harán las modificaciones necesarias. Para que estas modificaciones queden grabadas, se deberá pulsar en el botón "*Save Changes*" que aparece en la parte inferior de la pantalla. Al hacer clic en este botón, se generará la secuencia de instrucciones SQL necesarias para hacer las modificaciones indicadas. En una ventana emergente saldrá dicha secuencia de instrucciones y se pedirá una confirmación para su ejecución. Si no se quieren aplicar los cambios, se podrá hacer clic en el botón "Revert Changes" que aparece también en la parte inferior de la ventana.

Las tareas que se pueden realizar con esta herramienta son las siguientes:

- **Información del servidor**. En la figura 3.3 se muestra la primera sección que muestra información correspondiente al servidor MySQL, tal como versión del servidor, nombre de la red y la dirección IP. También se muestra información acerca del cliente, como la versión, el nombre de la red, la dirección IP, información de hardware y sistema operativo; e información sobre la conexión (usuario, *host* y nombre del *socket*). Esta pantalla es meramente informativa.
- **Control del servidor**. En esta sección se puede ver el estado del servidor (en ejecución o no). Se puede también arrancar y detener el servidor MySQL con una simple pulsación de ratón. En las versiones actuales, sólo se permite el inicio y la detención del servidor si éste está en la misma máquina que MySQL Administrator. Pero según el propio fabricante, futuras versiones contendrán la posibilidad de iniciar y detener remotamente el servidor. En esta ventana también se muestra una pequeña ventana denominada "*Startup Log Messages*" en la que se pueden ver todos los mensajes que se generan durante el inicio del servidor.
- **Parámetros de inicio**. Esta sección permite establecer variables que el servidor MySQL lee al iniciarse. Estas variables se almacenan en un fichero determinado del servidor, y este, al iniciarse, lee dicho fichero para obtener los valores de las variables. En esta ventana se pueden configurar una gran cantidad de parámetros (parámetros generales, parámetros MyISAM, parámetros InnoDB, parámetros de rendimiento, ficheros de registro, parámetros de replicación, parámetros de seguridad, parámetros de red, etc.).
- **Administración de usuarios**. En la sección "*User administrator*" se pueden configurar todos los aspectos referente a los usuarios.
- **Conexiones al servidor**. En este apartado se muestra información detallada sobre todas las conexiones existentes al servidor (también llamadas hilos). En la ventana de trabajo se muestra un listado de todos los hilos activos (*Threads*) con información sobre Id del proceso, usuario, *host*, base de datos que está usando, tarea que está ejecutando, etc. También se muestra un listado de todos los hilos activos ordenados por usuarios (*User Conections*). En la parte inferior de la ventana aparecen una serie de botones que permiten actualizar el listado mostrado, terminar un usuario o terminar un proceso.
- **Rendimiento (*Health*)**. La sección *Health* permite monitorizar gráficamente, y en tiempo real, el estado del servidor MySQL. En la ventana principal hay una

serie de gráficas de estado predefinidas, que proporcionan información básica del estado del servidor, carga del mismo y uso de memoria. Aparte de estas gráficas predefinidas, se pueden crear otras según otros criterios.

- **Registros del servidor**. En este apartado se muestra el campo "*Server Logs*", a través del cual se puede tener fácil acceso a varios archivos de registro (*log files*) generados por el servidor. A través de esta utilidad se podrá acceder a los registros de error del servidor y al registro general. Para cada registro existe una pestaña, que se divide en dos secciones, que permite navegar a través de archivos de registro grandes dividiéndolo en páginas individuales.
- **Salvaguarda de datos**. En la sección "*Backup*" se pueden realizar copias de seguridad de los datos de cualquier base de datos de manera sencilla en intuitiva. Esta utilidad tiene una característica bastante interesante, que es la capacidad de planificar las copias de seguridad, es decir, que se le puede indicar al programa que planifique las copias de seguridad cada cierto tiempo. Esto se configura en la pestaña "*Schedule Backup*".
- **Recuperación de datos**. En la opción "*Restore Backup*", se permitirá recuperar los datos previamente salvaguardados en caso de desastre. Esto es una operación sumamente sencilla, pues sólo se tendrá que indicar el camino donde esta el archivo de la correspondiente copia de seguridad y pulsar en "*Restore Backup*". Al seleccionar el archivo, se muestra en el área de trabajo información relativa al mismo (Numero de tablas, tamaño de las tablas, etc.) y algunas opciones de configuración de la restauración, como la posibilidad de omitir los posibles errores SQL y la posibilidad de indicar si se desea la creación de las bases de datos en caso de que no existan.
- **Estado de replicación**. En el apartado "*Replication Status*" se muestra una lista de los maestros y esclavos replicados. En esta pantalla se puede seleccionar qué máquinas esclavas están conectadas al maestro y cuales no lo están. Para usar esta opción se debe estar conectado al servidor que actúa como maestro en el esquema de replicación. Para usar esta funcionalidad es necesario disponer de una versión del servidor mayor o igual a la 4.0. Para que un servidor esclavo sea mostrado en esta ventana, se debe "autoregistrar" en el servidor maestro. Para permitir que el servidor esclavo haga esto, hay que establecer la opción "*report-host*" en la configuración del servidor maestro. Una vez que está todo correctamente configurado, todos los servidores se listan en el área de trabajo (tanto el maestro como los esclavos), con su Id de servidor, número de puerto y estado actual. Para actualizar el estado de los servidores, se deberá pulsar en el botón "Actualizar" de la parte inferior de la pantalla. Junto a este botón aparecen otros dos, que permiten agregar /quitar servidores para monitorizar.
- **Administración de base de datos y tablas**. Bajo la sección "*Catalogs*" se muestra una vista gráfica de las bases de datos existentes, las tablas y sus correspondientes propiedades. Desde esta pantalla se le permite al administrador de la base de datos realizar una gran cantidad de operaciones sobre las tablas de una base de datos, e incluso crear o eliminar bases de datos y tablas. Una opción bastante potente que ofrece MySQL Administrator es el "administrador de tablas". Si en la pestaña correspondiente a las tablas se hace doble clic en una de las tablas (o se selecciona la opción editar en el menú contextual) o si se pulsa en "*Create Table*" aparecerá el editor de tablas, que permitirá crear o modificar tablas.

A continuación se muestran algunas capturas de pantalla que muestran la interfaz de MySQL Administrator.

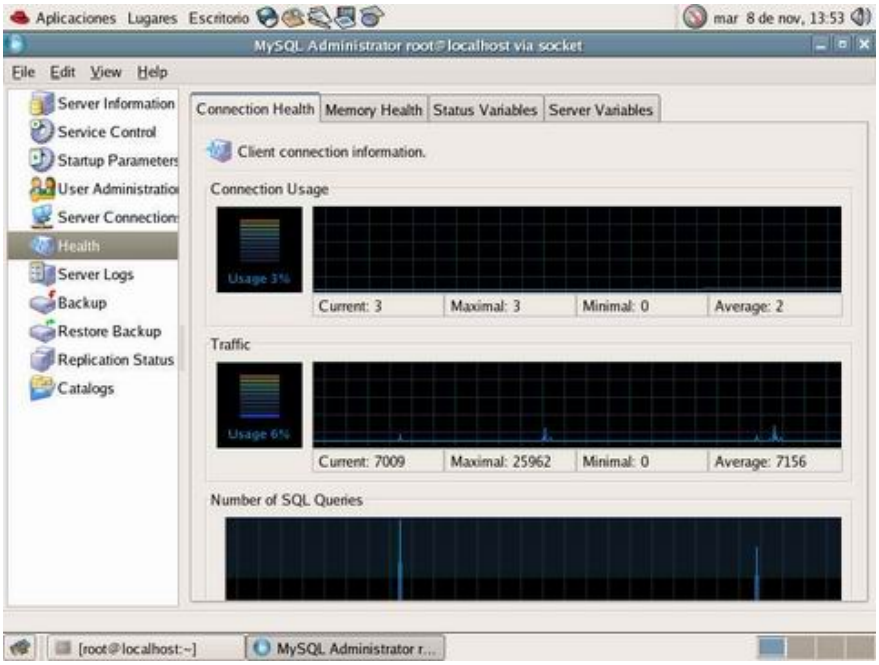

Figura 3.4. *Monitorización del estado del servidor.*

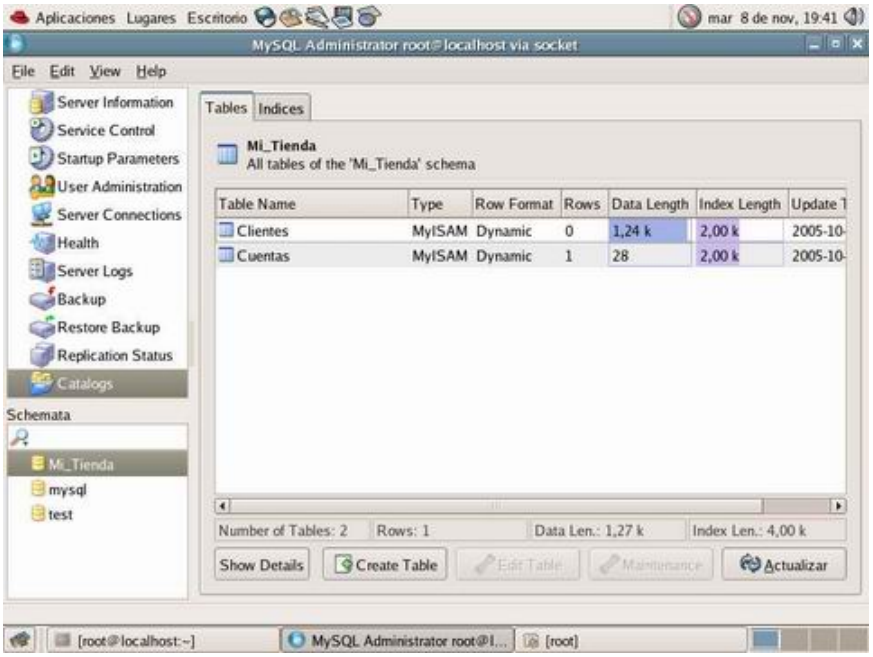

Figura 3.5. *Administración de tablas.*

3.8.8. Localización y juegos de caracteres.

3.8.8.1.Juegos de caracteres.

Por defecto, MySQL emplea el juego de caracteres cp1252 (Latin1). Todas las distribuciones binarias de MySQL se compilan con la opción:

--with-extra-charsets=complex

Esto añade el código necesario a todos los programas estándar para habilitar el uso de los todos juegos de caracteres mutlti-byte, además del Latin1. Otros juegos de caracteres se cargan desde un fichero de juego de caracteres cuando se necesitan. El juego de caracteres empleado determina qué caracteres se permiten en los nombres, y cómo se deben ordenar los resultados en sentencias SELECT que empleen las cláusulas ORDER BY y GROUP BY.
MySQL proporciona diversos juegos de caracteres, que pueden cambiarse cuando se inicia el servidor. Los juegos de caracteres disponibles dependen de la opción de configuración *--with-extra-charsets* y de los ficheros de configuración de juegos de caracteres listados en el directorio SHAREDIR/Charsets/Index. Los juegos de caracteres disponibles en MySQL se muestran en el siguiente listado.

- big5
- dec8
- cp850
- hp8
- koi8r
- latin1
- latin2
- swe7
- ascii
- ujis
- sjis
- hebrew
- tis620
- euckr
- koi8u
- gb2312
- greek
- cp1250
- gbk
- latin5
- armscii8
- utf8
- ucs2
- cp866
- keybcs2
- macce
- macroman
- cp852
- latin7
- cp1251
- cp1256
- cp1257
- binary
- geostd8
- cp932
- eucjpms

3.8.8.2.Localización.

A partir de la versión 4.1.3 de MySQL el servidor mantiene varios parámetros relativos a la zona horaria:

- La zona horaria del sistema. Cuando el servidor MySQL arranca, intenta determinar la zona horaria de la máquina anfitrión, y usa esta para establecer la variable del sistema *system_time_zone*.
- La zona horaria actual del sistema. La variable global del sistema *time_zone* indica la zona horaria en la que el servidor MySQL está trabajando actualmente. El valor inicial es ‘SYSTEM’, que indica que esta operando en la misma zona horaria que el sistema operativo.
- Zonas horarias por conexión. Cada cliente que se conecta tiene su propia zona horaria, dada por la variable de sesión *time_zone*. Inicialmente el contenido es el mismo que el de la variable global.

Los valores de zonas horarias se pueden especificar como un *offset* a partir del UTC, ó bien con nombres de zonas horarias (p.ej. ‘Europe/Helsinki’, ‘US/Eastern’ o ‘MET’).

3.9. IMPORTAR Y EXPORTAR DATOS.

La importación de datos es un proceso importante en la gestión de una base de datos, puesto que es el proceso que permite poblar una base de datos. Existen diversos

métodos para introducir datos en las tablas de una base de datos, pero no todos ellos son igual de sencillos e intuitivos. En este apartado se explicarán algunos de los métodos que proporciona MySQL para introducir datos en tablas, bien sea introduciéndolos directamente desde la línea de comandos o importándolos desde otras bases de datos. Otra forma bastante habitual de introducir los datos es a través de alguna aplicación cliente, que empleará alguna secuencia de comandos SQL para que un usuario pueda introducir los datos.

En este apartado también se abordará el proceso contrario, es decir, la exportación de datos desde MySQL hacia otras bases de datos. Esto resulta de gran utilidad si se desea migrar hacia otra base de datos o para hacer copias de seguridad de alguna base de datos existente.

Existen diversas formas de importar datos desde una base de datos existente a otra en MySQL. La forma más común de hacerlo es mediante el BCP (Protocolo de Copia Masiva), puesto que son muchas las bases de datos que utilizan este protocolo. Otro método común para importar datos es a través del proceso de conversión de archivos, mediante el cual, un archivo que está escrito en una base de datos se convierte a otro formato adecuado para la nueva base de datos. Otra forma más es exportar los datos de una base de datos a un formato que otra base de datos pueda leer (es similar a BCP, pero mas rudimentario).

Una de las formas más recientes de transferir datos es con el XML (Lenguaje Extensible de Marcado). Este es un lenguaje similar a HTML, pero es más robusto. XML ofrece una forma fácil y confiable de transferir datos a través de una red.

Como se puede ver, existen diferentes formas de poblar una base de datos con MySQL, pero no todas ellas resultan igual de sencillas y cómodas para el usuario. A continuación se verán algunas de ellas.

3.9.1. Herramientas para importar datos.

En MySQL se puede poblar una base de datos de diferentes maneras. La primera forma que se puede pensar es introduciendo los datos directamente desde la línea de comandos (bastante engorroso). Pero existen otros métodos que facilitan esta tarea.

3.9.1.1. Introducir datos desde la línea de comandos.

Desde el monitor MySQL se pueden introducir datos mediante el comando INSERT de SQL. Este comando permite añadir nuevas filas de datos a una tabla existente. La sintaxis es la siguiente:

INSERT INTO *nombre_tabla*(*campo1*, *campo2*, *campo3*, …)
VALUES (*valor_campo1*, *valor_campo2*, *valor_campo3*, …);

No es necesario especificar todos los campos de la tabla, sino solo aquellos que se quieran introducir. Normalmente, este comando se usa en secuencias SQL empleadas por alguna API, al igual que otros comandos SQL (SELECT, UPDATE, DELETE,…), pero no se suele utilizar para introducir datos directamente en una base de datos.

3.9.1.2.mysqlimport.

MySQL tiene una herramienta hecha especialmente para cargar datos en una tabla. Es un poco rústica, pero efectiva. Esta herramienta se llama *mysqlimport*. Esta herramienta se utiliza a través de la línea de comandos de Linux, y toma dos parámetros más una serie de opciones. Con esta herramienta se puede coger un archivo de texto e importarlo a la tabla que se desee. Lo que hace esta herramienta es coger el contenido de un archivo de texto y cargarlo en la tabla especificada por el nombre del archivo hasta el primer punto. El comando que se deberá introducir desde la línea de comandos de Linux es:

mysqlimport *nombre_bd nombre_tabla*.txt [opciones]

En la orden anterior, *nombre_bd* es el nombre de la base de datos donde se encuentra la tabla en la que se quiere introducir los datos y *nombre_tabla* es el nombre de la tabla en cuestión. Hay que hacer especial énfasis en que el nombre del archivo de texto debe ser exactamente igual (respetando mayúsculas y minúsculas) al de la tabla donde se quieren introducir los datos.

El formato que deberá tener el archivo de texto viene determinado por las opciones con las que se llame a este comando.

Las opciones que se pueden emplear con *mysqlimport* se muestran en la tabla 3.22:

Opción	Acción
-d	Borra todos los datos existentes antes de importar los nuevos.
-f	Fuerza a continuar insertando datos sin importar los errores que pueda encontrar.
-i	Salta o ignora las filas que comparten un único número.
-L	Fuerza a usar un archivo de la máquina local, y no de la máquina donde se encuentra el servidor MySQL (en caso de que no sea la local).
-l	Bloquea cada tabla antes de que cada dato sea insertado.
-r	Reemplaza el campo de la tabla que comparte el mismo valor único.
--fields-enclosed-by=char	Especifica qué carácter delimita los datos en el archivo. De manera predeterminada, MySQL asume que no hay caracteres delimitadores.
--fields-escaped-by=char	Indica cuál es el carácter de “escape” de entre los caracteres especiales. El valor predeterminado es sin carácter de escape.
--fields-terminated-by=char	Especifica qué carácter separa los valores uno a uno. Como valor predeterminado se asume el tabulador
--lines-terminated-by=str	Especifica con qué se termina una línea de registros. Por defecto se asume el carácter de avance de línea. Puede utilizarse tanto un carácter como una cadena de caracteres.

Tabla 3.22. *Opciones de mysqlimport.*

Aunque es un poco rudimentaria, esta herramienta puede servir para introducir en nuestra base de datos elementos que se han exportado de hojas de cálculo, bases de datos u otras aplicaciones en un determinado formato. No obstante, es un proceso bastante engorroso y propenso a errores.

3.9.1.3.Procesamiento por lotes.

Otra forma que tiene MySQL de importar datos es aprovechando la característica del procesamiento por lotes. Esto se vio en un apartado anterior cuando se comentó el uso de MySQL en modo *batch*. Básicamente lo que se hace es englobar en un archivo de texto todas las órdenes que se deberían teclear desde la línea de comandos del monitor MySQL. Para introducir los datos se tendrían que tener varias líneas con la sentencia INSERT INTO. El archivo de texto que se cree se debe guardar con extensión .sql y posteriormente, una vez que se esté ejecutando el servidor MySQL, se debe introducir el siguiente comando desde el *shell* de Linux:

mysql –p < /*ruta_archivo*/*nombre_archivo*.sql

3.9.1.4.LOAD DATA INFILE.

Esta forma de importar datos a la base de datos MySQL es muy similar a *mysqlimport*, pero este método permite ejecutarlo desde la línea de comandos del monitor MySQL. Esto significa que también se puede ejecutar desde cualquier otra aplicación que se haya escrito mediante alguna API, pudiendo así importar los datos desde alguna aplicación que se haya desarrollado.

Así, para importar los datos con este comando, desde el monitor MySQL se tendría que utilizar el siguiente comando (el servidor MySQL debe estar ejecutándose y la base de datos que se quiere poblar debe estar en uso):

LOAD DATA INFILE "/*ruta_archivo*/*nombre_archivo*.txt"
INTO TABLE *nombre_tabla*;

Este comando posee algunos parámetros opcionales, al igual que los tenía *mysqlimport*. Algunos de estos parámetros se muestran en negrita en la tabla 3.23:

Parámetro	Acción
LOAD DATA **LOCAL** INFILE	Fuerza a usar un archivo de la máquina local, y no de la máquina donde se encuentra el servidor MySQL.
LOAD DATA **LOW_PRIORITY** INFILE	MySQL esperará a que nadie lea la tabla para insertar los datos.
REPLACE INTO TABLE	Reemplaza el campo de la tabla que comparte el mismo valor único.
… **TERMINATED BY** 'carácter'	Describe lo que separa a los campos del archivo. El valor predeterminado es el tabulador.
… **ENCLOSED BY** 'carácter'	Describe lo que encierra a los caracteres.
… **ESCAPED BY** 'carácter'	Describe los caracteres de escape. Por defecto es la diagonal invertida (\).

Tabla 3.23. *Parámetros opcionales de LOAD DATA INFILE.*

Una ventaja de este método respecto a *mysqlimport* es la capacidad de referirse a columnas específicas dentro de las cuales se van poner los datos. Esto puede resultar interesante si se desea hacer una importación parcial de los datos. Para hacer esto, bastará con que se pongan detrás del nombre de la tabla, entre paréntesis, los nombres de las columnas en las que se quieren introducir los datos, separados por comas.

3.9.1.5.Importar datos desde aplicaciones cliente.

Los métodos vistos hasta ahora para poblar una tabla de datos pueden resultar bastante engorrosos. En general, MySQL en sí puede resultar complicado e incómodo para usuarios que no estén acostumbrados a un entorno de línea de comandos. Para facilitar las cosas existe software adicional que dota a MySQL de un entorno gráfico para la gestión y el manejo de bases de datos. Este es el caso de MySQL Administrator, que se verá mas adelante. Otra posibilidad para importar e introducir datos de forma gráfica es utilizar MySQL con alguna base de datos más conocida.

Para conectar MySQL con otras bases de datos, como por ejemplo OpenOffice Base (que es el equivalente de libre distribución a la popular base de datos de Microsoft Ms Access), se puede emplear el conector MyODBC. MySQL proporciona soporte para ODBC a través de los controladores MyODBC.

ODBC (*Open Database Conectivity*) proporciona un método para que programas cliente accedan a un amplio rango de bases de datos o fuentes de datos. ODBC es una API estandarizada que permite conexiones a servidores de base de datos basadas en SQL.

Para poder utilizar MyODBC se deben tener instalados los controladores apropiados y las librerías MySQL correspondientes. Estos controladores y librerías vienen en la práctica totalidad de las distribuciones de Linux, y se instalan junto con MySQL. De todas formas, si no se tienen instalados, se pueden descargar desde varios sitios e instalar fácilmente.

Una vez que se cumplen todos los requisitos, hay que hacer unas operaciones para poder utilizar MyODBC. Lo primero que se debe hacer es configurar los ficheros .ini utilizados por ODBC. Estos ficheros son sólo ficheros de texto que se pueden editar usando cualquier editor. Los ficheros a editar son dos:

/etc/odbcinst.ini
Este fichero lista los controladores ODBC instalados en el sistema. Si no está definido el controlador MySQL, habrá que definirlo. El contenido debe ser éste:

```
# Driver from the MyODBC package
# Setup from the unixODBC package
[MySQL]
Description     = ODBC for MySQL
Driver          = /usr/lib/libmyodbc.so
Setup           = /usr/lib/libodbcmyS.so
FileUsage       = 1
```

/etc/odbc.ini
Este archivo define los enlaces que los usuarios usarán para enlazar las bases de datos. Los enlaces en odbc.ini están disponibles para todos los usuarios del sistema. El contenido de este fichero debe ser algo como lo siguiente:

```
[Nombre_bd]
Description     = "descripción"
```

Driver = MySQL
Server = localhost
Port = 3306
Database = *Nombre_bd*

Para cada base de datos que se utilice habrá de existir una entrada de la forma anterior en el archivo odbc.ini.

Una vez que se tienen instalados todos los controladores y librerías necesarios y que se tienen correctamente configurados los archivos .ini correspondientes, se puede proceder a configurar la conexión de alguna aplicación cliente con MySQL. Una de estas aplicaciones cliente podría ser OpenOffice Base, que presenta un asistente bastante sencillo e intuitivo para configurar la conexión.

Una vez hecha la conexión entre la aplicación cliente y MySQL, se podrán realizar operaciones sobre bases de datos MySQL a través de la aplicación cliente.

3.9.2. Herramientas para exportar datos.

Como se ha podido ver en el apartado anterior, MySQL tiene una gran variedad de utilerías de importación. Pero también se debe tener en cuenta el paso contrario, es decir, la exportación de datos desde MySQL hacia otras bases de datos

Existen diversas formas de exportar los datos. Todas son bastante similares a la forma en que se importan los datos. Esto, en definitiva, es bastante lógico, pues únicamente se trata de un cambio de perspectiva.

3.9.2.1.mysqldump

La utilería *mysqldump* es la contrapartida de *mysqlimport*. Las dos comparten opciones comunes, pero *mysqldump* hace algunas cosas más. Con esta herramienta se puede tomar toda la base de datos y volcarla en un solo archivo de texto. Este archivo de texto contendrá todos los comandos SQL necesarios para recrear la base de datos. Se puede decir que esta herramienta hace una ingeniería inversa de la base de datos. Como todo está en un archivo de texto, se puede importar de nuevo la base de datos con un simple procesamiento por lotes. Para hacer este volcado de una base de datos hacia un fichero de texto se deberá introducir el siguiente comando desde el *shell* de Linux:

mysqldump –p *nombre_bd* > *nombre_archivo*.txt

Donde *nombre_bd* es el nombre de la base de datos que se desea exportar y *nombre_archivo* es el nombre del archivo que contendrá el volcado de la base de datos.

Si sólo se quiere exportar una tabla en concreto, bastará con especificar el nombre de la tabla después del de la base de datos, separándolos con un espacio en blanco

Las opciones disponibles para esta utilidad se muestran en la tabla 3.24.

Opción	Acción
--add-drop-table	Se añadirá una instrucción DROP TABLE IF EXIST antes de cada tabla, para así evitar errores posibles en la importación.
--add-locks	Protege las instrucciones INSERT con comandos LOCK TABLE y UNLOCK TABLE. Previene que los usuarios puedan hacer algo a la tabla mientras se reintroducen los datos.
-c	Se nombra cada columna en la instrucción INSERT.
--delayed-insert	Se usa la opción DELAYED en los comandos INSERT.
-F	Se vaciarán los archivos de registro del servidor MySQL antes de ejecutar el vaciado.
-f	Se fuerza a que se continúe vaciando aunque se produzcan errores.
-full	Se añade información adicional a las instrucciones CREATE TABLE.
-l	El servidor bloquea las tablas que están siendo vaciadas.
-t	Evita que *mysqldump* escriba cualquier instrucción CREATE TABLE. Esto puede ser útil si sólo se quieren los datos.
-d	Se evita que se escriba cualquier instrucción INSERT. Esto se podría usar si lo que interesa es la estructura de la base de datos, y no los datos que contiene.
--opt	Activa todas las opciones que ayudaran a acelerar el proceso de vaciado y crear un archivo más rápido de recarga.
-q	Evita que MySQL lea todo el archivo de vaciado y después lo ejecute. En vez de eso, escribirá al archivo conforme lo vaya leyendo.
-T *ruta*	Se crearán dos archivos. Uno contendrá la tabla de generación de instrucciones y el otro contendrá los datos.
-w= "*condición*"	Sirve para filtrar los datos que se van a exportar.

Tabla 3.24. *Opciones de mysqldump.*

Como se puede comprobar, *mysqldump* es una herramienta bastante potente que permite volcar total o parcialmente el contenido de una base de datos en un fichero de texto.

3.9.2.2.SELECT INTO OUTFILE.

MySQL también posee un comando que puede considerarse como la contrapartida de LOAD DATA INFILE. Este comando es SELECT INTO OUTFILE. Los dos comandos tienen mucho en común. En primer lugar, comparten las mismas opciones, la única diferencia es que uno importa los datos y otro los exporta. Para utilizar este comando se deberá emplear desde el monitor MySQL la siguiente sintaxis:

SELECT *condición* INTO OUTFILE */ruta/nombre_archivo*.txt [OPCIONES]

Esta instrucción es como un SELECT común, excepto que en vez de mostrar el resultado por pantalla lo vuelca a un archivo de texto. Esto significa que se pueden hacer consultas avanzadas usando JOINs y múltiples tablas. Las opciones que se pueden utilizar son las mismas que para LOAD DATA INFILE

3.9.2.3.Exportar datos a aplicaciones cliente.

Al igual que se pueden utilizar aplicaciones cliente como OpenOffice Base para introducir datos en una tabla, se podrá aprovechar la conexión que permite ODBC entre estas dos bases de datos para volcar los datos de las tablas de una base de datos MySQL hacia otra base de datos. El proceso para hacer esto es bastante simple. Una vez que ya se tiene correctamente configurada la conexión tal y como se vio en un apartado anterior, desde la aplicación cliente se podrá acceder al contenido de las bases de datos MySQL.

3.10. SEGURIDAD.

La seguridad es un componente fundamental en cualquier sistema gestor de bases de datos. Un buen sistema de gestión de bases de datos deberá tener las herramientas adecuadas para asegurar la seguridad en las bases de datos.

MySQL emplea un sistema de seguridad basado en listas de control de acceso (ACLs) para todas las conexiones, consultas y operaciones que los usuarios intenten realizar. También posee soporte para conexiones encriptadas mediante SSL entre clientes y servidor.

Anteriormente, en el apartado “usuarios” se comento cómo funciona de manera bastante simple el sistema de usuarios en MySQL. En este apartado se profundizará mucho más en este aspecto, y se verá como este sistema de gestión de usuario permite implementar un sistema de seguridad a varios niveles.

El sistema de seguridad de MySQL es muy flexible. Con el se pueden asignar diferentes niveles de acceso a los usuarios. Se puede asignar desde qué máquina se puede conectar un usuario, si tiene acceso a todo el sistema de bases de datos, o solo a determinadas bases de datos, a determinadas tablas o incluso sólo a ciertas columnas. Además, se puede especificar qué tareas en concreto puede realizar el usuario sobre las bases de datos, tablas y/o columnas.

3.10.1. Tablas de control de acceso.

Todos los permisos y privilegios de los usuarios están contenidos en la base de datos llamada *mysql*. Esta base de datos se crea automáticamente con la instalación de MySQL. Esta base de datos es muy importante, pues la pérdida total o parcial de datos en esta base de datos podría ocasionar graves problemas para el funcionamiento de MySQL. Por tanto, esta debe ser una base de datos a la cual sólo tengan acceso los administradores de la base de datos. El número de tablas que contiene la base de datos *mysql* depende de la versión de MySQL, pero todas las versiones tienen en común cinco tablas, que son las más importantes de cara a la seguridad. Estas cinco tablas son: *user*, *db*, *host*, *tables_priv* y *columns_priv*, y se conocen como tablas de permiso de acceso. Cada columna de estas tablas muestra los permisos que tiene una persona para realizar una operación determinada (almacenando Y o N). A continuación se detalla el contenido de estas tablas.

3.10.1.1. Tabla *user*.

Esta tabla contiene los datos de los permisos de todos los usuarios que tienen acceso a MySQL. Las columnas que posee esta tabla se muestran a continuación.

Columna	Descripción
Host	Nombre de la computadora del usuario. Con esto se puede restringir el acceso de una persona basándose en la ubicación desde la cual se está conectando.
User	Es el nombre de usuario.
Password	Contraseña de usuario.

Columna	Descripción
Select_priv	Permite que el usuario realice consultas SELECT.
Insert_priv	Permite que el usuario agregue datos a la base de datos por medio de instrucciones INSERT.
Update_priv	Permite al usuario editar las tablas de las base de datos con la instrucción UPDATE.
Delete_priv	Permite que el usuario elimine datos de las bases de datos con la instrucción DELETE.
Create_priv	Permite que el usuario agregue tablas y bases de datos al servidor.
Drop_priv	Permite que el usuario elimine tablas y bases de datos del servidor.
Reload_priv	Permite que el usuario actualice las tablas de permiso de acceso empleando la instrucción FLUSH.
Shutdown_priv	Permite que el usuario apague el servidor.
Proccess_priv	Permite que el usuario consulte los procesos de MySQL empleando el comando *mysqladmin proccesslist* o la instrucción SHOW PROCCESSLIST.
File_priv	Permite que el usuario lea y escriba archivos residentes en el servidor.
Grant_priv	Permite que el usuario conceda privilegios a otros usuarios.
Referentes_priv	Actualmente sin uso.
Index_priv	Permite que el usuario cree o elimine índices en las tablas.
Alter_priv	Permite que el usuario cambie la estructura de una tabla.

Tabla 3.25. *Columnas de la tabla user.*

Al otorgar permisos a este nivel, se le proporciona al usuario acceso global a la base de datos, es decir, que si un usuario determinado tiene un privilegio determinado para realizar alguna tarea, la podrá realizar sin ningún tipo de restricción en cualquier base de datos. Para limitar el acceso de un usuario sólo a determinadas bases de datos, o a determinadas tablas, o incluso a determinadas columnas, se emplean las otras tablas.

3.10.1.2. Tabla *db*.

La tabla *db* contiene los permisos para todas las bases de datos contenidas en el servidor MySQL. Los permisos otorgados en esta tabla son solamente para la base de datos seleccionada. Las columnas de esta tabla son prácticamente las mismas que las de la tabla *user*, pero con algunas excepciones. Al regir los permisos a nivel de bases de datos, no existen privilegios del nivel de administrador como pueden ser *Reload_priv*, *Shutdown_priv*, *Proccess_priv* y *File_priv*. Por tanto, las columnas correspondientes a esos privilegios no aparecen en la tabla *db*. Además, se incluye una nueva columna, denominada *Db*, la cual indica la base de datos para la cual son aplicables los privilegios.

3.10.1.3. Tabla *host*.

La tabla host controla el acceso al limitar los host que pueden conectarse a la base de datos. Esta tabla contiene las mismas columnas que la tabla *db*.

3.10.1.4. Tabla *tables_priv*.

Esta tabla rige los permisos para la tabla de una determinada base de datos. Mediante esta tabla se puede limitar lo que un usuario puede hacer a alguna tabla en concreto de una base de datos determinada. La tabla *tables_priv* contiene las siguientes columnas:

Columna	Descripción
Host	Host desde el que se conecta el usuario.

Columna	Descripción
Db	La base de datos que contiene las tablas a las que se le están aplicando privilegios
User	Nombre de usuario de la persona a la que se le conceden los permisos.
Table_name	Nombre de la tabla de la base de datos en la que se están estableciendo los permisos.
Column_priv	Columna que controla el acceso que tiene el usuario a cualquier columna. Puede contener los siguientes valores: SELECT, INSERT, UPDATE y REFERENCES. Si se quiere conceder más de un privilegio, se han de separar por comas.
Timesamp	Contiene el registro de la hora en la que se hicieron cambios.
Grantor	Contiene el nombre de la persona que concede los permisos.
Table_priv	Contiene los permisos para la tabla. Los valores que puede almacenar son: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT, REFERENCES, INDEX y ALTER. Si se especifica más de un privilegio, se han de separar por comas.

Tabla 3.26. *Columnas de la tabla tables_priv.*

3.10.1.5. Tabla *columns_priv*.

La tabla *columns_priv* contiene los permisos para una determinada columna de una tabla de la base de datos. Los campos que contiene esta tabla son los mismos que los de la tabla *tables_priv*, salvo las columnas *Grantor* y *Tables_priv*, que no aparecen *columns_priv*. Además, esta tabla posee una columna que no aparece en *tables_priv*. Esta columna es *Column_name*, que indica el nombre de la columna sobre la que se van a aplicar los privilegios. Dichos privilegios son los que se muestran en la columna *Column_priv*.

3.10.2. Implementación de la seguridad en MySQL.

En el sistema de bases de datos MySQL existe una jerarquía de seguridad. Cuando un usuario se conecta a la base de datos, MySQL busca en la tabla *user* para verificar si el usuario y host existen en la base e datos, y si la contraseña introducida coincide. Si el usuario y host existen, y la contraseña introducida es correcta, se le proporciona al usuario acceso al sistema. Este es, por tanto, un primer nivel de seguridad. Existe otro nivel de seguridad, que se aplica cuando un usuario envía una consulta a la base de datos. Cada vez que un usuario envía una consulta, lo primero que hace MySQL es consultar la tabla *user* para ver qué privilegios tiene ese usuario. Si el usuario no tiene privilegios en esa tabla, entonces MySQL consulta *db*. Nuevamente verifica en esa tabla una coincidencia con el nombre de usuario, *host* y *password*. Si hay coincidencia, MySQL consultara los privilegios de ese usuario, si no tiene los privilegios necesarios, entonces MySQL explorará la tabla *tables_priv* de la misma forma, y después la tabla *columns_priv*. Si no puede encontrar permiso en ninguna tabla, generará un error. Como se puede ver, existen dos puntos de control. Uno es la verificación de la conexión y otro es la verificación de la solicitud. Estos dos puntos de control proporcionan un entorno muy seguro para la base de datos. A continuación se verá con un poco mas de detalle estos dos niveles de seguridad.

3.10.2.1. Verificación de la conexión.

La verificación de la conexión ocurre en el momento en que un usuario intenta conectarse a MySQL. Toda conexión a MySQL requiere un nombre de usuario, un *host* y una contraseña. El nombre de usuario es el nombre de la persona que está tratando de conectarse. La contraseña es una herramienta de verificación adicional para asegurar

que la persona que se esta conectando a la base de datos es quien dice ser. El nombre de *host* es el nombre de la computadora desde la cual se está conectando el usuario. MySQL no solo puede limitar la conexión de una persona, sino que también puede restringir la conexión de una máquina.

El proceso de verificación de la conexión es un proceso bastante sencillo. MySQL verifica la solicitud contra la información de la tabla de permisos de acceso *user* en busca de una coincidencia con el nombre de usuario, el nombre del *host* y el *password*. Si MySQL no encuentra ninguna coincidencia dentro de la taba *user*, deniega el acceso. En la siguiente figura se muestra gráficamente este proceso. Cabe comentar que MySQL encripta las claves para transmitirlas empleando su propio algoritmo. Este método es diferente del usado en el proceso de identificación de usuario de Unix, y es el mismo que el implementado por la función SQL PASSWORD().

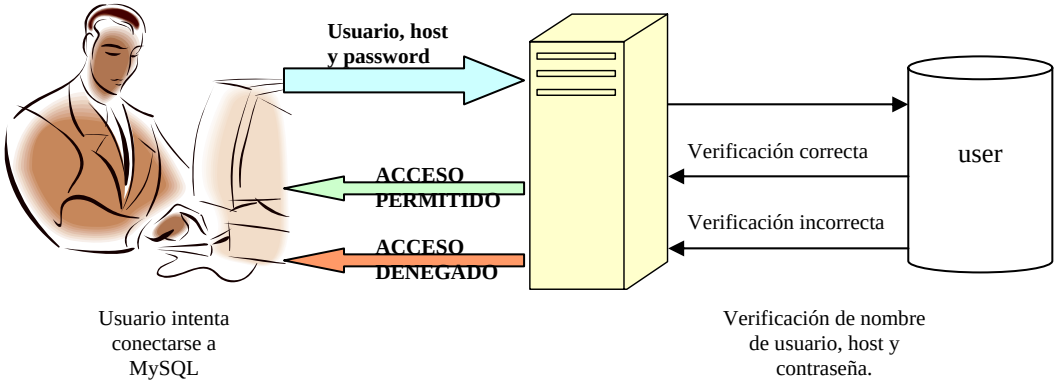

Figura 3.6. *Verificación de la conexión en MySQL.*

3.10.2.2. Verificación de la solicitud.

La verificación de la solicitud ocurre cada vez que un usuario envía una consulta a la base de datos. Cada comando enviado después de realizar una conexión pasa por el mismo proceso. Esto asegura que los usuarios trabajarán con la base de datos bajo las condiciones que les han sido impuestas. También proporciona una barrera de seguridad adicional en el caso de que alguna persona ajena consiga entrar en la base de datos.

El proceso de verificación es muy sencillo, siendo muy similar al anterior. Siempre que se envía una solicitud, MySQL verifica los permisos que tiene el usuario en la tabla *user*. Si la persona tiene permisos en esta tabla para ejecutar el comando, podrá hacerlo para cualquier base de datos. Si MySQL no encuentra los privilegios necesarios para llevar a cabo el comando, consulta la tabla *db*.

La tabla *db* es la siguiente línea de seguridad. Los permisos concedidos aquí sólo son aplicables a la base de datos especificada. Si el usuario no tiene los permisos para ejecutar el comando en la base de datos seleccionada, se pasa a verificar la tabla *tables_priv*. Si los privilegios necesarios se encuentran aquí, MySQL ejecuta la consulta. En caso contrario, MySQL busca en última instancia en la tabla *columns_priv*. Esta

tabla es la última que verifica MySQL. Si el usuario no tiene permiso aquí, MySQL devuelve un error, indicando que la solicitud es denegada.

A pesar de contener varios niveles, y de chequear varia tablas, este proceso se realiza muy rápidamente, tanto que el rendimiento de MySQL no se ve afectado. En la figura 3.7 se muestra gráficamente este proceso.

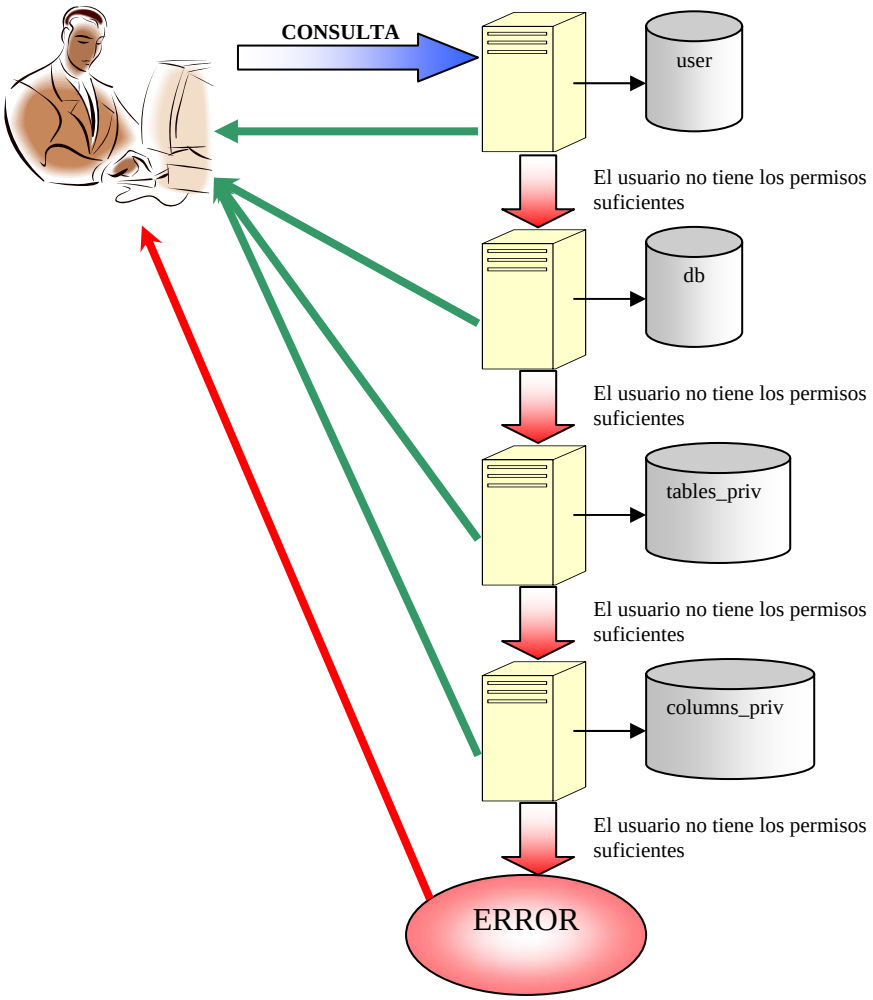

Figura 3.7. *Verificación de solicitud en MySQL*

3.10.3. Modificación de los privilegios de usuario.

Al igual que la mayoría de tareas que se pueden hacer en MySQL, existen varias formas de crear, editar y eliminar los privilegios de acceso de un usuario. En este apartado se verán algunas de estas formas.

3.10.3.1. Tablas de permiso de acceso.

Esta es la forma más difícil de cambiar los permisos de acceso de un usuario. Como las tablas de la base de datos *mysql* son como el resto de las tablas, se pueden utilizar sobre ellas las instrucciones INSERT, DELETE, UPDATE y SELECT. Por tanto, con estas instrucciones se puede agregar un usuario, eliminarlo, asignarle privilegios, modificarlos y revocarlos.

3.10.3.2. El comando GRANT.

Actualizar los permisos de acceso escribiendo directamente en las tablas de permisos de acceso puede resultar una tarea bastante incómoda, además de ser un método bastante propenso a errores. Por eso, MySQL proporciona un comando para asignar y modificar privilegios a usuarios. Este es el comando GRANT. La sintaxis del comando GRANT es la siguiente:

```
GRANT [privilegios] (nombres de columnas)
ON nombre_bd.nombre_tabla
TO nombre_usuario@nombre_host
IDENTIFIED BY "contraseña"
[WITH GRANT OPTIONS];
```

Después de la palabra clave GRANT se deben listar todos los privilegios que se desean conceder al nuevo usuario. Los privilegios que se le pueden conceder se muestran en la tabla 3.27.

Privilegio	Descripción
ALL	Todos los permisos disponibles.
ALTER	Modificación de tablas, columnas e índices.
CREATE	Crear tablas o bases de datos.
DELETE	Borrar registros de las tablas.
DROP	Borrar tablas o bases de datos.
FILE	Leer y escribir archivos en el servidor
INDEX	Agregar o eliminar índices.
INSERT	Agregar registros a la base de datos.
PROCESS	Visualizar y cancelar procesos del sistema MySQL.
REFERENCES	Sin uso actualmente.
RELOAD	Uso de la instrucción FLUSH.
SELECT	Efectuar consultas SELECT.
SHUTDOWN	Apagar el servidor.
UPDATE	Editar los registros existentes en la base de datos.
USAGE	Permite que el usuario se conecte al servidor. Este usuario no cuenta con permiso alguno.

Tabla 3.27. *Lista de privilegios disponibles para los usuarios de MySQL.*

Después de los privilegios viene una parte opcional, que permite especificar las columnas de una tabla a la que se le van a aplicar los privilegios. Esto se usará en el caso de que sólo se quiera otorgar a un usuario privilegios para trabajar únicamente con determinadas columnas. Después de la palabra clave ON, se debe listar la base de datos y la tabla o tablas a las que se les desean aplicar los privilegios. Para especificar todas las tablas de una base de datos se puede utilizar el carácter comodín *. Por ejemplo, se podría indicar *nombre_bd*.*, que haría referencia a todas las tablas de la base de datos

nombre_bd. Si se quieren aplicar permisos globales se podría utilizar *.*. A la definición de las bases de datos y tablas le sigue la cláusula TO. Aquí es donde se debe indicar el usuario y el nombre de *host*. Aquí puede utilizarse el carácter comodín % para identificar subdominios completos. Al nombre de usuario y al nombre de *host* le sigue la cláusula IDENTIFIED BY, tras la cual se debe especificar la clave de acceso del usuario. No es necesario emplear la función PASSWORD para codificar la clave, el comando GRANT se encarga de codificarla. La última parte de la instrucción (WITH GRANT OPTIONS) es completamente opcional. Si se incluye se está dando al usuario la posibilidad de utilizar el comando GRANT.

Para eliminar usuarios y revocar privilegios, MySQL posee la herramienta opuesta al comando GRANT. Se trata del comando REVOKE. La sintaxis de este comando es la siguiente:

```
REVOKE [privilegios] (columnas)
ON nombre_bd.nombre_tabla FROM nombre_usuario@nombre_host;
```

Los privilegios que pueden revocarse con los mismos que se detallaron en la tabla 3.24. Como se puede ver, el comando REVOKE es muy parecido al comando GRANT. Con el comando REVOKE se pueden emplear los mismos caracteres comodines que con GRANT.

3.10.3.3. mysql_setpermission.

Otra forma para conceder permisos a los usuarios es el uso de *mysql_setpermission* desde el *shell* de Linux. Esta herramienta consta de una serie de comandos escrita en Perl que permite establecer los permisos de usuario a través de una navegación por una serie de menús. Al ejecutar esta función, lo primero que se pide es el *password*. Una vez que se haya introducido correctamente la clave aparecerán una serie de menús que permitirán el establecimiento del *password* de un determinado usuario, añadir privilegios para todo excepto para tareas administrativas para una base de datos existente, crear una base de datos y añadir privilegios para todo excepto para tareas administrativas, añadir privilegios para todo excepto para tareas administrativas, CREATE y DROP para una base de datos existente, crear una base de datos y añadir privilegios para todo excepto para tareas administrativas, CREATE y DROP.

3.10.3.4. MySQL Administrator.

En el apartado de administración de MySQL, se describió el programa MySQL Administrator. Como se vio, este programa posee una sección bastante clara e intuitiva para crear y eliminar usuarios. Para asignar y editar privilegios, solo habrá que situarse en la pestaña “*Schema Privileges*”. Desde ahí simplemente habrá que seleccionar los permisos deseados en la columna “Privilegios Disponibles” y arrastrarlos a la columna “Privilegios Asignados”. Esta es la forma más sencilla para administrar los usuarios en una base de datos MySQL.

3.10.4. Consideraciones adicionales.

A pesar de los mecanismos vistos anteriormente para mejorar la seguridad en un sistema de bases de datos basado en MySQL, se deben adoptar otros métodos adicionales para

aumentar el nivel de seguridad, sobre todo cuando se trata de bases de datos grandes, importantes y bastante confidenciales. Para administrar bases de datos de estas características hay que regirse por la máxima que indica que “toda precaución es poca”. Así, se podrían utilizar *Firewalls* u otras herramientas de seguridad para Linux. Además, se recomienda que nunca se ejecute *mysqld* como usuario *root*. El motivo principal es porque cualquiera que tenga privilegios FILE en MySQL puede acceder a cualquier archivo si el servidor esta corriendo como *root*. En lugar de ejecutar el servidor como *root* se podría ejecutar como otro usuario, tal como *mysqladmin* o *mysqlgroup* (son usuarios que se crean en la instalación de MySQL). Otra consideración a tener en cuenta para la seguridad es asegurar bien el subdirectorio de datos del directorio *mysql*, estableciendo los permisos oportunos desde Linux.

Por último, cabe comentar que el sistema de encriptación de claves en las últimas versiones de MySQL es bastante complejo, pues aumenta el número de bytes del *password* y hace que la función PASWORD() sea no repetitiva. Esto hace que el tráfico de datos conteniendo claves de usuario sea muy seguro. Además, si se desea seguridad adicional, se podría utilizar SSH para obtener una conexión TCP/IP encriptada entre el servidor MySQL y un cliente MySQL. Existen diversas herramientas de creación de clientes SSH, algunas de ellas *Open Source*.

3.11. INTERFACES

Para manipular una base de datos MySQL no sólo existen las herramientas que se han visto hasta ahora (monitor, MySQL Administrator,…). MySQL proporciona un gran número de interfaces para programar. Algo de esto ya se vio cuando se estudió la conexión entre MySQL y aplicaciones cliente.

Las interfaces que proporciona MySQL permiten construir aplicaciones que utilicen cualquier base de datos MySQL. Estas aplicaciones se conocen con el nombre de “*front-ends*”, mientras que el motor de la base de datos sobre la que interactúan se conoce como “*back-end*”.

Como ya se ha visto anteriormente, el servidor MySQL es un programa que se ejecuta continuamente en segundo plano. Para poder utilizar una base de datos que proporcione el servidor MySQL, una aplicación primero deberá establecer una conexión con dicho servidor. Esta conexión se establece a través de una interfaz de algún tipo. La interfaz es una capa entre el programa y la base de datos. Los comandos que se envían desde el programa van a través de la interfaz y son traducidos para la base de datos. La interfaz es una simple colección de funciones disponibles para el programador.

Las interfaces constan de dos partes. Una parte son las llamadas a funciones y variables que se usan en la aplicación y la otra es el intérprete o controlador. La primera parte es universal, es decir, que es común para todos los sistemas gestores de bases de datos. Lo único que es dependiente de la base de datos son los controladores. Estos controladores suelen ser creados por los fabricantes. En la figura 3.8 se puede ver un esquema del flujo de proceso para interfaces y controladores.

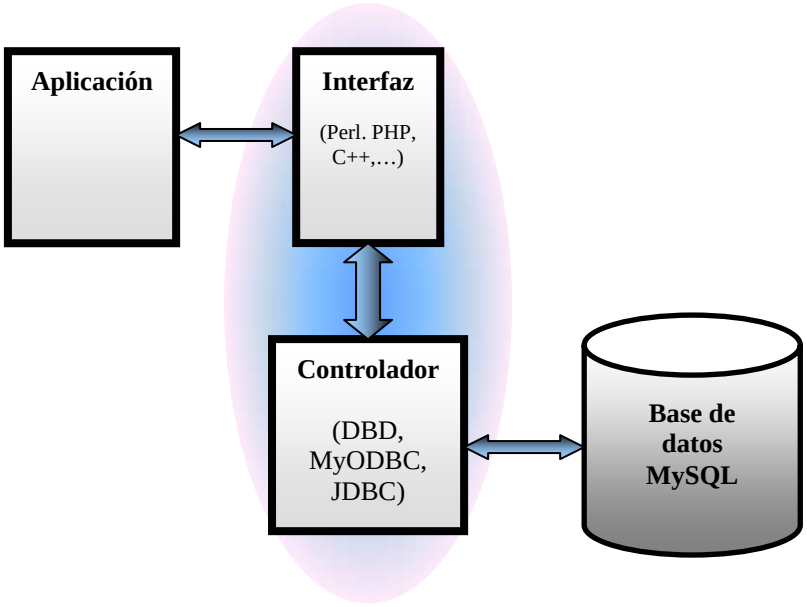

Figura 3.8. *Flujo de proceso para interfaces y controladores.*

MySQL posee un gran número de interfaces/controladores:

- Perl (DBD/DBI)
- PHP
- C/C++
- Python
- ODBC (MyODBC)
- Java (JDBC)
- NET
- MXJ
- Tcl
- Eiffel

Toda esta variedad de controladores y lenguajes utilizables da una idea del gran potencial que tiene MySQL para su empleo en cualquier tipo de aplicación.

Además de tener el controlador adecuado, para que una aplicación pueda acceder a una base de datos, el lenguaje en que se programa la aplicación deberá contener las funciones pertinentes para indicar los parámetros de la conexión a la base de datos (nombre o dirección del servidor MySQL, nombre de la base de datos que se quiere atacar, nombre de usuario y contraseña).

Todos estos interfaces y controladores disponibles hacen que MySQL sea un sistema gestor de bases de datos muy atractivo para aquellos desarrolladores y administradores de bases de datos que programen aplicaciones que necesiten acceso a bases de datos.

3.12. MOTORES DE ALMACENAMIENTO.

MySQL soporta varios motores de almacenamiento de datos. Los motores de almacenamiento de datos son las técnicas mediante las cuales una base de datos almacena los datos en memoria. Cada una de estas técnicas emplea diferentes características de indexado, almacenamiento y bloqueos. Dependiendo del propósito de la base de datos, será más recomendable el uso de un motor u otro. También se suelen conocer los motores de almacenamiento como tipos de tabla.

La flexibilidad que proporciona MySQL para almacenar datos es una de las características que han motivado la popularidad de MySQL como sistema gestor de base de datos. Para determinar los motores de almacenamiento disponibles, existe una

instrucción que se puede ejecutar desde el monitor MySQL. Esta instrucción es SHOW ENGINES. Al ejecutar dicha instrucción se mostrará una lista de los motores de almacenamiento soportados por MySQL, junto con una breve descripción de los mismos.

Existen varias maneras para especificar a MySQL qué motor de almacenamiento debe usar. Se puede cambiar el motor de almacenamiento predeterminado desde MySQL Administrator, o bien especificando dicho motor al crear la tabla. La instrucción quedaría así:

CREATE TABLE *nombre_tabla* (TIPOS_DE_DATOS) ENGINE = *motor*

Además, también se puede cambiar el tipo de una tabla con la instrucción ALTER TABLE, si más que añadir la cláusula ENGINE=*motor* al final de la instrucción. Hay que tener cuidado con esta última posibilidad, pues si se cambia el tipo de tabla hacia otro que no soporta los mismos índices, tipos de datos o tamaños, esto puede resultar en una pérdida de datos.

En la tabla 3.28 se pueden ver las características de los motores de almacenamiento soportados por MySQL.

Nombre	Instalación por defecto	Limitación de datos	Limitación de índices	Soporte para transacciones.	Nivel de bloqueo
MyISAM	Si	Ninguna	64 índices por tabla (32 en versiones anteriores a la 4.1.2). Máximo 16 columnas por índice.	No	Tabla
MERGE	Si	Las tablas subyacentes deben ser MyISAM.	No aplicable.	No	Tabla
MEMORY (HEAP)	Si	Tipos BLOB y TEXT no soportados.	Ninguna	No	Tabla
EXAMPLE	No	No aplicable.	No aplicable.	No	No aplicable.
FEDERATED	No	Limitados por la base de datos remota.	No aplicable.	No	No
ARCHIVE	No	Solo permite inserción de datos, no modificaciones.	No aplicable.	No	No aplicable.
CSV	No	Ninguna.	No soporta indxado.	No	Tabla.
BLACKHOLE	No	No se almacenan datos, sino que las sentencias se escriben a un registro binario (que después se distribuye a las bases de datos esclavas)	No aplicable.	No	No aplicable.
ISAM	No (actualmente en desuso).	4 GB tamaño máximo de la BD.	Máximo 16 índices por tabla.	No	Tabla
Berkeley DB (BDB)	Si	Ninguna.	Máximo 16 índices por tabla, 16 columnas por índice.	Si	Página
InnoDB	Si	Ninguna	Ninguna	Si	Fila

Tabla 3.28. *Características de los diferentes motores de almacenamiento en MySQL.*

A continuación se va a ver una breve descripción de cada tipo de almacenamiento.

MyISAM
Es el motor de almacenamiento por defecto en la mayoría de las instalaciones de MySQL. Deriva del motor de almacenamiento ISAM, que era el que se empleaba en las primeras versiones de MySQL. Este motor proporciona la mejor combinación entre rendimiento y funcionalidad, aunque carece de control transaccional y usa bloqueo a nivel de tabla.

MERGE
Este motor de almacenamiento permite combinar un número de terminado de tablas idénticas en una sola tabla. De esta manera, se pueden ejecutar consultas a múltiples tablas como si fueran una sola tabla. Cada tabla fusionada debe tener la misma definición, y sólo se pueden fusionar tablas del tipo MyISAM.

MEMORY (HEAP)
El tipo de tabla MEMORY (anteriormente conocido como HEAP) almacena todos los datos en la memoria. Una vez que se detenga el servidor MySQL, cualquier información almacenada en una base de datos MEMORY se perderá. No obstante, el formato de las tablas se mantiene, permitiendo la creación de tablas temporales que pueden utilizarse para almacenar información de rápido acceso sin tener que recrear las tablas cada vez que se inicia el servidor.

EXAMPLE
El motor EXAMPLE es un ejemplo de programación de motor de almacenamiento. Sirve como base para la creación de otros motores. No soporta inserción de datos y no es un motor práctico para acceso a base de datos, pero resulta una buena guía para desarrollar un motor de almacenamiento propio.

FEDERATED
Este motor se ha añadido en la última versión de MySQL (MySQL 5.0.3). Permite el acceso a datos desde una base de datos MySQL remota como si fuera una base de datos local. De hecho, el servidor MySQL actúa como un Proxy al servidor remoto, utilizando las librerías del cliente MySQL para conectar al host remoto.

ARCHIVE
Este motor soporta únicamente las instrucciones INSERT y SELECT, y no soporta la mayoría de los tipos de datos. La información almacenada en una tabla ARCHIVE es comprimida y no puede ser modificada. La información se almacena muy eficientemente al ser comprimida, pero por el contrario, para hacer una instrucción SELECT se debe leer toda la tabla, lo que conlleva la descompresión de la información. Esto hace que se incremente el tiempo empleado en realizar consultas complejas.

CSV
CSV almacena los datos no en un formato binario, sino en formato de archivo CSV (*Command Separated Values*). Esto conlleva limitaciones para los datos almacenados. No posee indexado, y no es un método eficiente para almacenar grandes volúmenes de datos. Sin embargo, el formato CSV resulta un motor bastante portable, en el sentido de

que resulta bastante fácil para diversos paquetes de software (Excel, OpenOffice, Access, etc.) importar los datos desde los archivos CSV.

BLACKHOLE

Por extraño que parezca, este motor no almacena ningún dato. Se pueden crear tablas e índices, pero todas las instrucciones que añaden o actualizan información de la base de datos se ejecutan sin escribir ningún dato. Esto se utiliza para probar estructuras de bases de datos y "jugar" con las definiciones de tabla sin crear ningún dato. Las instrucciones SQL en bases de datos BLACKHOLE se escriben en un registro.

ISAM

Este fue el motor de almacenamiento original disponible en las primeras versiones de MySQL. Actualmente esta en desuso debido a su gran cantidad de limitaciones.

Berkeley DB (BDB)

El motor de almacenamiento BDB es un motor bastante estable en cuanto a la integridad de los datos. Posee soporte para ejecutar operaciones COMMIT y ROLLBACK en transacciones. Esto hace que recuperar información sea increíblemente rápido. Sin embargo, está limitado en otros aspectos. Aunque usa bloqueo de página, bloquea sólo 8192 bytes de una tabla, en lugar de la tabla entera, lo que puede conllevar problemas en el caso de que se realicen un gran número de actualizaciones en la misma página. Además, a pesar de que posee gran velocidad para recuperar los datos, una pérdida en la clave principal o en los registros binarios podría hacer que los datos de la base de datos sean completamente irrecuperables.

InnoDB

Este motor de almacenamiento es proporcionado por Innobase Oy. Proporciona toda la funcionalidad del motor MyISAM, pero además provee de la capacidad de transacción y bloqueo a nivel de fila. En esta estructura, tanto los índices como los datos se almacenan en la caché y en disco, lo que proporciona una recuperación de datos rápida y eficiente incluso con grandes cantidades de datos. Son muchísimos los tipos de datos que se pueden almacenar con este motor, al igual que con MyISAM, pero con las ventajas que aporta el control transaccional y el bloqueo a nivel de fila.

3.13. REPLICACIÓN.

La replicación permite que un servidor de bases de datos sea duplicado en otro sistema. Esta característica está disponible en MySQL a partir de la versión 3.23.15. El funcionamiento es sencillo: un servidor actúa como maestro, mientras que uno o varios servidores actúan como esclavos. El servidor maestro escribe las actualizaciones en sus bases de datos en ficheros de registro binarios, y mantiene un índice de estos ficheros. Estos ficheros de registro sirven como una grabación de actualizaciones que se envían a los servidores esclavos. Cuando un servidor esclavo se conecta al servidor maestro, informa de su última posición dentro del listado de registros que contiene el servidor maestro desde que se propagó satisfactoriamente la última actualización. El servidor esclavo toma pues cualesquiera actualizaciones que hayan ocurrido desde entonces y se bloquea esperando más notificaciones de actualización del servidor maestro.
Los beneficios fundamentales de la replicación son los siguientes:

- Se incrementa la robustez de los sistemas, puesto que si se dan problemas en el servidor maestro, se puede cambiar a algún servidor esclavo como copia de seguridad.
- Proporciona mejores tiempos de respuesta en la ejecución de consultas, puesto que es posible dividir la carga de tráfico entre los servidores maestros y esclavos en consultas que no realizan ninguna modificación en las bases de datos (si se hiciera alguna modificación en un servidor esclavo, éste tendría que notificarlo al servidor maestro, por lo que no dejarían de estar sincronizados).
- Se pueden realizar copias de seguridad usando un servidor esclavo sin interferir en el servidor maestro.

Esta característica de replicación no está presente en otros sistemas gestores de bases de datos de similares características a MySQL.

3.14. NOVEDADES EN LA VERSIÓN 5.0.

Durante la redacción de este documento, MySQL ha lazado la versión 5.0 de MySQL. Esta versión incluye algunas características que se echaban en falta en versiones anteriores. Las nuevas características implementadas en la versión 5.0 son las siguientes.

- Tipo de dato BIT.
- Cursores. Proporciona soporte elemental para cursores en el servidor, que es una herramienta que permite recorrer el resultado de una consulta y hacer operaciones en cada paso de ésta.
- Directorio de datos. En esta versión se introduce una base de datos especial, llamada INFORMATION_SCHEMA. Esta base de datos proporciona los medios de acceso para acceder a los datos del servidor MySQL acerca de las bases de datos y objetos que contiene.
- Matemática de precisión. Se introducen criterios más estrictos para la aceptación o el rechazo de datos, y se implementa una nueva librería para aritmética en punto fijo. Esto contribuye a un mayor grado de precisión en operaciones matemáticas y a un mayor control sobre valores no válidos.
- Nuevos motores de almacenamiento. Se incluyen los motores de almacenamiento ARCHIVE y FEDERATED.
- Soporte para funciones y procedimientos almacenados.
- Seguimiento más estricto del estándar SQL.
- Soporte para disparadores.
- Mejora en el tipo de dato VARCHAR. A partir de la versión 5.0, el tamaño máximo de los tipos VARCHAR se incrementa a 65.532 bytes.
- Incorpora soporte para la creación, modificación y eliminación de vistas.