TUTORIAL DE MYSQL

MySQL es un gestor de base de datos rápido y sencillo de usar. También es uno de los motores de base de datos más usados en Internet, la principal razón de esto es que es gratis para aplicaciones no comerciales.

Las características principales de MySQL son:

Es un gestor de base de datos. Una base de datos es un conjunto de datos y un gestor de base de datos es una aplicación capaz de manejar este conjunto de datos de manera eficiente y cómoda.

- **Es una base de datos relacional**. Una base de datos relacional es un conjunto de datos que están almacenados en tablas entre las cuales se establecen unas relaciones para manejar los datos de una forma eficiente y segura. Para usar y gestionar una base de datos relacional se usa el lenguaje estándar de programación SQL.

- **Es Open Source**. El código fuente de MySQL se puede descargar y está accesible a cualquiera. Por otra parte, usa la licencia GPL para aplicaciones no comerciales.

- **Es una base de datos muy rápida**, segura y fácil de usar. Gracias a la colaboración de muchos usuarios, la base de datos se ha ido mejorando optimizándose en velocidad. Por eso es una de las bases de datos más usadas en Internet.

- **Existe una gran cantidad de software que la usa**.

El objetivo de este tutorial es mostrar el uso del programa cliente mysql para crear y usar una sencilla base de datos. MySQL (algunas veces referido como "monitor mysql") es un programa interactivo que permite conectarnos a un servidor MySQL, ejecutar algunas consultas, y ver los resultados. MySQL puede ser usado también en modo batch: es decir, se pueden colocar toda una serie de consultas en un archivo, y posteriormente decirle a MySQL que ejecute dichas consultas.

Este tutorial asume que MySQL está instalado en alguna máquina y que disponemos de un servidor MySQL al cual podemos conectarnos.

Para ver la lista de opciones proporcionadas por mysql, lo invocamos con la opción --help:

shell> mysql --help

También nos podemos conectar a mysql y después teclear help. A continuación, se describe el proceso completo de creación y uso de una base de datos en MySQL.

Para conectarse al servidor, usualmente necesitamos de un nombre de usuario (login) y de una contraseña (password), y si el servidor al que nos deseamos conectar está en una

máquina diferente de la nuestra, también necesitamos indicar el nombre o la dirección IP de dicho servidor. Una vez que conocemos estos tres valores, podemos conectarnos de la siguiente manera:

shell> mysql -h NombreDelServidor -u NombreDeUsuario -p

Cuando ejecutamos este comando, se nos pedirá que proporcionemos también la contraseña para el nombre de usuario que estamos usando.

Nosotros vamos a usar XAMPP para trabajar en clase con MySql, por defecto no tenemos password para el usuario root, pero podemos cambiarlo a posteriori. Si la conexión al servidor MySQL se pudo establecer de manera satisfactoria, recibiremos el mensaje de bienvenida y estaremos en el prompt de mysql:

```
shell>mysql -h casita -u blueman -p
Enter password: ******

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5563 to server version: 3.23.41

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

mysql>
```

Este prompt nos indica que mysql está listo para recibir comandos.

Algunas instalaciones permiten que los usuarios se conecten de manera anónima al servidor corriendo en la máquina local. Si es el caso de nuestra máquina, debemos de ser capaces de conectarnos al servidor invocando a mysql sin ninguna opción:

shell> mysql

Después de que nos hemos conectado de manera satisfactoria, podemos desconectarnos en cualquier momento al escribir "quit", "exit", o presionar CONTROL+D.

La mayoría de los ejemplos siguientes asume que estamos conectados al servidor, lo cual se indica con el prompt de mysql.

En este momento debemos de haber podido conectarnos ya al servidor MySQL, aun cuando no hemos seleccionado alguna base de datos para trabajar. Lo que haremos a continuación es escribir algunos comandos para irnos familiarizando con el funcionamiento de mysql.

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 3.23.41   | 2002-10-01   |
+-----------+--------------+
1 row in set (0.03 sec)

mysql>
```

Este comando ilustra distintas cosas acerca de mysql:

- Un comando normalmente consiste en una sentencia SQL seguida por un punto y coma.
- Cuando emitimos un comando, mysql lo manda al servidor para que lo ejecute, nos muestra los resultados y vuelve a poner el prompt indicando que está listo para recibir más consultas.
- mysql muestra los resultados de la consulta como una tabla (filas y columnas). La primera fila contiene etiquetas para las columnas. Las filas siguientes muestran los resultados de la consulta. Normalmente, las etiquetas de las columnas son los nombres de los campos de las tablas que estamos usando en alguna consulta. Si lo que estamos recuperando es el valor de una expresión (como en el ejemplo anterior) las etiquetas en las columnas son la expresión en sí.
- mysql muestra cuántas filas fueron devueltas y cuánto tiempo tardó en ejecutarse la consulta, lo cual puede darnos una idea de la eficiencia del servidor, aunque estos valores pueden ser un tanto imprecisos ya que no se muestra la hora de la CPU, y porque pueden verse afectados por otros factores, tales como la carga del servidor y la velocidad de comunicación en una red.
- Las palabras clave pueden ser escritas usando mayúsculas y minúsculas.

Las siguientes consultas son equivalentes:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Aquí está otra consulta que demuestra cómo se pueden escribir algunas expresiones matemáticas y trigonométricas:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+------------+---------+
|    0.707107 |      25 |
+------------+---------+
```

Aunque hasta este momento se han escrito sentencias sencillas de una sola línea, es posible escribir más de una sentencia por línea, siempre y cuando estén separadas por punto y coma:

```
mysql> SELECT VERSION(); SELECT NOW();
+-----------+
| VERSION() |
+-----------+
| 3.23.41   |
+-----------+
1 row in set (0.01 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2002-10-28 14:26:04 |
+---------------------+
```

```
1 row in set (0.01 sec)
```

Un comando no necesita ser escrito en una sola línea, así que los comandos que requieran de varias líneas no son un problema. mysql determinará en dónde finaliza la sentencia cuando encuentre el punto y coma, no cuando encuentre el fin de línea.

Aquí está un ejemplo que muestra una consulta simple escrita en varias líneas:

```
mysql> SELECT
    -> USER(),
    -> CURRENT_DATE;
+------------------+--------------+
| USER()           | CURRENT_DATE |
+------------------+--------------+
| blueman@localhost | 2002-09-14   |
+------------------+--------------+
1 row in set (0.00 sec)

mysql>
```

En este ejemplo debe notarse como cambia el prompt (de mysql> a ->) cuando se escribe una consulta en varias líneas. Esta es la manera en cómo mysql indica que está esperando a que finalice la consulta. Sin embargo si deseamos no terminar de escribir la consulta, podemos hacerlo al escribir \c como se muestra en el siguiente ejemplo:

```
mysql> SELECT
    -> USER(),
    -> \c
mysql>
```

De nuevo, volvemos al comando prompt mysql> que nos indica que mysql está listo para una nueva consulta.

Prompt	Significado
mysql>	Listo para una nueva consulta.
->	Esperando la línea siguiente de una consulta multi-línea.
'>	Esperando la siguiente línea para completar una cadena que comienza con una comilla sencilla (').
">	Esperando la siguiente línea para completar una cadena que comienza con una comilla doble (").

Los comandos multi-línea comúnmente ocurren por accidente cuando tecleamos ENTER, pero olvidamos escribir el punto y coma. En este caso mysql se queda esperando para que finalicemos la consulta:

```
mysql> SELECT USER()
    ->
```

Si esto llega a suceder, muy probablemente mysql estará esperando por un punto y coma, de manera que si escribimos el punto y coma podremos completar la consulta y mysql podrá ejecutarla:

```
mysql> SELECT USER()
    -> ;
+----------------+
| USER()         |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)

mysql>
```

Los prompts '> y "> ocurren durante la escritura de cadenas. En mysql podemos escribir cadenas utilizando comillas sencillas o comillas dobles (por ejemplo, 'hola' y "hola"), y mysql nos permite escribir cadenas que ocupen múltiples líneas. De manera que cuando veamos el prompt '> o "> , mysql nos indica que hemos empezado a escribir una cadena, pero no la hemos finalizado con la comilla correspondiente.

Aunque esto puede suceder si estamos escribiendo una cadena muy grande, es más frecuente que obtengamos alguno de estos prompts si inadvertidamente escribimos alguna de estas comillas.

Por ejemplo:

```
mysql> SELECT * FROM mi_tabla WHERE nombre = "Lupita AND edad < 30;
">
```

Si escribimos esta consulta SELECT y entonces presionamos ENTER para ver el resultado, no sucederá nada. En lugar de preocuparnos porque la consulta ha tomado mucho tiempo, debemos notar la pista que nos da mysql cambiando el prompt. Esto nos indica que mysql está esperando que finalicemos la cadena iniciada ("Lupita).

En este caso, ¿qué es lo que debemos hacer?. La cosa más simple es cancelar la consulta. Sin embargo, no basta con escribir \c, ya que mysql interpreta esto como parte de la cadena que estamos escribiendo. En lugar de esto, debemos escribir antes la comilla correspondiente y después \c :

```
mysql> SELECT * FROM mi_tabla WHERE nombre = "Lupita AND edad < 30;
"> " \c
mysql>
```

El prompt cambiará de nuevo al ya conocido mysql>, indicándonos que mysql está listo para una nueva consulta.

Es sumamente importante conocer lo que significan los prompts '> y ">, ya que si en algún momento nos aparece alguno de ellos, todas las líneas que escribamos a continuación serán consideradas como parte de la cadena, inclusive cuando escribimos QUIT. Esto puede ser confuso, especialmente si no sabemos que es necesario escribir la comilla correspondiente para finalizar la cadena, para que podamos escribir después algún otro comando, o terminar la consulta que deseamos ejecutar.

Ahora que conocemos como escribir y ejecutar sentencias, es tiempo de acceder a una base de datos.

Supongamos que tenemos diversas mascotas en casa (nuestro pequeño zoológico) y deseamos tener registros de los datos acerca de ellas. Podemos hacer esto al crear tablas que guarden esta información, para que posteriormente la consulta de estos datos sea bastante fácil y de manera muy práctica. Esta sección muestra cómo crear una base de datos, crear una tabla, incorporar datos en una tabla, y recuperar datos de las tablas de diversas maneras.

La base de datos "zoológico" será muy simple, pero no es difícil pensar en situaciones del mundo real en las cuales una base de datos similar puede ser usada.

Primeramente usaremos la sentencia SHOW para ver cuáles son las bases de datos existentes en el servidor al que estamos conectados:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)

mysql>
```

Es probable que la lista de bases de datos que veamos sea diferente en nuestro caso, pero seguramente las bases de datos "mysql" y "test" estarán entre ellas. En particular, la base de datos "mysql" es requerida, ya que ésta tiene la información de los privilegios de los usuarios de MySQL. La base de datos "test" es creada durante la instalación de MySQL con el propósito de servir como área de trabajo para los usuarios que inician en el aprendizaje de MySQL.

Es posible que no veamos todas las bases de datos si no tenemos el privilegio SHOW DATABASES. Se recomienda revisar la sección del manual de MySQL dedicada a los comandos GRANT y REVOKE.

Si la base de datos "test" existe, hay que intentar acceder a ella:

```
mysql> USE test
Database changed
mysql>
```

Observar que USE, al igual que QUIT, no requiere el uso del punto y coma, aunque si se usa, no hay ningún problema. El comando USE es especial también de otra manera: éste debe ser usado en una sola línea.

Podríamos usar la base de datos "test" (si tenemos acceso a ella) para los ejemplos que vienen a continuación, pero cualquier cosa que hagamos puede ser eliminada por cualquier otro usuario que tenga acceso a esta base de datos. Por esta razón, es recomendable que preguntemos al administrador MySQL acerca de la base de datos que

podemos usar. Supongamos que deseamos tener una base de datos llamada "zoologico" (nótese que no se está acentuando la palabra) a la cual sólo nosotros tengamos acceso, para ello el administrador necesita ejecutar un comando como el siguiente:

```
mysql> GRANT ALL on zoologico.* TO MiNombreUsuario@MiComputadora
    -> IDENTIFIED BY 'MiContraseña';
```

En donde MiNombreUsuario es el nombre de usuario asignado dentro del contexto de MySQL, MiComputadora es el nombre o la dirección IP de la computadora desde la que nos conectamos al servidor MySQL, y MiContraseña es la contraseña que se nos ha asignado, igualmente, dentro del ambiente de MySQL exclusivamente. Ambos, nombre de usuario y contraseña no tienen nada que ver con el nombre de usuario y contraseña manejados por el sistema operativo (si es el caso).

Una vez que estamos en la base de datos, si no nos acordamos de qué base de datos es:

```
select database();
```

Si el administrador creó la base de datos en el momento de asignar los permisos, podemos hacer uso de ella. De otro modo, nosotros debemos crearla:

```
mysql> USE zoologico
ERROR 1049: Unknown database 'zoologico'
mysql>
```

El mensaje anterior indica que la base de datos no ha sido creada, por lo tanto necesitamos crearla.

```
mysql> CREATE DATABASE zoologico;
Query OK, 1 row affected (0.00 sec)

mysql> USE zoologico
Database changed
mysql>
```

Bajo el sistema operativo Unix, los nombres de las bases de datos son sensibles al uso de mayúsculas y minúsculas (no como las palabras clave de SQL), por lo tanto debemos de tener cuidado de escribir correctamente el nombre de la base de datos. Esto es cierto también para los nombres de las tablas.

Al crear una base de datos no se selecciona ésta de manera automática; debemos hacerlo de manera explícita, por ello usamos el comando USE en el ejemplo anterior.

La base de datos se crea sólo una vez, pero nosotros debemos seleccionarla cada vez que iniciamos una sesión con MySQL. Por ello es recomendable que se indique la base de datos sobre la que vamos a trabajar al momento de invocar al monitor de MySQL. Por ejemplo:

```
shell>mysql -h casita -u blueman -p zoologico

Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 17 to server version: 3.23.38-nt

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

mysql>
```

Observar que "zoologico" no es la contraseña que se está proporcionando desde la línea de comandos, sino el nombre de la base de datos a la que deseamos acceder. Si deseamos proporcionar la contraseña en la línea de comandos después de la opción "-p", debemos de hacerlo sin dejar espacios (por ejemplo, -phola123, no como -p hola123). Sin embargo, escribir nuestra contraseña desde la línea de comandos no es recomendado, ya que es bastante inseguro.

Crear la base de datos es la parte más fácil, pero en este momento la base de datos está vacía, como lo indica el comando SHOW TABLES:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

La parte un tanto complicada es decidir la estructura que debe tener nuestra base de datos: qué tablas se necesitan y qué columnas estarán en cada tabla.

En principio, necesitamos una tabla que contenga un registro para cada una de nuestras mascotas. Ésta puede ser una tabla llamada mascotas, y debe contener por lo menos el nombre de cada uno de nuestros animales. Ya que el nombre en sí no es muy interesante, la tabla debe contener alguna otra información. Por ejemplo, si más de una persona en nuestra familia tiene una mascota, es probable que tengamos que guardar la información acerca de quién es el dueño de cada mascota. Así mismo, también sería interesante contar con alguna información más descriptiva tal como la especie, y el sexo de cada mascota.

¿Y qué sucede con la edad? Esto puede ser también de interés, pero no es una buena idea almacenar este dato en la base de datos. La edad cambia conforme pasa el tiempo, lo cual significa que debemos de actualizar los registros frecuentemente. En vez de esto, es una mejor idea guardar un valor fijo, tal como la fecha de nacimiento. Entonces, cuando necesitemos la edad, la podemos calcular como la diferencia entre la fecha actual y la fecha de nacimiento. MySQL proporciona funciones para hacer operaciones entre fechas, así que no hay ningún problema.

Al almacenar la fecha de nacimiento en lugar de la edad tenemos algunas otras ventajas:

Podemos usar la base de datos para tareas tales como generar recordatorios para cada cumpleaños próximo de nuestras mascotas. Podemos calcular la edad en relación a otras fechas que la fecha actual. Por ejemplo, si almacenamos la fecha en que murió nuestra mascota en la base de datos, es fácil calcular que edad tenía nuestro animal cuando falleció. Es probable que estemos pensando en otro tipo de información que sería igualmente útil en la tabla "mascotas", pero para nosotros será suficiente por ahora contar con información de nombre, propietario, especie, nacimiento y fallecimiento.

Usaremos la sentencia CREATE TABLE para indicar como estarán conformados los registros de nuestras mascotas.

```
mysql> CREATE TABLE mascotas(
    -> nombre VARCHAR(20), propietario VARCHAR(20),
    -> especie VARCHAR(20), sexo CHAR(1), nacimiento DATE,
    -> fallecimiento DATE);
Query OK, 0 rows affected (0.02 sec)

mysql>
```

VARCHAR es una buena elección para los campos nombre, propietario, y especie, ya que los valores que almacenarán son de longitud variable. No es necesario que la longitud de estas columnas sea la misma, ni tampoco que sea de 20. Se puede especificar cualquier longitud entre 1 y 255, lo que se considere más adecuado. Si resulta que la elección de la longitud de los campos que hemos hecho no resultó adecuada, MySQL proporciona una sentencia ALTER TABLE que nos puede ayudar a solventar este problema.

El campo sexo puede ser representado en una variedad de formas, por ejemplo, "m" y "f", o tal vez "masculino" y "femenino", aunque resulta más simple la primera opción.

El uso del tipo de dato DATE para los campos nacimiento y fallecimiento resulta obvio.

Ahora que hemos creado la tabla, la sentencia SHOW TABLES debe producir algo como:

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_zoologico |
+---------------------+
| mascotas            |
+---------------------+
1 row in set (0.00 sec)

mysql>
```

Para verificar que la tabla fue creada como nosotros esperábamos, usaremos la sentencia DESCRIBE:

```
mysql> DESCRIBE mascotas;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| nombre       | varchar(20) | YES  |     | NULL    |       |
| propietario  | varchar(20) | YES  |     | NULL    |       |
| especie      | varchar(20) | YES  |     | NULL    |       |
| sexo         | char(1)     | YES  |     | NULL    |       |
| nacimiento   | date        | YES  |     | NULL    |       |
| fallecimento | date        | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql>
```

Podemos hacer uso de la sentencia DESCRIBE en cualquier momento, por ejemplo, si olvidamos los nombres o el tipo de las columnas en la tabla.

Después de haber creado la tabla, ahora podemos incorporar algunos datos en ella, para lo cual haremos uso de las sentencias INSERT y LOAD DATA.

Supongamos que los registros de nuestras mascotas pueden ser descritos por los datos mostrados en la siguiente tabla.

Nombre	Propietario	Especie	Sexo	Nacimiento	Fallecimento
Fluffy	Arnoldo	Gato	f	1999-02-04	
Mau	Juan	Gato	m	1998-03-17	
Buffy	Arnoldo	Perro	f	1999-05-13	
FanFan	Benito	Perro	m	2000-08-27	
Kaiser	Diana	Perro	m	1998-08-31	1997-07-29
Chispa	Omar	Ave	f	1998-09-11	
Wicho	Tomás	Ave		2000-02-09	
Skim	Benito	Serpiente	m	2001-04-29	

Debemos observar que MySQL espera recibir fechas en el formato YYYY-MM-DD, que puede ser diferente a lo que nosotros estamos acostumbrados.

Ya que estamos iniciando con una tabla vacía, la manera más fácil de poblarla es crear un archivo de texto que contenga un registro por línea para cada uno de nuestros animalitos para que posteriormente carguemos el contenido del archivo en la tabla únicamente con una sentencia (¡Ojo! Eso no es SQL).

Por tanto, debemos de crear un archivo de texto "mascotas.txt" que contenga un registro por línea con valores separados por tabuladores, cuidando que el orden de las columnas sea el mismo que utilizamos en la sentencia CREATE TABLE. Para valores que no conozcamos podemos usar valores nulos (NULL). Para representar estos valores en nuestro archivo debemos usar \N.

Usaremos el archivo mascotas.txt.

Para cargar el contenido del archivo en la tabla mascotas, usaremos el siguiente comando:

```
mysql> LOAD DATA LOCAL INFILE "/mascotas.txt" INTO TABLE mascotas
LINES TERMINATED BY '\r\n';
o
mysql> LOAD DATA LOCAL INFILE "\\mascotas.txt" INTO TABLE mascotas
LINES TERMINATED BY '\r\n';
```

La sentencia LOAD DATA (de Mysql) nos permite especificar cuál es el separador de columnas, y el separador de registros, por defecto el tabulador es el separador de columnas (campos), y el salto de línea es el separador de registros, que en este caso son

suficientes para que la sentencia LOAD DATA lea correctamente el archivo "mascotas.txt".

Tendremos que indicar el directorio donde tengamos el archivo de texto. Por defecto cogerá el path donde se almacena el esquema de la base de datos:

C:\xampp\mysql\data\zoologico

La sentencia quedaría:

LOAD DATA LOCAL INFILE "C:/xampp/mysql/data/zoologico/mascotas.txt" INTO TABLE mascotas LINES TERMINATED BY '\r\n';

Si lo que deseamos es añadir registro a registro, entonces debemos hacer uso de la sentencia INSERT (de SQL). En la manera más simple, debemos proporcionar un valor para cada columna en el orden en el cual fueron listados en la sentencia CREATE TABLE. Supongamos que Diana compra un nuevo hámster llamado Pelusa. Podemos usar la sentencia INSERT para agregar su registro en nuestra base de datos.

```
mysql> INSERT INTO mascotas
    -> VALUES('Pelusa','Diana','Hamster','f','2000-03-30',NULL);
```

Los valores de cadenas y fechas deben estar encerrados entre comillas. También, con la sentencia INSERT podemos insertar el valor NULL directamente para representar un valor nulo, un valor que no conocemos. En este caso no se usa \N como en el caso de la sentencia LOAD DATA.

La sentencia SELECT de SQL es usada para obtener la información guardada en una tabla. La forma general de esta sentencia es:

```
SELECT LaInformaciónQueDeseamos FROM DeQueTabla WHERE
CondiciónASatisfacer
```

Aquí, LaInformaciónQueDeseamos es la información que queremos ver. Esta puede ser una lista de columnas, o un * para indicar "todas las columnas". DeQueTabla indica el nombre de la tabla de la cual vamos a obtener los datos. La claúsula **WHERE** es opcional. Si está presente, la CondiciónASatisfacer especifica las condiciones que los registros deben satisfacer para que puedan ser mostrados.

Seleccionando todos los datos

La manera más simple de la sentencia **SELECT** es cuando se recuperan todos los datos de una tabla:

```
mysql> SELECT * FROM mascotas;
+--------+------------+-----------+------+------------+-------------
-+
| nombre | propietario | especie   | sexo | nacimiento | fallecimiento
|
+--------+------------+-----------+------+------------+-------------
-+
```

```
| Fluffy | Arnoldo     | Gato      | f    | 1999-02-04 | NULL
|
| Mau    | Juan        | Gato      | m    | 1998-03-17 | NULL
|
| Buffy  | Arnoldo     | Perro     | f    | 1999-05-13 | NULL
|
| FanFan | Benito      | Perro     | m    | 2000-08-27 | NULL
|
| Kaiser | Diana       | Perro     | m    | 1998-08-31 | 1997-07-29
|
| Chispa | Omar        | Ave       | f    | 1998-09-11 | NULL
|
| Wicho  | Tomás       | Ave       | NULL | 2000-02-09 | NULL
|
| Skim   | Benito      | Serpiente | m    | 2001-04-29 | NULL
|
| Pelusa | Diana       | Hamster   | f    | 2000-03-30 | NULL
|
+--------+-------------+-----------+------+------------+-------------
-+
9 rows in set (0.00 sec)
```

Esta forma del **SELECT** es útil si deseamos ver los datos completos de la tabla, por ejemplo, para asegurarnos de que están todos los registros después de la carga de un archivo.

Por ejemplo, en este caso que estamos tratando, al consultar los registros de la tabla, nos damos cuenta de que hay un error en el archivo de datos (mascotas.txt): parece que Kaiser ha nacido después de que ha fallecido. Al revisar un poco los datos de Kaiser encontramos que la fecha correcta de nacimiento es el año 1989, no 1998.

Hay por lo menos un par de maneras de solucionar este problema:

Editar el archivo "mascotas.txt" para corregir el error, eliminar los datos de la tabla mascotas con la sentencia **DELETE**, y cargar los datos nuevamente con el comando **LOAD DATA**:

```
mysql> DELETE FROM mascotas;
mysql> LOAD DATA LOCAL INFILE "mascotas.txt" INTO TABLE mascotas;
```

Sin embargo, si hacemos esto, debemos introducir a mano los datos de Pelusa, la mascota de Diana.

La segunda opción consiste en corregir sólo el registro erróneo con una sentencia **UPDATE**:

```
mysql> UPDATE mascotas SET nacimiento="1989-08-31"
WHERE nombre="Kaiser";
```

Como se mostró anteriormente, es muy fácil recuperar los datos de una tabla completa. Pero típicamente no deseamos hacer esto, particularmente cuando las tablas son demasiado grandes. En vez de ello, estaremos más interesados en responder preguntas particulares, en cuyo caso debemos especificar algunas restricciones para la información que deseamos ver.

Podemos seleccionar solo registros particulares de una tabla. Por ejemplo, si deseamos verificar el cambio que hicimos a la fecha de nacimiento de Kaiser, seleccionamos solo el registro de Kaiser de la siguiente manera:

```
mysql> SELECT * FROM mascotas WHERE nombre="Kaiser";
+--------+-------------+---------+------+------------+-------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimento |
+--------+-------------+---------+------+------------+-------------+
| Kaiser | Diana       | Perro   | m    | 1989-08-31 | 1997-07-29   |
+--------+-------------+---------+------+------------+-------------+
1 row in set (0.00 sec)
```

La salida mostrada confirma que el año ha sido corregido de 1998 a 1989.

La comparación de cadenas en MySQL es normalmente no sensitiva, así que podemos especificar el nombre como "kaiser", "KAISER", etc. El resultado de la consulta será el mismo.

Podemos además especificar condiciones sobre cualquier columna, no sólo el "nombre". Por ejemplo, si deseamos conocer qué mascotas nacieron después del 2000, tendríamos que usar la columna "nacimiento":

```
mysql> SELECT * FROM mascotas WHERE nacimiento >= "2000-1-1";
+--------+-------------+-----------+------+------------+-------------
-+
| nombre | propietario | especie   | sexo | nacimiento | fallecimiento
|
+--------+-------------+-----------+------+------------+-------------
-+
| FanFan | Benito      | Perro     | m    | 2000-08-27 | NULL
|
| Wicho  | Tomás       | Ave       | NULL | 2000-02-09 | NULL
|
| Skim   | Benito      | Serpiente | m    | 2001-04-29 | NULL
|
| Pelusa | Diana       | Hamster   | f    | 2000-03-30 | NULL
|
+--------+-------------+-----------+------+------------+-------------
-+
4 rows in set (0.00 sec)
```

Hay que tener cuidado en especificar la fecha entre dobles comillas. Podemos también combinar condiciones, por ejemplo, para localizar a los perros hembras:

```
mysql> SELECT * FROM mascotas WHERE especie="Perro" AND sexo="f";
+--------+-------------+---------+------+------------+--------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimiento |
+--------+-------------+---------+------+------------+--------------+
| Buffy  | Arnoldo     | Perro   | f    | 1999-05-13 | NULL          |
+--------+-------------+---------+------+------------+--------------+
1 row in set (0.00 sec)
```

La consulta anterior usa el operador lógico **AND**. Hay también un operador lógico **OR**:

```
mysql> SELECT * FROM mascotas WHERE especie = "Ave" OR especie =
"Gato";
```

```
+--------+-------------+---------+------+------------+---------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimiento |
+--------+-------------+---------+------+------------+---------------+
| Fluffy | Arnoldo     | Gato    | f    | 1999-02-04 | NULL          |
| Mau    | Juan        | Gato    | m    | 1998-03-17 | NULL          |
| Chispa | Omar        | Ave     | f    | 1998-09-11 | NULL          |
| Wicho  | Tomás       | Ave     | NULL | 2000-02-09 | NULL          |
+--------+-------------+---------+------+------------+---------------+
4 rows in set (0.00 sec)
```

El operador **AND** y el operador **OR** pueden ser intercambiados. Si hacemos esto, es buena idea usar paréntesis para indicar como deben ser agrupadas las condiciones:

```
mysql> SELECT * FROM mascotas WHERE (especie = "Gato" AND sexo = "m")
    -> OR (especie = "Perro" AND sexo = "f");
+--------+-------------+---------+------+------------+---------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimiento |
+--------+-------------+---------+------+------------+---------------+
| Mau    | Juan        | Gato    | m    | 1998-03-17 | NULL          |
| Buffy  | Arnoldo     | Perro   | f    | 1999-05-13 | NULL          |
+--------+-------------+---------+------+------------+---------------+
2 rows in set (0.00 sec)
```

Si no deseamos ver los registros completos de una tabla, entonces tenemos que usar los nombres de las columnas en las que estamos interesados separándolas por coma. Por ejemplo, si deseamos conocer la fecha de nacimiento de nuestras mascotas, debemos seleccionar la columna "nombre" y "nacimiento":

```
mysql> SELECT nombre, nacimiento FROM mascotas;
+--------+------------+
| nombre | nacimiento |
+--------+------------+
| Fluffy | 1999-02-04 |
| Mau    | 1998-03-17 |
| Buffy  | 1999-05-13 |
| FanFan | 2000-08-27 |
| Kaiser | 1989-08-31 |
| Chispa | 1998-09-11 |
| Wicho  | 2000-02-09 |
| Skim   | 2001-04-29 |
| Pelusa | 2000-03-30 |
+--------+------------+
9 rows in set (0.00 sec)
```

Para conocer quién tiene alguna mascota, usaremos la siguiente consulta:

```
mysql> SELECT propietario FROM mascotas;
+-------------+
| propietario |
+-------------+
| Arnoldo     |
| Juan        |
| Arnoldo     |
| Benito      |
| Diana       |
| Omar        |
| Tomás       |
| Benito      |
```

```
| Diana       |
+-------------+
9 rows in set (0.00 sec)
```

Sin embargo, debemos notar que la consulta recupera el nombre del propietario de cada mascota, y algunos de ellos aparecen más de una vez. Para minimizar la salida, agregaremos la palabra clave **DISTINCT**:

```
mysql> SELECT DISTINCT propietario FROM mascotas;
+-------------+
| propietario |
+-------------+
| Arnoldo     |
| Juan        |
| Benito      |
| Diana       |
| Omar        |
| Tomás       |
+-------------+
6 rows in set (0.03 sec)
```

Se puede usar también una cláusula **WHERE** para combinar selección de filas con selección de columnas. Por ejemplo, para obtener la fecha de nacimiento de los perritos y los gatitos, usaremos la siguiente consulta:

```
mysql> SELECT nombre, especie, nacimiento FROM mascotas
    -> WHERE especie = "perro" OR especie = "gato";
+--------+---------+------------+
| nombre | especie | nacimiento |
+--------+---------+------------+
| Fluffy | Gato    | 1999-02-04 |
| Mau    | Gato    | 1998-03-17 |
| Buffy  | Perro   | 1999-05-13 |
| FanFan | Perro   | 2000-08-27 |
| Kaiser | Perro   | 1989-08-31 |
+--------+---------+------------+
5 rows in set (0.00 sec)
```

Se debe notar en los ejemplos anteriores que las filas devueltas son mostradas sin ningún orden en particular. Sin embargo, frecuentemente es más fácil examinar la salida de una consulta cuando las filas son ordenadas en alguna forma útil. Para ordenar los resultados, tenemos que usar una cláusula **ORDER BY**.

Aquí aparecen algunos datos ordenados por fecha de nacimiento:

```
mysql> SELECT nombre, nacimiento FROM mascotas ORDER BY nacimiento;
+--------+------------+
| nombre | nacimiento |
+--------+------------+
| Kaiser | 1989-08-31 |
| Mau    | 1998-03-17 |
| Chispa | 1998-09-11 |
| Fluffy | 1999-02-04 |
| Buffy  | 1999-05-13 |
| Wicho  | 2000-02-09 |
| Pelusa | 2000-03-30 |
| FanFan | 2000-08-27 |
```

```
| Skim   | 2001-04-29 |
+--------+------------+
9 rows in set (0.00 sec)
```

En las columnas de tipo carácter, la ordenación es ejecutada normalmente de forma no sensitiva, es decir, no hay diferencia entre mayúsculas y minúsculas. Sin embargo, se puede forzar un ordenamiento sensitivo al usar el operador **BINARY**.

```
INSERT INTO mascotas
    -> VALUES('pelusa','Diana','Hamster','f','2000-03-30',NULL);

SELECT nombre, nacimiento FROM mascotas ORDER BY binary(nombre);

SELECT nombre, nacimiento FROM mascotas ORDER BY nombre;
```

Para ordenar en orden inverso, debemos agregar la palabra clave **DESC** al nombre de la columna que estamos usando en el ordenamiento:

```
mysql> SELECT nombre, nacimiento FROM mascotas ORDER BY
    -> nacimiento DESC;
+--------+------------+
| nombre | nacimiento |
+--------+------------+
| Skim   | 2001-04-29 |
| FanFan | 2000-08-27 |
| Pelusa | 2000-03-30 |
| Wicho  | 2000-02-09 |
| Buffy  | 1999-05-13 |
| Fluffy | 1999-02-04 |
| Chispa | 1998-09-11 |
| Mau    | 1998-03-17 |
| Kaiser | 1989-08-31 |
+--------+------------+
9 rows in set (0.00 sec)
```

Podemos ordenar múltiples columnas. Por ejemplo, para ordenar por tipo de animal, y poner al inicio los animales más pequeños de edad, usaremos la siguiente consulta:

```
mysql> SELECT nombre, especie, nacimiento FROM mascotas
    -> ORDER BY especie, nacimiento DESC;
+--------+-----------+------------+
| nombre | especie   | nacimiento |
+--------+-----------+------------+
| Wicho  | Ave       | 2000-02-09 |
| Chispa | Ave       | 1998-09-11 |
| Fluffy | Gato      | 1999-02-04 |
| Mau    | Gato      | 1998-03-17 |
| Pelusa | Hamster   | 2000-03-30 |
| FanFan | Perro     | 2000-08-27 |
| Buffy  | Perro     | 1999-05-13 |
| Kaiser | Perro     | 1989-08-31 |
| Skim   | Serpiente | 2001-04-29 |
+--------+-----------+------------+
9 rows in set (0.00 sec)
```

Notar que la palabra clave **DESC** aplica sólo a la columna nombrada que le precede.