# MySql Basico

## Administración de bases de Datos

**Por**

**Edward Haidygguert González Rodríguez**
**Código: 71729158**

**UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA - UNAD**
**ESCUELA DE CIENCIAS BÁSICAS TECNOLOGÍA E INGENIERÍA**
**Medellín**

# Tutorial básico de MySQL

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 cuál podemos conectarnos. Si este no es el caso, tenemos que contactar con nuestro administrador MySQL. (Si nosotros somos los administradores, es necesario consultar la documentación de MySQL que se refiere a la instalación y configuración del servidor MySQL).

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

```
shell> mysql --help
```

A continuación se describe el proceso completo de creación y uso de una base de datos en MySQL. Si se está interesado sólo en el acceso y uso de una base de datos existente, se pueden omitir las secciones que describen como crear la base de datos y las tablas correspondientes.

Puesto que es imposible que se describan a detalle muchos de los tópicos cubiertos en este artículo, se recomienda que se consulte el manual de MySQL para obtener más información al respecto.

## *Conectándose y desconectándose al servidor 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.
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**.

*Ejecutando algunas consultas*

En este momento debimos de haber podido conectarnos ya al servidor MySQL, aún 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 de un 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 regresa 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 regresadas y cuanto 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 del 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 como 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 sóla 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 sóla 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 un 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, se nos regresa el comando el prompt mysql> que nos indica que mysql está listo para una nueva consulta.

En la siguiente tabla se muestran cada uno de los prompts que podemos obtener y una breve descripción de su significado para mysql:

| 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 la 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.

### *Creando y usando una base de datos*

Ahora que conocemos como escribir y ejecutar sentencias, es tiempo de accesar 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 como 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 (deliveradamente), pero no es difícil pensar de

situaciones del mundo real en la cual 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.

Se debe anotar también que 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 accesar a ella:

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

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

Si el administrador creó la base de datos al 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 autómatica; 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.

*Creando una tabla*

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 animalitos. 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 quien 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 que 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 animalito 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,
    -> fallecimento 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 fallecimento debe de resultar 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 fué creada como nosotros esperabamos, 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 ejempo, si olvidamos los nombres ó el tipo de las columnas en la tabla.

### *Cargando datos en una 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 | Fallecimiento |
|---|---|---|---|---|---|
| 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.

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.

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;
```

La sentencia LOAD DATA nos permite especificar cuál es el separador de columnas, y el separador de registros, por default 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".

Si lo que deseamos es añadir un registro a la vez, entonces debemos hacer uso de la sentencia INSERT. 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 nuestra hermana Diana

compra un nuevo hamster nombrado 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);
```

Notar que 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.

De este ejemplo, debemos ser capaces de ver que es un poco más la tarea que se tiene que realizar si inicialmente cargamos los registros con varias sentencias INSERT en lugar de una única sentencia LOAD DATA.

*Recuperando información de una tabla*

La sentencia SELECT 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 el pedigree 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 ingresar los datos de Pelusa, la mascota de nuestra hermana 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.

### *- Seleccionando registros particulares*

Podemos seleccionar sólo registros particulares de una tabla. Por ejemplo, si deseamos verificar el cambio que hicimos a la fecha de nacimiento de *Kaiser*, seleccionamos sólo 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 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)
```

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)
```

***- Seleccionando columnas particulares***

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 claúsula 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)
```

*Ordenando registros*

Se debe notar en los ejemplos anteriores que las filas regresadas 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 caracter, el ordenamiento es ejecutado 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.

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 animalitos 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.

**Tutorial básico de MySQL – II**

El objetivo de este tutorial es mostrar el uso del programa cliente **mysql** para crear y usar una base de datos

## *Acerca de este documento*

Esta es la parte complementaria del artículo publicado con anterioridad titulado Tutorial básico de MySQL. Seguramente se tendrá que leer primero dicho artículo para entender lo que se explica en este documento.

### *Cálculos con fechas*

MySQL proporciona diversas funciones que se pueden usar para efectuar cálculos sobre fechas, por ejemplo, para calcular edades o extraer partes de una fecha (día, mes, año, etc).

Para determinar la edad de cada una de nuestras mascotas, tenemos que calcular la diferencia de años de la fecha actual y la fecha de nacimiento, y entonces substraer uno si la fecha actual ocurre antes en el calendario que la fecha de nacimiento. Las siguientes consultas muestran la fecha actual, la fecha de nacimiento y la edad para cada mascota.

```
mysql> SELECT nombre, nacimiento, CURRENT_DATE,
    -> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
    -> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5)) AS edad FROM mascotas;
+--------+------------+--------------+------+
| nombre | nacimiento | CURRENT_DATE | edad |
+--------+------------+--------------+------+
| Fluffy | 1999-02-04 | 2002-12-23   |    3 |
| Mau    | 1998-03-17 | 2002-12-23   |    4 |
| Buffy  | 1999-05-13 | 2002-12-23   |    3 |
| FanFan | 2000-08-27 | 2002-12-23   |    2 |
| Kaiser | 1989-08-31 | 2002-12-23   |   13 |
| Chispa | 1998-09-11 | 2002-12-23   |    4 |
| Wicho  | 2000-02-09 | 2002-12-23   |    2 |
| Skim   | 2001-04-29 | 2002-12-23   |    1 |
| Pelusa | 2000-03-30 | 2002-12-23   |    2 |
+--------+------------+--------------+------+
9 rows in set (0.01 sec)
```

Aquí, YEAR( ) obtiene únicamente el año y RIGHT( ) obtiene los cinco caracteres más a la derecha de cada una de las fechas, que representan el mes y el día (MM-DD). La parte de la expresión que compara los valores MM-DD se evalúa a 1 o 0, y permite ajustar el valor de la edad en el caso de que el valor MM-DD de la fecha actual ocurra antes del valor MM-DD de la fecha de nacimiento.

Dado que la expresión en sí es bastante fea, se ha usado un alias (edad) que es el que aparece como etiqueta en la columna que muestra el resultado de la consulta.

Esta consulta debe trabajar bien, pero el resultado puede ser de alguna manera más útil si las filas son presentadas en algún orden. Para ello haremos uso de la cláusula ORDER BY.

Por ejemplo, para ordenar por nombre, usaremos la siguiente consulta:

```
mysql> SELECT nombre, nacimiento, CURRENT_DATE,
    -> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
    -> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5))
    -> AS edad FROM mascotas ORDER BY nombre;
+--------+------------+--------------+------+
| nombre | nacimiento | CURRENT_DATE | edad |
+--------+------------+--------------+------+
| Buffy  | 1999-05-13 | 2002-12-23   |    3 |
| Chispa | 1998-09-11 | 2002-12-23   |    4 |
| FanFan | 2000-08-27 | 2002-12-23   |    2 |
| Fluffy | 1999-02-04 | 2002-12-23   |    3 |
| Kaiser | 1989-08-31 | 2002-12-23   |   13 |
| Mau    | 1998-03-17 | 2002-12-23   |    4 |
| Pelusa | 2000-03-30 | 2002-12-23   |    2 |
| Skim   | 2001-04-29 | 2002-12-23   |    1 |
| Wicho  | 2000-02-09 | 2002-12-23   |    2 |
+--------+------------+--------------+------+
9 rows in set (0.00 sec)
```

Para ordenar por edad en lugar de nombre, únicamente tenemos que usar una cláusula ORDER BY diferente:

```
mysql> SELECT nombre, nacimiento, CURRENT_DATE,
    -> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
    -> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5))
    -> AS edad FROM mascotas ORDER BY edad;
+--------+------------+--------------+------+
| nombre | nacimiento | CURRENT_DATE | edad |
+--------+------------+--------------+------+
| Skim   | 2001-04-29 | 2002-12-23   |    1 |
| FanFan | 2000-08-27 | 2002-12-23   |    2 |
| Wicho  | 2000-02-09 | 2002-12-23   |    2 |
| Pelusa | 2000-03-30 | 2002-12-23   |    2 |
| Fluffy | 1999-02-04 | 2002-12-23   |    3 |
| Buffy  | 1999-05-13 | 2002-12-23   |    3 |
| Mau    | 1998-03-17 | 2002-12-23   |    4 |
| Chispa | 1998-09-11 | 2002-12-23   |    4 |
| Kaiser | 1989-08-31 | 2002-12-23   |   13 |
+--------+------------+--------------+------+
9 rows in set (0.01 sec)
```

Una consulta similar puede ser usada para determinar la edad que tenía una mascota cuando falleció. Para determinar que animalitos ya fallecieron, la condición es que el valor en el campo fallecimiento no sea nulo (NULL). Entonces, para los registros con valor no-nulo, calculamos la diferencia entre los valores fallecimiento y nacimiento.

```
mysql> SELECT nombre, nacimiento, fallecimiento,
    -> (YEAR(fallecimiento) - YEAR(nacimiento))
    -> - (RIGHT(fallecimiento,5) < RIGHT(nacimiento,5))
    -> AS edad FROM mascotas WHERE fallecimiento IS NOT NULL;
+--------+------------+--------------+------+
| nombre | nacimiento | fallecimiento | edad |
+--------+------------+--------------+------+
| Kaiser | 1989-08-31 | 1997-07-29   |    7 |
+--------+------------+--------------+------+
1 row in set (0.01 sec)
```

La consulta usa fallecimiento IS NOT NULL, en vez de fallecimiento < > NULL porque NULL es una valor especial. Esto será explicando más a detalle posteriormente.

¿Qué sucede si deseamos conocer cuáles de nuestras mascotas cumplen años el próximo mes? Para este tipo de cálculos, el año y el día son irrelevantes; simplemente tenemos que extraer el valor del mes en la columna nacimiento. Como se mencionó anteriormente, MySQL proporciona diversas funciones para trabajar y manipular fechas, en este caso haremos uso de la función MONTH( ). Para ver como trabaja, vamos a ejecutar una consulta muy simple que muestra tanto el valor de una fecha como el valor que regresa la función MONTH( ).

```
mysql> SELECT nombre, nacimiento, MONTH(nacimiento) FROM mascotas;
+--------+------------+--------------------+
| nombre | nacimiento | MONTH(nacimiento)  |
+--------+------------+--------------------+
| Fluffy | 1999-02-04 |                  2 |
| Mau    | 1998-03-17 |                  3 |
| Buffy  | 1999-05-13 |                  5 |
| FanFan | 2000-08-27 |                  8 |
| Kaiser | 1989-08-31 |                  8 |
| Chispa | 1998-09-11 |                  9 |
| Wicho  | 2000-02-09 |                  2 |
| Skim   | 2001-04-29 |                  4 |
| Pelusa | 2000-03-30 |                  3 |
+--------+------------+--------------------+
9 rows in set (0.00 sec)
```

Encontrar los animalitos cuyo cumpleaños es el próximo mes es muy sencillo. Suponiendo que el mes actual es Abril (valor 4), entonces tenemos que buscar los registros cuyo valor de mes sea 5 (Mayo).

```
mysql> SELECT nombre, nacimiento FROM mascotas WHERE MONTH(nacimiento) = 5;
+--------+------------+
| nombre | nacimiento |
+--------+------------+
| Buffy  | 1999-05-13 |
+--------+------------+
1 row in set (0.00 sec)
```

Aquí habrá por supuesto una complicación si el mes actual es Diciembre. No podemos simplemente agregar uno al número del mes (12) y buscar los registros cuyo mes de nacimiento sea 13 porque dicho mes no existe. En vez de esto, tenemos que buscar los animalitos que nacieron en Enero (mes 1).

Sin embargo, lo mejor es que podemos escribir una consulta que funcione no importando cuál sea el mes actual. La función DATE_ADD( ) nos permite agregar un intervalo de tiempo a una fecha dada. Si agregamos un mes al valor regresado por la función NOW( ), y entonces extraemos el valor del mes con la función MONTH( ), el resultado es que siempre obtendremos el mes siguiente.

La consulta que resuelve nuestro problema queda así:

```
mysql> SELECT nombre, nacimiento FROM mascotas
    -> WHERE MONTH(nacimiento) = MONTH(DATE_ADD(NOW(), INTERVAL 1 MONTH));
```

Trabajando con valores nulos

El valor NULL puede sorprendermos mientras no hayamos trabajado con él. Conceptualmente, NULL significa un valor que hace falta, o un valor desconocido, y es tratado de una manera diferente a otros valores. Para verificar si un valor es NULL no podemos usar los operadores de comparación tales como =, > , o < .
Para probar esto ejecutemos la siguiente consulta:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
1 row in set (0.00 sec)
```

Claramente observamos que no obtenemos resultados con algún significado con estos operadores. Es por ello que tenemos que usar los operadores **IS NULL** e **IS NOT NULL**:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
1 row in set (0.00 sec)
```

En MySQL, 0 o NULL significan falso y cualquier otro valor significa verdadero. El valor que se considera verdadero por default es 1.

Cuando se usa un ORDER BY, los valores NULL son siempre ordenados primero, aún cuando se use la cláusula DESC.

*Coincidencia de patrones*

MySQL proporciona métodos de coincidencia de patrones basados en SQL estándar, así como también basados en expresiones regulares, de manera similar a las utilerías de Unix tales como vi, grep y sed.

La coincidencia de patrones basada en SQL nos permite usar _ (guión bajo) para un solo caracter y **%** para un arbitrario número de caracteres. En MySQL, los patrones SQL no son sensibles al uso de

mayúsculas y minúsculas.

Es importante notar que no se usan los operadores = o < > cuando se usan los patrones SQL; en su lugar se usan los operadores **LIKE** y **NOT LIKE**.

A continuación, se presentan algunos ejemplos.

Para encontrar los nombres que comienzan con **b**:

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

Para encontrar los nombres que finalizan con fy:

```
mysql> SELECT * FROM mascotas WHERE nombre LIKE "%fy";
+--------+-------------+---------+------+------------+---------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimiento |
+--------+-------------+---------+------+------------+---------------+
| Fluffy | Arnoldo     | Gato    | f    | 1999-02-04 | NULL          |
| Buffy  | Arnoldo     | Perro   | f    | 1999-05-13 | NULL          |
+--------+-------------+---------+------+------------+---------------+
2 rows in set (0.00 sec)
```

Para encontrar nombres que contienen una s:

```
mysql> SELECT * FROM mascotas WHERE nombre LIKE "%s%";
+--------+-------------+-----------+------+------------+---------------+
| nombre | propietario | especie   | sexo | nacimiento | fallecimiento |
+--------+-------------+-----------+------+------------+---------------+
| Kaiser | Diana       | Perro     | m    | 1989-08-31 | 1997-07-29    |
| Chispa | Omar        | Ave       | f    | 1998-09-11 | NULL          |
| Skim   | Benito      | Serpiente | m    | 2001-04-29 | NULL          |
| Pelusa | Diana       | Hamster   | f    | 2000-03-30 | NULL          |
+--------+-------------+-----------+------+------------+---------------+
4 rows in set (0.01 sec)
```

El otro tipo de coincidencia de patrones proporcionado por MySQL hace uso de expresiones regulares. Para hacer uso de estos tipos de patrones se tienen que usar los operadores **REGEXP** y **NOT REGEXP** ( o RLIKE y NOT RLIKE, los cuáles son sinónimos).

Algunas características de las expresiones regulares son:

- El caracter punto ( . ) coincide con cualquier caracter.

- Una clase de caracteres [...] coincide con cualquier caracter dentro de los paréntesis cuadrados. Por ejemplo, [abc] coincide con a, b o c. Para nombrar un rango de caracteres, se usa el guión. [a-z] coincide con cualquier letra minúscula, mientras que [0-9] coincide con cualquier digito.

- El caracter asterisco ( * ) coincide con cero o más instancias de lo que le preceda. Por ejemplo, x* coincide con cualquier número de caracteres x, [0-9]* coincide con cualquier número de digitos, y .* (punto asterisco) coincide con cualquier cosa.

- El patrón coincide si éste ocurre en cualquier parte del valor que está siendo evaluado. (Los patrones SQL coinciden únicamente en los valores completos.)

- Para indicar el inicio o el final de un valor que está siendo evaluado se usan los caracteres ^ y **$** respectivamente.

Para demostrar como se usan las expresiones regulares, se van a mostrar los ejemplos presentados anteriormente con el operador LIKE, ahora con el operador REGEXP.

Para encontrar los nombre que inician con **b**:

```
mysql> SELECT * FROM mascotas WHERE nombre REGEXP "^b";
+--------+-------------+---------+------+------------+----------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimiento |
+--------+-------------+---------+------+------------+----------------+
| Buffy  | Arnoldo     | Perro   | f    | 1999-05-13 | NULL           |
+--------+-------------+---------+------+------------+----------------+
1 row in set (0.01 sec)
```

Antes de la versión 3.23.4 de MySQL, el operador REGEXP era sensible al uso de mayúsculas y minúsculas, así que dependiendo de la versión de MySQL con la que se está trabajando puede que obtengamos o no algún resultado en la consulta anterior. Se puede usar también la siguiente consulta para buscar los nombres que inician con la letra b, no importando si es mayúscula o minúscula.

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

Desde la versión 3.23.4, para forzar que el operador REGEXP sea sensible al uso de mayúsculas y minúsculas, se tiene que usar la palabra clave BINARY para hacer de una de las cadenas, una cadena binaria. Observar los resultados de la siguientes consultas.

```
mysql> SELECT * FROM mascotas WHERE nombre REGEXP BINARY "^b";
Empty set (0.00 sec)

mysql> SELECT * FROM mascotas WHERE nombre REGEXP BINARY "^B";
+--------+-------------+---------+------+------------+----------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimiento |
+--------+-------------+---------+------+------------+----------------+
| Buffy  | Arnoldo     | Perro   | f    | 1999-05-13 | NULL           |
+--------+-------------+---------+------+------------+----------------+
1 row in set (0.01 sec)
```

Para encontrar los nombres que finalizan con la palabra fy, haremos uso del caracter $.

```
mysql> SELECT * FROM mascotas WHERE nombre  REGEXP "fy$";
+--------+-------------+---------+------+------------+---------------+
| nombre | propietario | especie | sexo | nacimiento | fallecimiento |
+--------+-------------+---------+------+------------+---------------+
| Fluffy | Arnoldo     | Gato    | f    | 1999-02-04 | NULL          |
| Buffy  | Arnoldo     | Perro   | f    | 1999-05-13 | NULL          |
+--------+-------------+---------+------+------------+---------------+
2 rows in set (0.00 sec)
```

Para encontrar los nombres que contienen una letra **s**, la consulta sería:

```
mysql> SELECT * FROM mascotas WHERE nombre  REGEXP "s";
+--------+-------------+-----------+------+------------+---------------+
| nombre | propietario | especie   | sexo | nacimiento | fallecimiento |
+--------+-------------+-----------+------+------------+---------------+
| Kaiser | Diana       | Perro     | m    | 1989-08-31 | 1997-07-29    |
| Chispa | Omar        | Ave       | f    | 1998-09-11 | NULL          |
| Skim   | Benito      | Serpiente | m    | 2001-04-29 | NULL          |
| Pelusa | Diana       | Hamster   | f    | 2000-03-30 | NULL          |
+--------+-------------+-----------+------+------------+---------------+
4 rows in set (0.00 sec)
```

*Conteo de filas*

Las bases de datos son usadas frecuentemente para responder una pregunta, "¿Con qué frecuencia ocurre un cierto tipo de dato en una tabla?". Por ejemplo, tal vez queremos conocer cuántas mascotas tenemos, o cuántas mascotas tiene cada uno de los propietarios.

Contar el número total de animalitos que tenemos es lo mismo que hacer la siguiente pregunta "¿Cuántas filas hay en la tabla mascotas?" ya que hay un registro por mascota. La función COUNT( ) es la que nos ayuda en esta situación.

```
mysql> SELECT COUNT(*) FROM mascotas;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
1 row in set (0.00 sec)
```

*Si deseamos conocer cuántas mascotas tiene cada uno de los propietarios, la consulta es la siguiente:*

```
mysql> SELECT propietario, COUNT(*) FROM mascotas GROUP BY propietario ;
+-------------+----------+
| propietario | COUNT(*) |
+-------------+----------+
| Arnoldo     |        2 |
| Benito      |        2 |
| Diana       |        2 |
| Juan        |        1 |
| Omar        |        1 |
| Tomás       |        1 |
+-------------+----------+
6 rows in set (0.00 sec)
```

*Se debe notar que se ha usado una cláusula GROUP BY para agrupar todos los registros de cada propietario. Si no hacemos esto, obtendremos un mensaje de error:*

```
mysql> SELECT propietario, COUNT(*) FROM mascotas;
ERROR 1140: Mixing of GROUP columns (MIN(),MAX(),COUNT()...) with no
GROUP columns is illegal if there is no GROUP BY clause
```

*En efecto, el uso de la función COUNT( ) en conjunto con la cláusula GROUP BY es muy útil en diversas situaciones. A continuación se muestran algunos ejemplos.*

*El número de animalitos por especie:*

```
mysql> SELECT especie, COUNT(*) FROM mascotas GROUP BY especie ;
+-----------+----------+
| especie   | COUNT(*) |
+-----------+----------+
| Ave       |        2 |
| Gato      |        2 |
| Hamster   |        1 |
| Perro     |        3 |
| Serpiente |        1 |
+-----------+----------+
5 rows in set (0.00 sec)
```

El número de animalitos por sexo:

```
mysql> SELECT sexo, COUNT(*) FROM mascotas GROUP BY sexo:
+------+----------+
| sexo | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
3 rows in set (0.01 sec)
```

El número de animalitos por combinación de especie y sexo:

```
mysql> SELECT especie, sexo, COUNT(*) FROM mascotas GROUP BY especie, sexo ;
+-----------+------+----------+
| especie   | sexo | COUNT(*) |
+-----------+------+----------+
| Ave       | NULL |        1 |
| Ave       | f    |        1 |
| Gato      | f    |        1 |
| Gato      | m    |        1 |
| Hamster   | f    |        1 |
| Perro     | f    |        1 |
| Perro     | m    |        2 |
| Serpiente | m    |        1 |
+-----------+------+----------+
8 rows in set (0.00 sec)
```

No es necesario que se obtengan todos los datos de una tabla cuando se usa la función COUNT( ). Por ejemplo, en la consulta anterior, para ver únicamente los datos de perritos y gatitos, la consulta queda de la siguiente manera:

```
mysql> SELECT especie, sexo, COUNT(*) FROM mascotas
    -> WHERE especie="Perro" OR especie="Gato"
    -> GROUP BY especie, sexo;
+---------+------+----------+
| especie | sexo | COUNT(*) |
+---------+------+----------+
| Gato    | f    |        1 |
| Gato    | m    |        1 |
| Perro   | f    |        1 |
| Perro   | m    |        2 |
+---------+------+----------+
4 rows in set (0.00 sec)
```

O bien, si deseamos el número de animalitos por sexo, y cuyo sexo es conocido:

```
mysql> SELECT especie, sexo, COUNT(*) FROM mascotas
    -> WHERE sexo IS NOT NULL
    -> GROUP BY especie, sexo ;
+-----------+------+----------+
| especie   | sexo | COUNT(*) |
+-----------+------+----------+
| Ave       | f    |        1 |
| Gato      | f    |        1 |
| Gato      | m    |        1 |
| Hamster   | f    |        1 |
| Perro     | f    |        1 |
| Perro     | m    |        2 |
| Serpiente | m    |        1 |
+-----------+------+----------+
7 rows in set (0.00 sec)
```

*Usando más de una tabla*

La tabla mascotas nos ha servido hasta este momento para tener guardados los datos acerca de los animalitos que tenemos. Si deseamos guardar algún otro tipo de información acerca de ellos, tal como los eventos en sus vidas -visitas al veterinario, nacimientos de una camada, etc- necesitaremos de otra tabla. ¿Cómo deberá estar conformada esta tabla?. Lo que necesitamos es:

- El nombre de la mascota para saber a cuál de ellas se refiere el evento.
- Una fecha para saber cuando ocurrió el evento.
- Una descripción del evento.
- Un campo que indique el tipo de evento, si deseamos categorizarlos.

Dadas estas condiciones, la sentencia para crear la tabla eventos queda de la siguiente manera:

```
mysql> CREATE TABLE eventos(nombre varchar(20), fecha date,
    -> tipo varchar(15), descripcion varchar(255));
Query OK, 0 rows affected (0.03 sec)
```

De manera similar a la tabla mascotas, es más fácil cargar los datos de los registros iniciales al crear un archivo de texto delimitado por tabuladores en el que se tenga la siguiente información:

| nombre | fecha | tipo | descripción |
|---|---|---|---|
| Fluffy | 2001-05-15 | camada | 4 gatitos, 3 hembras, 1 macho |
| Buffy | 2001-06-23 | camada | 5 perritos, 2 hembras, 3 machos |
| Buffy | 2002-06-19 | camada | 2 perritos, 1 hembra, 1 macho |
| Chispa | 2000-03-21 | veterinario | Una pata lastimada |
| FanFan | 2001-08-27 | cumpleaños | Primera vez que se enfermo de la panza |
| FanFan | 2002-08-03 | veterinario | Dolor de panza |
| Whicho | 2001-02-09 | cumpleaños | Remodelación de casa |

El archivo eventos.txt

Cargamos los datos en este archivo con la siguiente sentencia:

```
mysql> LOAD DATA LOCAL INFILE "eventos.txt" INTO TABLE eventos;
Query OK, 7 rows affected (0.02 sec)
Records: 7  Deleted: 0  Skipped: 0  Warnings: 0
```

Tomando en cuenta lo que hemos aprendido en la ejecución de consultas sobre la tabla mascotas, debemos de ser capaces de recuperar algunos datos de la tabla eventos; los principios son los mismos. Sin embargo puede suceder que la tabla eventos por sí misma sea insuficiente para darnos las respuestas que necesitamos.

Supongamos que desemos conocer la edad de cada mascota cuando tuvieron una camada. La tabla eventos indica cuando ocurrió dicho evento, pero para calcular la edad de la madre, necesitamos sin duda su fecha de nacimiento. Dado que este dato está almacenado en la tabla mascotas, necesitamos de ambas tablas para realizar esta consulta.

```
mysql> SELECT mascotas.nombre,
    -> (TO_DAYS(fecha) - TO_DAYS(nacimiento))/365 AS edad,
    -> descripcion FROM mascotas, eventos
    -> WHERE mascotas.nombre=eventos.nombre
    -> AND tipo='camada';
+--------+------+---------------------------------+
| nombre | edad | descripcion                     |
+--------+------+---------------------------------+
| Fluffy | 2.28 | 4 gatitos, 3 hembras, 1 macho   |
| Buffy  | 2.12 | 5 perritos, 2 hembras, 3 machos |
| Buffy  | 3.10 | 2 perritos, 1 hembra, 1 macho   |
+--------+------+---------------------------------+
3 rows in set (0.05 sec)
```

Hay diversas cosas que notar acerca de esta consulta:

- La cláusula FROM lista dos tablas dado que la consulta necesita información que se encuentra en ambas tablas.
- Cuando se combina (junta) información de múltiples tablas, es necesario especificar los registros de una tabla que pueden coincidir con los registros en la otra tabla. En nuestro caso, ambas columnas tienen una columna "nombre". La consulta usa la cláusula WHERE para obtener los registros cuyo valor en dicha columna es el mismo en ambas tablas.
- Dado que la columna "nombre" ocurre en ambas tablas, debemos de especificar a cuál de las columnas nos referimos. Esto se hace al anteponer el nombre de la tabla al nombre de la columna.

Nota: La función TO_DAYS( ) regresa el número de días transcurridos desde el año 0 hasta la fecha dada.

No es necesario que se tengan dos tablas diferentes para que se puedan juntar. Algunas veces es útil juntar una tabla consigo misma si se desean comparar registros de la misma tabla. Por ejemplo, para encontrar las posibles parejas entre nuestras mascotas de acuerdo a la especie, la consulta sería la siguiente:

```
mysql> SELECT m1.nombre, m1.sexo, m2.nombre, m2.sexo, m1.especie
    -> FROM mascotas AS m1, mascotas AS m2
    -> WHERE m1.especie=m2.especie AND m1.sexo="f" AND m2.sexo="m";
+--------+------+--------+------+---------+
| nombre | sexo | nombre | sexo | especie |
+--------+------+--------+------+---------+
| Fluffy | f    | Mau    | m    | Gato    |
| Buffy  | f    | FanFan | m    | Perro   |
| Buffy  | f    | Kaiser | m    | Perro   |
+--------+------+--------+------+---------+
3 rows in set (0.00 sec)
```

En esta consulta se ha especificado un alias para el nombre de la tabla, y es éste el que se utiliza para referirse a las columnas.

*Usando mysql en modo batch*

En todos los ejemplos mostrados anteriormente, hemos usado **mysql** de manera interactiva para ejecutar algunas consultas y ver los resultados. Sin embargo, es posible usar **mysql** en modo batch. Para hacer esto tenemos que poner los comandos que deseamos ejecutar dentro de un archivo, y entonces decirle a **mysql** que lea los comandos de dicho archivo:

```
shell>  mysql < archivo-batch
```

Si se usa **mysql** de esta manera, se está creando un pequeño script, y posteriormente se está ejecutando dicho script.

Al ejecutar las sentencias y comandos que se encuentran en el script, es posible que suceda algún error.

Si se desea que se continuen ejecutando las demás sentencias, a pesar de que haya ocurrido un error, se tiene que usar la opción **--force**

```
shell>  mysql --force < archivo-batch
```

Así mismo, es posible especificar los parámetros de conexión desde la línea de comandos. Por ejemplo:

```
shell>  mysql -h casita -u blueman -p < archivo-batch
```

¿Por qué usar un script?

Aquí hay algunas cuantas razones:

- Si se ejecutan un cierto número de consultas frecuentemente (cada día, o cada semana), al hacer un script nos evitamos tener que volver a teclear cada una de las consultas.

- Se pueden generar nuevas consultas que sean similares a las existentes al copiar y editar estos scripts.

- Al escribir consultas de varias líneas, los scripts ayudan bastante para que no se tengan que escribir todas las líneas nuevamente si se comete algún error.

- Si se están ejecutando consultas que producen una gran cantidad de datos, es posible usar un paginador para examinar los resultados de una mejor manera.

```
shell>  mysql < archivo-batch | less
```

- Se puede guardar la salida en un archivo para revisarla posteriormente.

```
shell>  mysql < archivo-batch > salida-del-script.txt
```

- Se pueden distribuir los scripts a otras personas para que puedan ejecutar también nuestros comandos y sentencias.

- En algunas situaciones no se permite el uso interactivo de **mysql**. Por ejemplo cuando se ejecuta un **cron**. En este caso, es indispensable usar el modo batch.

Cabe mencionar que el formato de la salida es diferente (más conciso) cuando se ejecuta mysql en modo batch, que cuando se usa de manera interactiva.

Ver el siguiente ejemplo.

La consulta es: SELECT DISTINCT especie FROM mascotas.

Si se ejecuta en modo interactivo:

```
mysql> SELECT DISTINCT especie FROM mascotas;
+-----------+
| especie   |
+-----------+
| Gato      |
| Perro     |
| Ave       |
| Serpiente |
+-----------+
4 rows in set (0.00 sec)
```

Si se ejecuta en modo batch:

```
shell>  mysql -h casita -u blueman -p < especies-distintas.sql
Enter password: ******
especie
Gato
Perro
Ave
Serpiente
```

Si se desea obtener la salida que proporciona el modo interactivo, se tiene que usar la opción -t.

```
shell>  mysql -t -h casita -u blueman -p < especies-distintas.sql
Enter password: ******
+-----------+
| especie   |
+-----------+
| Gato      |
| Perro     |
| Ave       |
| Serpiente |
+-----------+
```

El archivo especies-distintas. SQL