8 Lenguaje SQL
Inserción y modificación de datos

Llegamos ahora a un punto interesante, una base de datos sin datos no sirve para mucho, de modo que veremos cómo agregar, modificar o eliminar los datos que contienen nuestras bases de datos.

Inserción de nuevas filas

La forma más directa de insertar una fila nueva en una tabla es mediante una sentencia INSERT. En la forma más simple de esta sentencia debemos indicar la tabla a la que queremos añadir filas, y los valores de cada columna. Las columnas de tipo cadena o fechas deben estar entre comillas sencillas o dobles, para las columnas númericas esto no es imprescindible, aunque también pueden estar entrecomilladas.

```
mysql> INSERT INTO gente VALUES ('Fulano','1974-04-12');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO gente VALUES ('Mengano','1978-06-15');
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO gente VALUES
    -> ('Tulano','2000-12-02'),
    -> ('Pegano','1993-02-10');
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM gente;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Fulano  | 1974-04-12 |
| Mengano | 1978-06-15 |
| Tulano  | 2000-12-02 |
| Pegano  | 1993-02-10 |
+---------+------------+
4 rows in set (0.08 sec)

mysql>
```

Si no necesitamos asignar un valor concreto para alguna columna, podemos asignarle el valor por defecto indicado para esa columna cuando se creó la tabla, usando la palabra *DEFAULT*:

```
mysql> INSERT INTO ciudad2 VALUES ('Perillo', DEFAULT);
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM ciudad2;
+---------+-----------+
| nombre  | poblacion |
+---------+-----------+
| Perillo |      5000 |
+---------+-----------+
1 row in set (0.02 sec)
mysql>
```

En este caso, como habíamos definido un valor por defecto para población de 5000, se asignará ese valor para la fila correspondiente a 'Perillo'.

Otra opción consiste en indicar una lista de columnas para las que se van a suministrar valores. A las columnas que no se nombren en esa lista se les asigna el valor por defecto. Este sistema, además, permite usar cualquier orden en las columnas, con la ventaja, con respecto a la anterior forma, de que no necesitamos conocer el orden de las columnas en la tabla para poder insertar datos:

```
mysql> INSERT INTO ciudad5 (poblacion,nombre) VALUES
    -> (7000000, 'Madrid'),
    -> (9000000, 'París'),
    -> (3500000, 'Berlín');
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|     1 | Madrid |   7000000 |
|     2 | París  |   9000000 |
|     3 | Berlín |   3500000 |
+-------+--------+-----------+
3 rows in set (0.03 sec)

mysql>
```

Cuando creamos la tabla "ciudad5" definimos tres columnas: 'clave', 'nombre' y 'poblacion' (por ese orden). Ahora hemos insertado tres filas, en las que hemos omitido la clave, y hemos alterado el orden de 'nombre' y 'poblacion'. El valor de la 'clave' se calcula automáticamente, ya que lo hemos definido

como auto-incrementado.

Existe otra sintaxis alternativa, que consiste en indicar el valor para cada columna:

```
mysql> INSERT INTO ciudad5
    -> SET nombre='Roma', poblacion=8000000;
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|     1 | Madrid |   7000000 |
|     2 | París  |   9000000 |
|     3 | Berlín |   3500000 |
|     4 | Roma   |   8000000 |
+-------+--------+-----------+
4 rows in set (0.03 sec)

mysql>
```

Una vez más, a las columnas para las que no indiquemos valores se les asignarán sus valores por defecto. También podemos hacer esto usando el valor *DEFAULT*.

Para las sintaxis que lo permitem, podemos observar que cuando se inserta más de una fila en una única sentencia, obtenemos un mensaje desde **MySQL** que indica el número de filas afectadas, el número de filas duplicadas y el número de avisos.

Para que una fila se considere duplicada debe tener el mismo valor que una fila existente para una clave principal o para una clave única. En tablas en las que no exista clave primaria ni índices de clave única no tiene sentido hablar de filas duplicadas. Es más, en esas tablas es perfectamente posible que existan filas con los mismos valores para todas las columnas.

Por ejemplo, en *mitabla5* tenemos una clave única sobre la columna 'nombre':

```
mysql> INSERT INTO mitabla5 (id, nombre) VALUES
    -> (1, 'Carlos'),
    -> (2, 'Felipe'),
    -> (3, 'Antonio'),
    -> (4, 'Carlos'),
    -> (5, 'Juan');
ERROR 1062 (23000): Duplicate entry 'Carlos' for key 1
```

```
mysql>
```

Si intentamos insertar dos filas con el mismo valor de la clave única se produce un error y la sentencia no se ejecuta. Pero existe una opción que podemos usar para los casos de claves duplicadas: *ON DUPLICATE KEY UPDATE*. En este caso podemos indicar a **MySQL** qué debe hacer si se intenta insertar una fila que ya existe en la tabla. Las opciones son limitadas: no podemos insertar la nueva fila, sino únicamente modificar la que ya existe. Por ejemplo, en la tabla 'ciudad3' podemos usar el último valor de población en caso de repetición:

```
mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('Madrid', 7000000);
Query OK, 1 rows affected (0.02 sec)

mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('París', 9000000),
    -> ('Madrid', 7200000)
    -> ON DUPLICATE KEY UPDATE poblacion=VALUES(poblacion);
Query OK, 3 rows affected (0.06 sec)
Records: 2  Duplicates: 1  Warnings: 0

mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| Madrid |   7200000 |
| París  |   9000000 |
+--------+-----------+
2 rows in set (0.00 sec)

mysql>
```

En este ejemplo, la segunda vez que intentamos insertar la fila correspondiente a 'Madrid' se usará el nuevo valor de población. Si en lugar de *VALUES(poblacion)* usamos poblacion el nuevo valor de población se ignora. También podemos usar cualquier expresión:

```
mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('París', 9100000)
    -> ON DUPLICATE KEY UPDATE poblacion=poblacion;
Query OK, 2 rows affected (0.02 sec)

mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
```

```
+--------+-----------+
| Madrid |   7200000 |
| París  |   9000000 |
+--------+-----------+
2 rows in set (0.00 sec)

mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('París', 9100000)
    -> ON DUPLICATE KEY UPDATE poblacion=0;
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| Madrid |   7200000 |
| París  |         0 |
+--------+-----------+
2 rows in set (0.00 sec)

mysql>
```

Reemplazar filas

Existe una sentencia REPLACE, que es una alternativa para INSERT, que sólo se diferencia en que si existe algún registro anterior con el mismo valor para una clave primaria o única, se elimina el viejo y se inserta el nuevo en su lugar.

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    VALUES ({expr | DEFAULT},...),(...),...
```

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
```

```
mysql> REPLACE INTO ciudad3 (nombre, poblacion) VALUES
    -> ('Madrid', 7200000),
    -> ('París', 9200000),
    -> ('Berlín', 6000000);
Query OK, 5 rows affected (0.05 sec)
```

```
Records: 3  Duplicates: 2  Warnings: 0

mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| Berlín |   6000000 |
| Madrid |   7200000 |
| París  |   9200000 |
+--------+-----------+
3 rows in set (0.00 sec)

mysql>
```

En este ejemplo se sustituyen las filas correspondientes a 'Madrid' y 'París', que ya existían en la tabla y se inserta la de 'Berlín' que no estaba previamente.

Las mismas sintaxis que existen para INSERT, están disponibles para REPLACE:

```
mysql> REPLACE INTO ciudad3 VALUES ('Roma', 9500000);
Query OK, 1 rows affected (0.03 sec)

mysql> REPLACE INTO ciudad3 SET nombre='Londres', poblacion=10000000;
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM ciudad3;
+---------+-----------+
| nombre  | poblacion |
+---------+-----------+
| Berlín  |   6000000 |
| Londres |  10000000 |
| Madrid  |   7200000 |
| París   |   9200000 |
| Roma    |   9500000 |
+---------+-----------+
5 rows in set (0.00 sec)

mysql>
```

Actualizar filas

Podemos modificar valores de las filas de una tabla usando la sentencia UPDATE. En su forma más simple, los cambios se aplican a todas las filas, y a las columnas que especifiquemos.

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Por ejemplo, podemos aumentar en un 10% la población de todas las ciudades de la tabla ciudad3 usando esta sentencia:

```
mysql> UPDATE ciudad3 SET poblacion=poblacion*1.10;
Query OK, 5 rows affected (0.15 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM ciudad3;
+---------+-----------+
| nombre  | poblacion |
+---------+-----------+
| Berlín  |   6600000 |
| Londres |  11000000 |
| Madrid  |   7920000 |
| París   |  10120000 |
| Roma    |  10450000 |
+---------+-----------+
5 rows in set (0.00 sec)

mysql>
```

Podemos, del mismo modo, actualizar el valor de más de una columna, separandolas en la sección *SET* mediante comas:

```
mysql> UPDATE ciudad5 SET clave=clave+10, poblacion=poblacion*0.97;
Query OK, 4 rows affected (0.05 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|    11 | Madrid |   6790000 |
|    12 | París  |   8730000 |
|    13 | Berlín |   3395000 |
```

```
|    14 | Roma   |   7760000 |
+-------+--------+-----------+
4 rows in set (0.00 sec)

mysql>
```

En este ejemplo hemos incrementado el valor de la columna 'clave' en 10 y disminuido el de la columna 'poblacion' en un 3%, para todas las filas.

Pero no tenemos por qué actualizar todas las filas de la tabla. Podemos limitar el número de filas afectadas de varias formas.

La primera es mediante la cláusula *WHERE*. Usando esta cláusula podemos establecer una condición. Sólo las filas que cumplan esa condición serán actualizadas:

```
mysql> UPDATE ciudad5 SET poblacion=poblacion*1.03
    -> WHERE nombre='Roma';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|    11 | Madrid |   6790000 |
|    12 | París  |   8730000 |
|    13 | Berlín |   3395000 |
|    14 | Roma   |   7992800 |
+-------+--------+-----------+
4 rows in set (0.00 sec)

mysql>
```

En este caso sólo hemos aumentado la población de las ciudades cuyo nombre sea 'Roma'. Las condiciones pueden ser más complejas. Existen muchas funciones y operadores que se pueden aplicar sobre cualquier tipo de columna, y también podemos usar operadores booleanos como *AND* u *OR*. Veremos esto con más detalle en otros capítulos.

Otra forma de limitar el número de filas afectadas es usar la cláusula *LIMIT*. Esta cláusula permite especificar el número de filas a modificar:

```
mysql> UPDATE ciudad5 SET clave=clave-10 LIMIT 2;
```

```
Query OK, 2 rows affected (0.05 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|     1 | Madrid |   6790000 |
|     2 | París  |   8730000 |
|    13 | Berlín |   3395000 |
|    14 | Roma   |   7992800 |
+-------+--------+-----------+
4 rows in set (0.00 sec)

mysql>
```

En este ejemplo hemos decrementado en 10 unidades la columna clave de las dos primeras filas.

Esta cláusula se puede combinar con *WHERE*, de modo que sólo las 'n' primeras filas que cumplan una determinada condición se modifiquen.

Sin embargo esto no es lo habitual, ya que, si no existen claves primarias o únicas, el orden de las filas es arbitrario, no tiene sentido seleccionarlas usando sólo la cláusula *LIMIT*.

La cláusula *LIMIT* se suele asociar a la cláusula *ORDER BY*. Por ejemplo, si queremos modificar la fila con la fecha más antigua de la tabla 'gente', usaremos esta sentencia:

```
mysql> UPDATE gente SET fecha="1985-04-12" ORDER BY fecha LIMIT 1;
Query OK, 1 row affected, 1 warning (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 1

mysql> SELECT * FROM gente;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Fulano  | 1985-04-12 |
| Mengano | 1978-06-15 |
| Tulano  | 2000-12-02 |
| Pegano  | 1993-02-10 |
+---------+------------+
4 rows in set (0.00 sec)

mysql>
```

Si queremos modificar la fila con la fecha más reciente, usaremos el orden inverso, es decir, el descendente:

```
mysql> UPDATE gente SET fecha="2001-12-02" ORDER BY fecha DESC LIMIT 1;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM gente;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Fulano  | 1985-04-12 |
| Mengano | 1978-06-15 |
| Tulano  | 2001-12-02 |
| Pegano  | 1993-02-10 |
+---------+------------+
4 rows in set (0.00 sec)

mysql>
```

Cuando exista una clave primaria o única, se usará ese orden por defecto, si no se especifica una cláusula *ORDER BY*.

Eliminar filas

Para eliminar filas se usa la sentencia DELETE. La sintaxis es muy parecida a la de UPDATE:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
       [WHERE where_definition]
       [ORDER BY ...]
       [LIMIT row_count]
```

La forma más simple es no usar ninguna de las cláusulas opcionales:

```
mysql> DELETE FROM ciudad3;
Query OK, 5 rows affected (0.05 sec)

mysql>
```

De este modo se eliminan todas las filas de la tabla.

Pero es más frecuente que sólo queramos eliminar ciertas filas que cumplan determinadas condiciones. La forma más normal de hacer esto es usar la cláusula *WHERE*:

```
mysql> DELETE FROM ciudad5 WHERE clave=2;
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|     1 | Madrid |   6790000 |
|    13 | Berlín |   3395000 |
|    14 | Roma   |   7992800 |
+-------+--------+-----------+
3 rows in set (0.01 sec)

mysql>
```

También podemos usar las cláusulas *LIMIT* y *ORDER BY* del mismo modo que en la sentencia UPDATE, por ejemplo, para eliminar las dos ciudades con más población:

```
mysql> DELETE FROM ciudad5 ORDER BY poblacion DESC LIMIT 2;
Query OK, 2 rows affected (0.03 sec)

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|    13 | Berlín |   3395000 |
+-------+--------+-----------+
1 row in set (0.00 sec)

mysql>
```

Vaciar una tabla

Cuando queremos eliminar todas la filas de una tabla, vimos en el punto anterior que podíamos usar una sentencia DELETE sin condiciones. Sin embargo, existe una sentencia alternativa, TRUNCATE, que realiza la misma tarea de una forma mucho más rápida.

La diferencia es que DELETE hace un borrado secuencial de la tabla, fila a fila. Pero TRUNCATE borra la tabla y la vuelve a crear vacía, lo que es mucho más eficiente.

```
mysql> TRUNCATE ciudad5;
Query OK, 1 row affected (0.05 sec)

mysql>
```

9 Lenguaje SQL
Selección de datos

Ya disponemos de bases de datos, y sabemos cómo añadir y modificar datos. Ahora aprenderemos a extraer datos de una base de datos. Para ello volveremos a usar la sentencia SELECT.

La sintaxis de SELECT es compleja, pero en este capítulo no explicaremos todas sus opciones. Una forma más general consiste en la siguiente sintaxis:

```
SELECT [ALL | DISTINCT | DISTINCTROW]
   expresion_select,...
   FROM referencias_de_tablas
   WHERE condiciones
   [GROUP BY {nombre_col | expresion | posicion}
        [ASC | DESC], ... [WITH ROLLUP]]
   [HAVING condiciones]
   [ORDER BY {nombre_col | expresion | posicion}
        [ASC | DESC] ,...]
   [LIMIT {[desplazamiento,] contador | contador OFFSET desplazamiento}]
```

Forma incondicional

La forma más sencilla es la que hemos usado hasta ahora, consiste en pedir todas las columnas y no especificar condiciones.

```
mysql>mysql> SELECT * FROM gente;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Fulano  | 1985-04-12 |
| Mengano | 1978-06-15 |
| Tulano  | 2001-12-02 |
| Pegano  | 1993-02-10 |
+---------+------------+
4 rows in set (0.00 sec)

mysql>
```

Limitar las columnas: proyección

Recordemos que una de las operaciones del álgebra relacional era la proyección, que consitía en seleccionar determinados atributos de una relación.

Mediante la sentencia SELECT es posible hacer una proyección de una tabla, seleccionando las columas de las que queremos obtener datos. En la sintaxis que hemos mostrado, la selección de columnas corresponde con la parte "expresion_select". En el ejemplo anterior hemos usado '*', que quiere decir que se muestran todas las columnas.

Pero podemos usar una lista de columnas, y de ese modo sólo se mostrarán esas columnas:

```
mysql> SELECT nombre FROM gente;
+---------+
| nombre  |
+---------+
| Fulano  |
| Mengano |
| Tulano  |
| Pegano  |
+---------+
4 rows in set (0.00 sec)

mysql> SELECT clave,poblacion FROM ciudad5;
Empty set (0.00 sec)

mysql>
```

Las expresiones_select no se limitan a nombres de columnas de tablas, pueden ser otras expresiones, incluso aunque no correspondan a ninguna tabla:

```
mysql> SELECT SIN(3.1416/2), 3+5, 7*4;
+------------------+-----+-----+
| SIN(3.1416/2)    | 3+5 | 7*4 |
+------------------+-----+-----+
| 0.99999999999325 |   8 |  28 |
+------------------+-----+-----+
1 row in set (0.00 sec)

mysql>
```

Vemos que podemos usar funciones, en este ejemplo hemos usando la función SIN para calcular el seno de π/2. En próximos capítulos veremos muchas de las funciones de las que disponemos en **MySQL**.

También podemos aplicar funciones sobre columnas de tablas, y usar esas columnas en expresiones para generar nuevas columnas:

```
mysql> SELECT nombre, fecha, DATEDIFF(CURRENT_DATE(),fecha)/365 FROM
gente;
+---------+------------+-----------------------------------+
| nombre  | fecha      | DATEDIFF(CURRENT_DATE(),fecha)/365 |
+---------+------------+-----------------------------------+
| Fulano  | 1985-04-12 |                             19.91 |
| Mengano | 1978-06-15 |                             26.74 |
| Tulano  | 2001-12-02 |                              3.26 |
| Pegano  | 1993-02-10 |                             12.07 |
+---------+------------+-----------------------------------+
4 rows in set (0.00 sec)

mysql>
```

Alias

Aprovechemos la ocasión para mencionar que también es posible asignar un alias a cualquiera de las expresiones select. Esto se puede hacer usando la palabra *AS*, aunque esta palabra es opcional:

```
mysql> SELECT nombre, fecha, DATEDIFF(CURRENT_DATE(),fecha)/365 AS edad
    -> FROM gente;
+---------+------------+-------+
| nombre  | fecha      | edad  |
+---------+------------+-------+
| Fulano  | 1985-04-12 | 19.91 |
| Mengano | 1978-06-15 | 26.74 |
| Tulano  | 2001-12-02 |  3.26 |
| Pegano  | 1993-02-10 | 12.07 |
+---------+------------+-------+
4 rows in set (0.00 sec)

mysql>
```

Podemos hacer "bromas" como:

```
mysql> SELECT 2+3 "2+2";
+-----+
| 2+2 |
+-----+
|   5 |
+-----+
1 row in set (0.00 sec)

mysql>
```

En este caso vemos que podemos omitir la palabra *AS*. Pero no es aconsejable, ya que en ocasiones puede ser difícil distinguir entre un olvido de una coma o de una palabra *AS*.

Posteriormente veremos que podemos usar los alias en otras cláusulas, como *WHERE*, *HAVING* o *GROUP BY*.

Mostrar filas repetidas

Ya que podemos elegir sólo algunas de las columnas de una tabla, es posible que se produzcan filas repetidas, debido a que hayamos excluido las columnas únicas.

Por ejemplo, añadamos las siguientes filas a nuestra tabla:

```
mysql> INSERT INTO gente VALUES ('Pimplano', '1978-06-15'),
    -> ('Frutano', '1985-04-12');
Query OK, 2 rows affected (0.03 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT fecha FROM gente;
+------------+
| fecha      |
+------------+
| 1985-04-12 |
| 1978-06-15 |
| 2001-12-02 |
| 1993-02-10 |
| 1978-06-15 |
| 1985-04-12 |
+------------+
6 rows in set (0.00 sec)

mysql>
```

Vemos que existen dos valores de filas repetidos, para la fecha "1985-04-12" y para "1978-06-15". La sentencia que hemos usado asume el valor por defecto (*ALL*) para el grupo de opciones *ALL*, *DISTINCT* y *DISTINCTROW*. En realidad sólo existen dos opciones, ya que las dos últimas: *DISTINCT* y *DISTINCTROW* son sinónimos.

La otra alternativa es usar *DISTINCT*, que hará que sólo se muestren las filas diferentes:

```
mysql> SELECT DISTINCT fecha FROM gente;
+------------+
| fecha      |
+------------+
| 1985-04-12 |
| 1978-06-15 |
| 2001-12-02 |
| 1993-02-10 |
+------------+
4 rows in set (0.00 sec)

mysql>
```

Limitar las filas: selección

Otra de las operaciones del álgebra relacional era la selección, que consitía en seleccionar filas de una realación que cumplieran determinadas condiciones.

Lo que es más útil de una base de datos es la posibilidad de hacer consultas en función de ciertas condiciones. Generalmente nos interesará saber qué filas se ajustan a determinados parámetros. Por supuesto, SELECT permite usar condiciones como parte de su sintaxis, es decir, para hacer selecciones. Concretamente mediante la cláusula *WHERE*, veamos algunos ejemplos:

```
mysql> SELECT * FROM gente WHERE nombre="Mengano";
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Mengano | 1978-06-15 |
+---------+------------+
1 row in set (0.03 sec)

mysql> SELECT * FROM gente WHERE fecha>="1986-01-01";
+--------+------------+
| nombre | fecha      |
```

```
+--------+------------+
| Tulano | 2001-12-02 |
| Pegano | 1993-02-10 |
+--------+------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM gente
    -> WHERE fecha>="1986-01-01" AND fecha < "2000-01-01";
+--------+------------+
| nombre | fecha      |
+--------+------------+
| Pegano | 1993-02-10 |
+--------+------------+
1 row in set (0.00 sec)

mysql>
```

En una cláusula *WHERE* se puede usar cualquier función disponible en **MySQL**, excluyendo sólo las de resumen o reunión, que veremos en el siguiente punto. Esas funciones están diseñadas específicamente para usarse en cláusulas *GROUP BY*.

También se puede aplicar lógica booleana para crear expresiones complejas. Disponemos de los operadores *AND*, *OR*, *XOR* y *NOT*.

En próximos capítulos veremos los operadores de los que dispone **MySQL**.

Agrupar filas

Es posible agrupar filas en la salida de una sentencia SELECT según los distintos valores de una columna, usando la cláusula *GROUP BY*. Esto, en principio, puede parecer redundante, ya que podíamos hacer lo mismo usando la opción *DISTINCT*. Sin embargo, la cláusula *GROUP BY* es más potente:

```
mysql> SELECT fecha FROM gente GROUP BY fecha;
+------------+
| fecha      |
+------------+
| 1978-06-15 |
| 1985-04-12 |
| 1993-02-10 |
| 2001-12-02 |
+------------+
4 rows in set (0.00 sec)
```

```
mysql>
```

La primera diferencia que observamos es que si se usa *GROUP BY* la salida se ordena según los valores de la columna indicada. En este caso, las columnas aparecen ordenadas por fechas.

Otra diferencia es que se eliminan los valores duplicados aún si la proyección no contiene filas duplicadas, por ejemplo:

```
mysql> SELECT nombre,fecha FROM gente GROUP BY fecha;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Mengano | 1978-06-15 |
| Fulano  | 1985-04-12 |
| Pegano  | 1993-02-10 |
| Tulano  | 2001-12-02 |
+---------+------------+
4 rows in set (0.00 sec)

mysql>
```

Pero la diferencia principal es que el uso de la cláusula *GROUP BY* permite usar funciones de resumen o reunión. Por ejemplo, la función COUNT(), que sirve para contar las filas de cada grupo:

```
mysql> SELECT fecha, COUNT(*) AS cuenta FROM gente GROUP BY fecha;
+------------+--------+
| fecha      | cuenta |
+------------+--------+
| 1978-06-15 |      2 |
| 1985-04-12 |      2 |
| 1993-02-10 |      1 |
| 2001-12-02 |      1 |
+------------+--------+
4 rows in set (0.00 sec)

mysql>
```

Esta sentencia muestra todas las fechas diferentes y el número de filas para cada fecha.

Existen otras funciones de resumen o reunión, como MAX(), MIN(), SUM(), AVG(), STD(), VARIANCE()...

Estas funciones también se pueden usar sin la cláusula *GROUP BY* siempre que no se proyecten otras columnas:

```
mysql> SELECT MAX(nombre) FROM gente;
+-------------+
| max(nombre) |
+-------------+
| Tulano      |
+-------------+
1 row in set (0.00 sec)

mysql>
```

Esta sentencia muestra el valor más grande de 'nombre' de la tabla 'gente', es decir, el último por orden alfabético.

Cláusula HAVING

La cláusula *HAVING* permite hacer selecciones en situaciones en las que no es posible usar *WHERE*. Veamos un ejemplo completo:

```
mysql> CREATE TABLE muestras (
    -> ciudad VARCHAR(40),
    -> fecha DATE,
    -> temperatura TINYINT);
Query OK, 0 rows affected (0.25 sec)

mysql> mysql> INSERT INTO muestras (ciudad,fecha,temperatura) VALUES
    -> ('Madrid', '2005-03-17', 23),
    -> ('París', '2005-03-17', 16),
    -> ('Berlín', '2005-03-17', 15),
    -> ('Madrid', '2005-03-18', 25),
    -> ('Madrid', '2005-03-19', 24),
    -> ('Berlín', '2005-03-19', 18);
Query OK, 6 rows affected (0.03 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SELECT ciudad, MAX(temperatura) FROM muestras
    -> GROUP BY ciudad HAVING MAX(temperatura)>16;
+--------+-----------------+
| ciudad | MAX(temperatura) |
+--------+-----------------+
```

```
| Berlín |               18 |
| Madrid |               25 |
+--------+------------------+
2 rows in set (0.00 sec)

mysql>
```

La cláusula *WHERE* no se puede aplicar a columnas calculadas mediante funciones de reunión, como en este ejemplo.

Ordenar resultados

Además, podemos añadir una cláusula de orden *ORDER BY* para obtener resultados ordenados por la columna que queramos:

```
mysql> SELECT * FROM gente ORDER BY fecha;
+----------+------------+
| nombre   | fecha      |
+----------+------------+
| Mengano  | 1978-06-15 |
| Pimplano | 1978-06-15 |
| Fulano   | 1985-04-12 |
| Frutano  | 1985-04-12 |
| Pegano   | 1993-02-10 |
| Tulano   | 2001-12-02 |
+----------+------------+
6 rows in set (0.02 sec)

mysql>
```

Existe una opción para esta cláusula para elegir el orden, ascendente o descendente. Se puede añadir a continuación *ASC* o *DESC*, respectivamente. Por defecto se usa el orden ascendente, de modo que el modificador *ASC* es opcional.

```
mysql> SELECT * FROM gente ORDER BY fecha DESC;
+----------+------------+
| nombre   | fecha      |
+----------+------------+
| Tulano   | 2001-12-02 |
| Pegano   | 1993-02-10 |
| Fulano   | 1985-04-12 |
| Frutano  | 1985-04-12 |
```

```
| Mengano  | 1978-06-15 |
| Pimplano | 1978-06-15 |
+----------+------------+
6 rows in set (0.00 sec)

mysql>
```

Limitar el número de filas de salida

Por último, la cláusula *LIMIT* permite limitar el número de filas devueltas:

```
mysql> SELECT * FROM gente LIMIT 3;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Fulano  | 1985-04-12 |
| Mengano | 1978-06-15 |
| Tulano  | 2001-12-02 |
+---------+------------+
3 rows in set (0.19 sec)

mysql>
```

Esta cláusula se suele usar para obtener filas por grupos, y no sobrecargar demasiado al servidor, o a la aplicación que recibe los resultados. Para poder hacer esto la clásula *LIMIT* admite dos parámetros. Cuando se usan los dos, el primero indica el número de la primera fila a recuperar, y el segundo el número de filas a recuperar. Podemos, por ejemplo, recuperar las filas de dos en dos:

```
mysql> Select * from gente limit 0,2;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Fulano  | 1985-04-12 |
| Mengano | 1978-06-15 |
+---------+------------+
2 rows in set (0.00 sec)

mysql> Select * from gente limit 2,2;
+--------+------------+
| nombre | fecha      |
+--------+------------+
| Tulano | 2001-12-02 |
```

```
| Pegano | 1993-02-10 |
+--------+------------+
2 rows in set (0.02 sec)

mysql> Select * from gente limit 4,2;
+----------+------------+
| nombre   | fecha      |
+----------+------------+
| Pimplano | 1978-06-15 |
| Frutano  | 1985-04-12 |
+----------+------------+
2 rows in set (0.00 sec)

mysql> Select * from gente limit 6,2;
Empty set (0.00 sec)

mysql>
```

10 Lenguaje SQL
Operadores

MySQL dispone de multitud de operadores diferentes para cada uno de los tipos de columna. Esos operadores se utilizan para construir expresiones que se usan en cláusulas *ORDER BY* y *HAVING* de la sentencia SELECT y en las cláusulas *WHERE* de las sentencias SELECT, DELETE y UPDATE. Además se pueden emplear en sentencias SET.

Operador de asignación

En **MySQL** podemos crear variables y usarlas porteriormente en expresiones.

Para crear una variable hay dos posibilidades. La primera consiste en ulsar la sentencia SET de este modo:

```
mysql> SET @hoy = CURRENT_DATE();
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT @hoy;
+------------+
| @hoy       |
+------------+
| 2005-03-23 |
+------------+
1 row in set (0.00 sec)

mysql>
```

La otra alternativa permite definir variables de usuario dentro de una sentencia SELECT:

```
mysql> SELECT @x:=10;
+--------+
| @x:=10 |
+--------+
|     10 |
+--------+
1 row in set (0.00 sec)

mysql> SELECT @x;
```