Lenguaje de Consulta Estructurado – SQL

¿Qué es el Lenguaje de Consulta Estructurado?

El lenguaje de consulta estructurado, **SQL,** es un lenguaje de base de datos estándar.
SQL, es un lenguaje estructurado que se utiliza para manipular datos de **Bases Relacionales**, como ser Access, Oracle, Interbase, **MySql**, etc. Dependiendo del motor de Base al que accedamos, la sintaxis puede tener algunas variantes.

El lenguaje SQL está compuesto por **comandos**, **cláusulas**, **operadores** y **funciones de agregado**. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen dos tipos de comandos SQL:

- Los DDL que permiten crear y definir nuevas bases de datos, tablas, campos e índices.
- Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DDL

- **CREATE:** Utilizado para crear nuevas tablas, Bases e índices.
- **DROP:** Empleado para eliminar tablas e índices.
- **ALTER:** Utilizado para modificar las tablas agregando campos o cambiando la definición delos campos.

Comandos DML

- **SELECT:** Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
- **INSERT:** Utilizado para cargar lotes de datos en la base de datos en una única operación.
- **UPDATE:** Utilizado para modificar los valores de los campos y registros especificados.
- **DELETE:** Utilizado para eliminar registros de una tabla de una base de datos.

Cláusulas

Las cláusulas son agregados de los comandos que nos permiten definir los datos que se desean seleccionar o manipular.

- **FROM:** Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
- **WHERE:** Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
- **GROUP BY:** Utilizada para separar los registros seleccionados en grupos específicos.
- **HAVING:** Utilizada para expresar la condición que debe satisfacer cada grupo.
- **ORDER BY:** Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico (ASC, DESC)

Operadores Lógicos

- **AND:** Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.

- **OR:** Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.

- **NOT:** Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de Comparación

< :Menor que
> :Mayor que
<> :Distinto de
<= :Menor ó Igual que
>= :Mayor ó Igual que
= :Igual que
BETWEEN: Utilizado para especificar un intervalo de valores.
LIKE: Utilizado en la comparación de un modelo
IN: Utilizado para especificar una lista de valores posibles

Funciones de agregado

Las funciones de agregación en SQL nos permiten efectuar operaciones sobre un conjunto de resultados, pero devolviendo un único valor agregado para todos ellos. Es decir, nos permiten obtener medias, máximos, etc... sobre un conjunto de valores.

- **COUNT**: devuelve el número total de filas seleccionadas por la consulta.
- **MIN**: devuelve el valor mínimo del campo que especifiquemos.
- **MAX**: devuelve el valor máximo del campo que especifiquemos.
- **SUM**: suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- **AVG**: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

Tipo de variable ~ Variables del usuario

El MySql soporta variables de usuario con la sintaxis **@variablename**.
Un nombre puede contener el **set** default de caracteres más el _, $ y el .
Las variables **no se inicializan**, ya que **por default** contienen **NULL**.
Se puede **setear el valor** de una variable con el **sentencia SET**:

SET @variable = {integer expresión | real expresión | string expresión}

También se puede asignar valor a estas variables **sin usar el SET**, pero en este caso **el operador de asignación es :=** , ya que el = se usa para comparación.
Ejemplo:
SELECT @t1 := (@t2 := 1) + @t3 := 4, @t2, @t3;

Este tipo de variable dura mientras viva la session.

FUNCIONES MYSQL ~Funciones STRING

CONCAT(str1,str2,...)
Retorna el string que resulta de concatenar los argumentos. Retorna NULL si algún argumento es NULL. Un argumento numérico es convertido al equivalente string:
mysql> select CONCAT('My', 'S', 'QL');
-> 'MySQL'

mysql> select CONCAT('My', NULL, 'QL');
-> NULL

mysql> select CONCAT(14.3);
-> '14.3'

Funciones STRING

CONCAT_WS(separator, str1, str2,...)
El primer argumento es el separador para el resto de los argumentos.
El separador puede ser un string como el resto de los argumentos.
Si el separador es NULL, el resultado será NULL. La función ignorará los strings NULLs and vacíos:

mysql> select CONCAT_WS(",","First name","Second name","Last Name");
-> 'First name,Second name,Last Name'

mysql> select CONCAT_WS(",","First name",NULL,"Last Name");
-> 'First name,Last Name

Funciones STRING

LENGTH(str)
CHAR_LENGTH(str)
CHARACTER_LENGTH(str)

Retorna la longitud de un string.
mysql> select LENGTH('text');
-> 4

mysql> select CHAR_LENGTH('hola');
-> 4

Mysql> select CHARACTER_LENGTH('str')
-> 3

Funciones STRING

LOCATE(substr,str)
POSITION(substr IN str)
Retorna la posición del substring dentro del string.
Retorna 0 en caso de no existir:

mysql> select LOCATE('bar', 'foobarbar');
-> 4

mysql> select LOCATE('xbar', 'foobar');
-> 0

Funciones STRING

LOCATE(substr,str,pos)

Retorna la posición del substring dentro del string, comenzando en la posición pos. Retorna 0 en caso de no existir.

mysql> select LOCATE('bar', 'foobarbar',5);
-> 7

Funciones STRING

LEFT(str,len)
Retorna los len caracteres de la izquierda del string:
mysql> select LEFT('foobarbar', 5);
-> 'fooba'

RIGHT(str,len)
Retorna los len caracteres de la derecha del string:
mysql> select RIGHT('foobarbar', 4);
-> 'rbar'

Funciones STRING

SUBSTRING(str,pos,len)
MID(str,pos,len)

Retorna los len caracteres del string, comenzando en la posición pos:
mysql> select SUBSTRING ('Quadratically',5,6);
-> 'ratica'

SUBSTRING(str,pos)
SUBSTRING(str FROM pos)
Retorna un substring del string a partir de la posición pos:
mysql> select SUBSTRING('Quadratically',5);
-> 'ratically'

Funciones STRING

LTRIM (str)

Retorna un string sin espacios en blanco a la izquierda, si existieran:

mysql> select LTRIM(' barbar');

-> 'barbar'

RTRIM (str)

Retorna un string sin espacios en blanco a la derecha, si existieran:

mysql> select RTRIM('barbar ');

-> 'barbar'

Funciones STRING

TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
Retorna el string sin los caracteres especificados en remstr, dependiendo de la opción usada. Si no se especifica ni BOTH, ni LEADING ni TRAILING, se asume el BOTH. Si no se especifica el remstr, se remueven los espacios en blanco:

```
mysql> select TRIM(' bar ');
-> 'bar'
mysql> select TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> select TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> select TRIM(TRAILING 'xyz' FROM 'barxxyz');
-> 'barx'
```

Funciones STRING

REPLACE(str,from_str,to_str)

Reemplaza en el string origen lo especificado en from_str por to_str:

mysql> select REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'

Funciones STRING

LCASE(str)
LOWER(str)
Convierte todos los caracteres del string a minúsculas:
mysql> select LCASE('QUADRATICALLY');
-> 'quadratically'

UCASE(str)
UPPER(str)
Convierte todos los caracteres del string a mayúsculas:
mysql> select UCASE('Hej');
-> 'HEJ'

Funciones numéricas

+ Suma:

```
mysql> SELECT 3+5;
    -> 8
```

- Resta:

```
mysql> SELECT 3-5;
    -> -2
```

- Menos unario: Cambia el signo del argumento.

```
mysql> SELECT - 2;
    -> -2
```

/ División:

```
mysql> SELECT 3/5;
    -> 0.60
```

División por cero produce un resultado NULL:

```
mysql> SELECT 102/(1-1);
    -> NULL
```

Funciones numéricas

DIV: División entera. Similar aFLOOR() pero funciona con valores BIGINT.

```
mysql> SELECT 5 DIV 2;
       -> 2
```

ABS(X): Retorna el valor absoluto de X.

```
mysql> SELECT ABS(2);
       -> 2
mysql> SELECT ABS(-32);
       -> 32
```

Funciones numéricas

ROUND(X), ROUND(X,D): Retorna el argumento X, redondeado al entero más cercano. Con dos argumentos, retorna X redondeado a D decimales. D puede ser negativo para redondear D dígitos a la izquierda del punto decimal del valor X.

```
mysql> SELECT ROUND(-1.23);
       -> -1
mysql> SELECT ROUND(-1.58);
       -> -2
mysql> SELECT ROUND(1.58);
       -> 2
mysql> SELECT ROUND(1.298, 1);
       -> 1.3
mysql> SELECT ROUND(1.298, 0);
       -> 1
mysql> SELECT ROUND(23.298, -1);
       -> 20
```

Funciones numéricas

TRUNCATE(X,D): Retorna el número X, truncado a D decimales. Si D es 0, el resultado no tiene punto decimal o parte fraccional. D puede ser negativo para truncar (hacer cero) D dígitos a la izquierda del punto decimal del valor X.

```
mysql> SELECT TRUNCATE(1.223,1);
    -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
    -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
    -> 1
mysql> SELECT TRUNCATE(-1.999,1);
    -> -1.9
mysql> SELECT TRUNCATE(122,-2);
    -> 100
```

Funciones numéricas

Todas las funciones matemáticas retornarán NULL en caso de error.
Otras funciones matemáticas que se pueden encontrar son:
ACOS(X), ASIN(X), ATAN(X), COS(X), COT(X), SIN(X), TAN(X)
LN(X) Logaritmo natural

Funciones de FECHA y HORA

DAYNAME (date)
Retorna el nombre del día de la semana de la fecha dada:
mysql> select DAYNAME("1998-02-05");
-> 'Thursday'

MONTHNAME(date)
Retorna el nombre del mes para la fecha dada:
mysql> select MONTHNAME("1998-02-05");
-> 'February'

YEAR(date)
Retorna el año de la fecha dada, en el rango de 1000 a 9999:
mysql> select YEAR('98-02-03');
-> 1998

Funciones de FECHA y HORA

EXTRACT(type FROM date)

Esta función extrae partes de una fecha, según el "type":

SECOND	segundos
MINUTE	minutos
HOUR	hora
DAY	día
MONTH	mes
YEAR	año
MINUTE_SECOND	"minutos:segundos"
HOUR_MINUTE	"hora:minutos"
DAY_HOUR	"día:hora"
YEAR_MONTH	"año:mes"
HOUR_SECOND	"hora:minutos:segundos"
DAY_MINUTE	"dia:hora:minutos"
DAY_SECOND	"dia:hora:minutos:segundos"

```
mysql> SELECT EXTRACT(YEAR FROM "1999-07-02");
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03");
-> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM "1999-07-02 01:02:03");
-> 20102
```

Funciones de FECHA y HORA

DATE_FORMAT(date,format)

Formatea una fecha de acuerdo a un formato string.

%M	Nombre del mes (January..December)
%W	Nombre del día de la semana (Sunday..Saturday)
%D	Día del mes con sufijo inglés (1st, 2nd, 3rd, etc.)
%Y	Año numérico de 4 dígitos
%y	Año numérico de 2 dígitos
%a	Día de la semana abreviado (Sun..Sat)
%d	Día del mes, numérico (00..31)
%e	Día del mes, numérico (00..31)
%m	Mes, numérico (01..12)
%c	Mes, numérico (1..12)
%b	Nombre del mes, abreviado (Jan..Dec)
%j	Día del año (001..366)

%H	hora (00..23)
%k	hora (0..23)
%h	hora (01..12)
%I	hora (01..12)
%i	minutos (00..59)
%r	Hora (base 12) (hh:mm:ss [AP]M)
%T	Hora (base 24) (hh:mm:ss)
%S	segundos (00..59)
%s	segundos (00..59)
%p	AM o PM
%w	Día de la semana, numérico (0=Sunday..6=Saturday)
%%	A literal `%'

Funciones de FECHA y HORA

Ejemplos de como aplicar la función **DATE_FORMAT**

mysql> select DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
-> 'Saturday October 1997'

mysql> select DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'

mysql> set @s1 = ‘2002-01-24’;
mysql> select date_format(@s1,%e %M %Y');
-> ’24 January 2002’;

Funciones de FECHA y HORA

CURDATE()
CURRENT_DATE
Retorna la fecha actual en formato YYYY-MM-DD o YYYYMMDD, dependiendo si la función se usa en un contexto numérico o string:
mysql> select CURDATE();
-> '1997-12-15'

mysql> select CURDATE() + 0;
-> 19971215

Funciones de FECHA y HORA

CURTIME()
CURRENT_TIME

Retorna la hora actual en formato HH:MM:SS o HHMMSS, dependiendo si la función se usa en un contexto numérico o string:
mysql> select CURTIME();
-> '23:50:26'

mysql> select CURTIME() + 0;
-> 235026

Funciones de FECHA y HORA

NOW()
SYSDATE()
CURRENT_TIMESTAMP
Retorna la fecha y hora actual en formato YYYY-MM-DD HH:MM:SS o YYYYMMDDHHMMSS, dependiendo si la función se usa en uncontexto numérico o string:

```
mysql> select NOW();
-> '1997-12-15 23:50:26'

mysql> select NOW() + 0;
->19971215235026
```

Comandos DML

Dentro de este grupo de sentencias de manipulación de datos tenemos:

- **SELECT**
- **INSERT**
- **UPDATE**
- **DELETE**

SELECT

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos.
Sintaxis:

```
SELECT [DISTINCT |ALL]
select_expression,...
[INTO {OUTFILE | DUMPFILE} '
file_name' export_options]
[FROM table_references
[WHERE where_definition]
[GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC],...]
[HAVING where_definition]
[ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC],...]
[LIMIT [offset,] rows]
```

SELECT

SELECT se usa para recuperar filas de una o más tablas.
select_expression indica las columnas que se quieren recuperar.
SELECT puede usarse también para recuperar filas sin referenciar a ninguna tabla.

Por ejemplo:
mysql> select 1 + 1;
-> 2

Todas las cláusulas deben usarse en el orden descrito. Por ejemplo una cláusula **HAVING** debe seguir a algún **GROUP BY** y estar antes del **ORDER BY** .

SELECT

Una **SELECT** puede usar alias (**AS**). El alias es como un nombre de columna y puede usarse luego en cláusulas **ORDER BY** o **HAVING**.

Por ejemplo:

```
mysql> select concat(city, ', ', id, ' ', name) as identify
           from customer
           order by identif;
```

La cláusula **table_references** indica las tablas desde las cuales se recuperarán las filas. En caso de haber más de una tabla, estaremos en presencia de un **join**.
Las columnas pueden referenciarse como: **db_name.tbl_name.col_name**, **Tbl_name.col_name** o simplemente **col_name.**
No se necesitan usar los prefijos de las columnas, a no ser que se hagan referencias ambiguas.
Por ejemplo:

```
 mysql> select concat(e.last_name, ', ', e.first_name), d.name
           from emp as e, depto as d
           where e.dept_id = d.id;
```

SELECT

Las columnas seleccionadas pueden referenciarse en un **ORDER BY** y **GROUP BY** usando los nombres de las columnas, los alias de las columnas o las posiciones de las mismas dentro de la select, comenzando con 1:

```
mysql> select name, city
           from customer
           order by city, name;

mysql> select name as n, city as c
           from customer
           order by c, n;

mysql> select name, city
           from customer
           order by 2, 1;
```

El ordenamiento por default es ascendente (**ASC**), pero si queremos invertir ese orden debemos poner la palabra **DESC** (descendente).

SELECT

En la cláusula **WHERE** se pueden usar cualquiera de las funciones soportadas por el MySQL.

LIMIT tiene 1 ó 2 argumentos numéricos. En el caso de haber dos argumentos, el primero especifica la 1er. fila a retornar y el segundo la cantidad máxima de filas:

```
    mysql> select *
                from emp
                limit 5,3; # Retrieve rows 6-8

mysql> select *
                from emp
                limit 5;   # Retrieve first 5 rows
```

Ordenamiento de los registros: ORDER BY

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula **ORDER BY** Lista de Campos. En donde Lista de campos representa los campos a ordenar.
Ejemplo:

```
mysql>      select last_name, first_name, id, dept_id
            from emp
            order by last_name;
```

Esta consulta devuelve los campos last_name, first_name, id, dept_id de la tabla emp ordenados por el campo last_name. Se pueden ordenar los registros por más de un campo, como por ejemplo:

```
mysql>      select last_name, first_name, id, dept_id
            from emp
            order by dept_id, last_name;
```

Incluso se puede especificar el orden de los registros: ascendente mediante la cláusula (ASC -se toma este valor por defecto) ó descendente (DESC)

```
mysql> select last_name, first_name, id, dept_id
        from emp
        order by dept_id desc, last_name asc;
```

Consultas con Predicado

El predicado se incluye entre la cláusula y el primer nombre del campo a recuperar, los posibles predicados son:

PREDICADO	DESCRIPCIÓN
ALL	Devuelve todos los campos de la tabla
DISTINCT	Omite los registros cuyos campos seleccionados coincidan totalmente

Consultas con Predicado

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No es conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL *
FROM Emp;
```

Consultas con Predicado

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos. Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
mysql> select distinct last_name
            from emp;
```

Con otras palabras el predicado **DISTINCT** devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza **DISTINCT** no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

Alias

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada **AS** que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
mysql> select id, last_name as Empleado
       from emp;
```

La cláusula WHERE

La cláusula **WHERE** puede usarse para determinar qué registros de las tablas enumeradas en la cláusula **FROM** aparecerán en los resultados de la instrucción **SELECT**. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. **WHERE** es opcional, pero cuando aparece debe ir a continuación de FROM.
Cabe aclarar que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples. Las fechas se deben escribir también encerradas entre comillas dobles o simples, respetando el formato de la columna tipo DATE a la que nos referimos. Sino, la consulta no nos retornará filas.

```
mysql> select last_name, salary
            from emp where salary > 1800;

mysql>select * from ord  where id <= 100;

mysql> select *  from ord  where date_ordered = '1992-09-04';
```

Ejemplos de WHERE

```
mysql> select id, name, address, city from customer
          where city = 'Lagos';

mysql> select * from emp
          where last_name like 'S%';

mysql> select * from emp
          where salary between 500 and 1000;

mysql> select * from emp
          where last_name between 'AD' and 'CL';

mysql> select name, address from customer
          where city in ('Hong Kong', 'Nogales', 'Prague');
```

Operadores Lógicos

Todas las comparaciones lógicas retornan 1 (TRUE) o 0 (FALSE). Los operadores lógicos soportados son: **AND**, **OR**, **NOT** y **IS**. A excepción de los dos últimos todos poseen la siguiente sintaxis:
<expresión1> operador <expresión2>
En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. Si a cualquiera de las anteriores condiciones le anteponemos el operador **NOT** el resultado de la operación será el contrario al devuelto sin el operador **NOT**.

```
mysql> select * from emp
          where salary > 1200 and salary < 1510;

mysql> select * from emp
          where NOT (salary > 1200 and salary < 1510);
```

Operadores Lógicos

```
mysql> select * from emp
       where (salary > 1200 and salary < 1510) or
             commission_pct = 15.00;
mysql> select * from emp
       where title NOT IN ('Stock Clerk','Warehouse Manager');
mysql> select * from emp
       where (salary > 1200 and salary < 1600) or
             (title != "Stock Clerk" and id = 30);
```

Debido a que el valor NULL es denominado valor desconocido o indefinido, para comparaciones no se pueden usar los operadores ya vistos, por lo cual se emplea el operador IS.

```
mysql> select * from emp
       where commission_pct is NULL;

mysql> select * from emp
       where commission_pct is not NULL;
```

Intervalos de Valores
El Operador BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador **BETWEEN** cuya **sintaxis** es:

campo [Not] Between valor1 And valor2 (la condición Not es opcional)

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

- mysql> select * from emp
  where last_name between “A” and “CZZZZZZ”;

(Devuelve los empleados cuyos apellidos estén comprendidos entre la “A” y la “C”).

El Operador LIKE

Se utiliza para comparar una expresión de cadena con un modelo o patrón, en una expresión SQL. Su sintaxis es:
expr LIKE pat [ESCAPE 'escape-char']
Con el **LIKE** se pueden usar 2 tipos de meta-caracteres:

%	Reemplaza de 0 a n caracteres
_	Reemplaza un carácter posicional

Con el comodín %
SELECT * FROM nombre_tablaWHERE nombre_columna LIKE '%PATRON%';
Realizamos una SELECT con el comodín '%', este tiene como finalidad ignorar todos los caracteres que estén antes y/o después del patrón.

Con el comodín _
SELECT * FROM nombre_tabla WHERE nombre_columna LIKE _%CARACTER_';
Realizamos una SELECT con el comodín '_', este tiene como finalidad ignorar un único caracter por cada '_' que esté antes y/o después del patrón.

El Operador LIKE - Ejemplos

Buscamos los clientes que contengan una ‘m’ en el nombre.
SELECT * FROM clientes WHERE nombre LIKE '%m%';

Buscamos los clientes cuyo nombre comienza por ‘M’.
SELECT * FROM clientes WHERE nombre LIKE 'M%';

Buscamos los clientes cuyo nombre termina con ‘a’.
SELECT * FROM clientes WHERE nombre LIKE '%a';

Buscamos los clientes cuyo nombre tiene como segundo caracter la ‘a’.
SELECT * FROM clientes WHERE nombre LIKE '_a%';

Buscamos los clientes cuyo nombre tiene 7 caracteres de longitud.
SELECT * FROM clientes WHERE nombre LIKE '_______';

Buscamos los clientes cuyo nombre tiene una ‘a’ y una ‘r’.
SELECT * FROM clientes WHERE nombre LIKE '%a%' or nombre LIKE '%r%' ;

El Operador IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los de la lista.
Su sintaxis es:

expresión [Not] In(valor1, valor2, . . .)

```
mysql> select * from emp
          where title in ('VP, Sales', 'President');
```