

Sumario

| | |
|-------------------------------------|----|
| Insertar registros en la tabla..... | 2 |
| Selección de registros..... | 3 |
| Sentencia Select..... | 3 |
| Condiciones Lógicas..... | 3 |
| Tabla de Operadores Lógicos..... | 4 |
| Cláusulas Especiales..... | 4 |
| Sentencia BETWEEN..... | 4 |
| Sentencia IN..... | 4 |
| Sentencia LIKE..... | 5 |
| Otros ejemplos de select..... | 5 |
| Consultas avanzadas..... | 12 |

Insertar registros en la tabla

Para insertar datos en una tabla utilizamos la orden INSERT. Con ella podemos ir añadiendo registros uno a uno, o añadir de golpe tantos registros como deseemos.

La sintaxis genérica de INSERT para crear un nuevo registro es la siguiente:

```
INSERT INTO NombreTabla (Columna1, ..., ColumnaX) VALUES (Valor1, ..., ValorX)
```

... siendo ...

- NombreTabla: la tabla en la que se van a insertar las filas o registros.
- (Campo1, ..., CampoX): representa el campo o campos en los que vamos a introducir valores.
- (Valor1, ..., ValorX): representan los valores que se van a almacenar en cada campo.

Ejemplo:

```
INSERT INTO Productos (Nombre, Precio, Marca, Categoria, Presentacion, Stock, Disponible)  
VALUES ('iPhone 5' , 499.99 , 'Apple' , 'Smartphone' , '16GB', 500, false);
```

Los campos deben coincidir exactamente igual a como han sido definidos mediante el comando CREATE.

Los valores se deben corresponder con cada uno de los campos que aparecen en la lista de campos, tanto en el tipo de dato que contienen como en el orden en el que se van a asignar. Es decir, si se indican una serie de campos en un orden determinado, la lista de valores debe especificar los valores a almacenar en dichos campos, en el mismo orden exactamente. Si un campo no está en la lista, se almacenará dentro de éste el valor NULL.

Valores NULL

La expresión NULL significa "dato desconocido" o "valor inexistente". No es lo mismo que un valor 0, una cadena vacía o una cadena de texto literal con la palabra NULL.

A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos. Por ejemplo, en la tabla de Productos se puede tener valores nulos en el campo Precio porque es posible que para algunos productos no se haya establecido el precio para la venta.

En contraposición, tenemos campos que no pueden estar vacíos jamás, por ejemplo, los campos que identifican cada registro, como los códigos de identificación, que son clave primaria.

Por defecto, es decir, si no lo aclaramos en la creación de la tabla, los campos permiten valores nulos.

Por ejemplo:

```
INSERT INTO Productos (Nombre, Precio, Marca, Categoria, Presentacion, Stock) VALUES  
( 'iPhone7S', NULL, 'Apple', 'Smartphone', '16GB', 500);
```

Nótese que para el campo Precio el valor NULL no es una cadena de caracteres, por lo que no se coloca entre comillas. También si un campo acepta valores nulos, podemos ingresar NULL cuando no conocemos el valor.

Además, si una columna fue definida en el comando CREATE con la especificación NULL (como ser en el campo Disponible), puede no incluirse en el listado de campos (puede entenderse como que el campo "no es obligatorio" de asignar un valor).

Selección de registros

Sentencia Select

La sentencia SELECT va a permitir realizar operaciones de selección, ordenación, agrupación y filtrado de registros, es decir:

- Se utiliza para seleccionar información registrada en una tabla.
- Para seleccionar todas la columnas se utiliza el * (asterisco)
- La cláusula WHERE se utiliza para establecer un criterio de búsqueda

Sintaxis #1: Seleccionar de la tabla Productos los valores de todas las columnas de todos los registros

```
SELECT * FROM Productos;
```

Sintaxis #2: Seleccionar de la tabla Productos los valores de la columna Nombre de todos los registros

```
SELECT Nombre FROM Productos;
```

Sintaxis #3: Seleccionar de la tabla Productos todos los valores de la columna Nombre y Precio de todos los registros

```
SELECT Nombre, Precio FROM Productos;
```

Sintaxis #4: Seleccionar de la tabla Productos la columna Nombre del registro cuyo valor de la columna idProducto es igual a 1

```
SELECT Nombre FROM Productos WHERE idProducto = 1;
```

Sintaxis #5: Seleccionar de la tabla Productos las columnas Nombre y Precio los registros cuyo valor de la columna Precio sea Mayor a 150

```
SELECT Nombre, Precio FROM Productos WHERE Precio > 150;
```

Condiciones Lógicas

Para crear expresiones lógicas disponemos de varios operadores de comparación. Estos operadores se aplican a cualquier tipo de columna: fechas, cadenas, números, etc, y devuelven valores lógicos: verdadero o falso (o bien 1 ó 0).

Si uno o los dos valores a comparar son NULL , el resultado es NULL, excepto con el operador <=> que es usado para una comparación con NULL segura.

El operador <=> funciona igual que el operador =, salvo que si en la comparación una o ambas de las expresiones es nula el resultado no es NULL. Si se comparan dos expresiones nulas, el resultado es verdadero.

Tabla de Operadores Lógicos

| Operador | Descripción |
|----------|---------------|
| = | Igual |
| < | Menor |
| > | Mayor |
| >= | Mayor o igual |
| <= | Menor o igual |
| <> | No es igual |

Los operadores IS NULL e IS NOT NULL sirven para verificar si una expresión determinada es o no nula.

Ejemplos de aplicación:

```
SELECT * FROM Productos WHERE Precio >= 500 OR Stock >= 100;  
SELECT Nombre, Presentacion, Precio, Stock FROM Productos WHERE Stock < 20 AND  
Precio >= 100 ;  
SELECT Nombre, Presentacion FROM Productos WHERE Precio IS NOT NULL;
```

Cláusulas Especiales

Sentencia BETWEEN

Entre los operadores de MySQL, hay uno para comprobar si una expresión está comprendida en un determinado rango de valores. La sintaxis es:

- BETWEEN mínimo AND máximo
- NOT BETWEEN mínimo AND máximo

Ejemplo de aplicación:

```
SELECT * FROM Productos WHERE Precio BETWEEN 100 AND 200;  
SELECT * FROM Productos WHERE Precio NOT BETWEEN 100 AND 200;
```

Sentencia IN

Los operadores IN y NOT IN sirven para averiguar si el valor de una expresión determinada está dentro de un conjunto indicado

- IN (<expr1>, <expr2>, <expr3>,...)
- NOT IN (<expr1>, <expr2>, <expr3>,...)

El operador IN devuelve un valor verdadero si el valor de la expresión es igual a alguno de los valores especificados en la lista. El operador NOT IN devuelve un valor falso en el mismo caso. Por ejemplo:

```
SELECT 10 IN ( 2 , 4 , 6 , 8 , 10 );
```

Ejemplo de aplicación:

```
SELECT * FROM Productos WHERE Nombre IN ('iPhone','iPad' );  
SELECT * FROM Productos WHERE Nombre NOT IN ('Galaxy S5', 'TV' );
```

Sentencia LIKE

El operador LIKE se usa para hacer comparaciones entre cadenas y patrones. El resultado es verdadero (1) si la cadena se ajusta al patrón, y falso (0) en caso contrario. Tanto si la cadena como el patrón son NULL, el resultado es NULL.

La sintaxis es:

```
SELECT * FROM Productos LIKE <patrón> [ESCAPE 'carácter_escape' ]
```

| Carácter | Descripción |
|----------|---|
| % | Coincidencia con cualquier número de caracteres, incluso ninguno. |
| _ | Coincidencia con un único carácter. |

La comparación es independiente del tipo de los caracteres, es decir, LIKE no distingue mayúsculas de minúsculas, salvo que se indique lo contrario (ver operadores de casting):

Como siempre que se usan caracteres concretos para crear patrones, se presenta la dificultad de hacer comparaciones cuando se deben buscar precisamente esos caracteres concretos. Esta dificultad se suele superar mediante secuencias de escape. Si no se especifica nada en contra, el carácter que se usa para escapar es '\'. De este modo, si queremos que nuestro patrón contenga los caracteres '%' o '_', los escaparemos de este modo: '\\' y '\\':

Como en cualquier otro lenguaje, los paréntesis se pueden usar para forzar el orden de la evaluación de determinadas operaciones dentro de una expresión. Cualquier expresión entre paréntesis adquiere mayor precedencia que el resto de las operaciones en el mismo nivel de paréntesis.

Ejemplo de aplicación:

```
-- Listar todos los nombres que contienen TV  
SELECT * FROM Productos WHERE Nombre LIKE '%TV%';
```

Otros ejemplos de select

```
SELECT COUNT(*) FROM <nombre_tabla>
```

Devuelve el total de registros en la tabla.

```
SELECT COUNT(*) FROM employees;
```

```
Terminal
499993 | 1963-06-04 | DeForest | Mullainathan | M | 1997-04-07
499994 | 1952-02-26 | Navin | Argence | F | 1990-04-24
499995 | 1958-09-24 | Dekang | Lichtner | F | 1993-01-12
499996 | 1953-03-07 | Zito | Baaz | M | 1990-09-27
499997 | 1961-08-03 | Berhard | Lenart | M | 1986-04-21
499998 | 1956-09-05 | Patricia | Breugel | M | 1993-10-13
499999 | 1958-05-01 | Sachin | Tsukuda | M | 1997-11-30
+-----+-----+-----+-----+-----+-----+
+
300024 rows in set (0.42 sec)

mysql> SELECT COUNT(*) FROM employees;
+-----+
| COUNT(*) |
+-----+
| 300024 |
+-----+
1 row in set (0.07 sec)

mysql>
```

SELECT * FROM <nombre_tabla> LIMIT <limite_filas> OFFSET <número_pagina>

Lista un número de registros limitado.

SELECT * FROM employees LIMIT 10;

```
Terminal

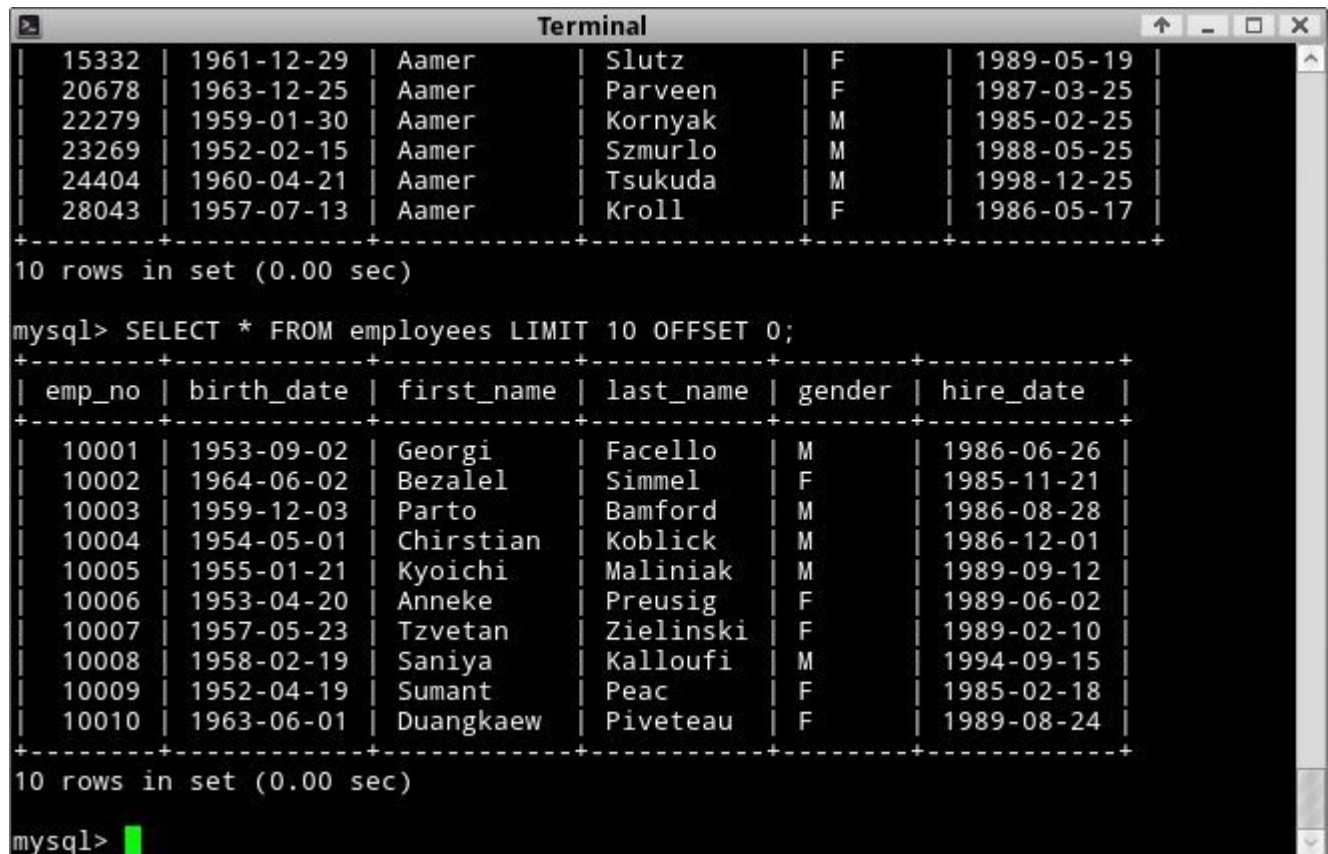
mysql> SELECT COUNT(*) FROM employees;
+-----+
| COUNT(*) |
+-----+
| 300024 |
+-----+
1 row in set (0.07 sec)

mysql> SELECT * FROM employees LIMIT 10;
+-----+-----+-----+-----+-----+-----+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
+-----+-----+-----+-----+-----+-----+
| 10001 | 1953-09-02 | Georgi | Facello | M | 1986-06-26 |
| 10002 | 1964-06-02 | Bezalel | Simmel | F | 1985-11-21 |
| 10003 | 1959-12-03 | Parto | Bamford | M | 1986-08-28 |
| 10004 | 1954-05-01 | Chirstian | Koblick | M | 1986-12-01 |
| 10005 | 1955-01-21 | Kyoichi | Maliniak | M | 1989-09-12 |
| 10006 | 1953-04-20 | Anneke | Preusig | F | 1989-06-02 |
| 10007 | 1957-05-23 | Tzvetan | Zielinski | F | 1989-02-10 |
| 10008 | 1958-02-19 | Saniya | Kalloufi | M | 1994-09-15 |
| 10009 | 1952-04-19 | Sumant | Peac | F | 1985-02-18 |
| 10010 | 1963-06-01 | Duangkaew | Piveteau | F | 1989-08-24 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql>
```

También podemos definir el número de página deseado.

```
SELECT * FROM employees LIMIT 10 OFFSET 0;
```



The image shows a terminal window titled "Terminal" with a black background and white text. It displays the output of a MySQL query. The first query is not explicitly shown but its results are displayed as a table with 6 columns: emp_no, birth_date, first_name, last_name, gender, and hire_date. The results show 10 rows of employee data. Below the table, it says "10 rows in set (0.00 sec)". The second query is "mysql> SELECT * FROM employees LIMIT 10 OFFSET 0;". Its results are also displayed as a table with the same 6 columns, showing 10 rows of employee data starting from the second row of the first query's results. Below this table, it also says "10 rows in set (0.00 sec)". The prompt "mysql>" is followed by a green cursor.

```
mysql> SELECT * FROM employees LIMIT 10 OFFSET 0;
```

| emp_no | birth_date | first_name | last_name | gender | hire_date |
|--------|------------|------------|-----------|--------|------------|
| 10001 | 1953-09-02 | Georgi | Facello | M | 1986-06-26 |
| 10002 | 1964-06-02 | Bezalel | Simmel | F | 1985-11-21 |
| 10003 | 1959-12-03 | Parto | Bamford | M | 1986-08-28 |
| 10004 | 1954-05-01 | Chirstian | Koblick | M | 1986-12-01 |
| 10005 | 1955-01-21 | Kyoichi | Maliniak | M | 1989-09-12 |
| 10006 | 1953-04-20 | Anneke | Preusig | F | 1989-06-02 |
| 10007 | 1957-05-23 | Tzvetan | Zielinski | F | 1989-02-10 |
| 10008 | 1958-02-19 | Saniya | Kalloufi | M | 1994-09-15 |
| 10009 | 1952-04-19 | Sumant | Peac | F | 1985-02-18 |
| 10010 | 1963-06-01 | Duangkaew | Piveteau | F | 1989-08-24 |

```
mysql>
```

Para obtener la siguiente página basta con incrementar el **OFFSET**.

```
SELECT * FROM employees LIMIT 10 OFFSET 1;
```

```
Terminal
+-----+-----+-----+-----+-----+-----+
| 10005 | 1955-01-21 | Kyoichi | Maliniak | M | 1989-09-12 |
| 10006 | 1953-04-20 | Anneke | Preusig | F | 1989-06-02 |
| 10007 | 1957-05-23 | Tzvetan | Zielinski | F | 1989-02-10 |
| 10008 | 1958-02-19 | Saniya | Kalloufi | M | 1994-09-15 |
| 10009 | 1952-04-19 | Sumant | Peac | F | 1985-02-18 |
| 10010 | 1963-06-01 | Duangkaew | Piveteau | F | 1989-08-24 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql> SELECT * FROM employees LIMIT 10 OFFSET 1;
+-----+-----+-----+-----+-----+-----+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
+-----+-----+-----+-----+-----+-----+
| 10002 | 1964-06-02 | Bezalel | Simmel | F | 1985-11-21 |
| 10003 | 1959-12-03 | Parto | Bamford | M | 1986-08-28 |
| 10004 | 1954-05-01 | Chirstian | Koblick | M | 1986-12-01 |
| 10005 | 1955-01-21 | Kyoichi | Maliniak | M | 1989-09-12 |
| 10006 | 1953-04-20 | Anneke | Preusig | F | 1989-06-02 |
| 10007 | 1957-05-23 | Tzvetan | Zielinski | F | 1989-02-10 |
| 10008 | 1958-02-19 | Saniya | Kalloufi | M | 1994-09-15 |
| 10009 | 1952-04-19 | Sumant | Peac | F | 1985-02-18 |
| 10010 | 1963-06-01 | Duangkaew | Piveteau | F | 1989-08-24 |
| 10011 | 1953-11-07 | Mary | Sluis | F | 1990-01-22 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql> █
```

Este es el concepto de una paginación.

SELECT * FROM <nombre_tabla> ORDER BY <nombre_campo>

Lista los registros ordenados por un campo. El ordenamiento puede ser ascendente o descendente.

```
SELECT * FROM employees ORDER BY first_name ASC LIMIT 10;
SELECT * FROM employees ORDER BY first_name DESC LIMIT 10;
```

Si ven, podemos combinarlos con las otras sentencias.

```
SELECT * FROM employees ORDER BY first_name ASC LIMIT 10 OFFSET 1;
```



```
Terminal
mysql> SELECT * FROM employees ORDER BY first_name ASC LIMIT 10 OFFSET 1;
+-----+-----+-----+-----+-----+-----+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
+-----+-----+-----+-----+-----+-----+
| 11935 | 1963-03-23 | Aamer | Jayawardene | M | 1996-10-26 |
| 12160 | 1954-12-11 | Aamer | Garrabrants | M | 1989-09-19 |
| 13011 | 1955-02-25 | Aamer | Glowinski | F | 1989-10-08 |
| 15332 | 1961-12-29 | Aamer | Slutz | F | 1989-05-19 |
| 20678 | 1963-12-25 | Aamer | Parveen | F | 1987-03-25 |
| 22279 | 1959-01-30 | Aamer | Kornyak | M | 1985-02-25 |
| 23269 | 1952-02-15 | Aamer | Szmurlo | M | 1988-05-25 |
| 24404 | 1960-04-21 | Aamer | Tsukuda | M | 1998-12-25 |
| 28043 | 1957-07-13 | Aamer | Kroll | F | 1986-05-17 |
| 28162 | 1960-04-06 | Aamer | Parhami | F | 1996-02-15 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.24 sec)

mysql> SELECT MAX(hire_date) FROM employees;
+-----+
| MAX(hire_date) |
+-----+
| 2000-01-28 |
+-----+
1 row in set (0.14 sec)

mysql>
```

SELECT MAX|MIN(<nombre_campo>) FROM <nombre_tabla>

Muestra el valor mayor o menor del campo en la tabla.

```
SELECT MAX(hire_date) FROM employees;
SELECT MIN(hire_date) FROM employees;
```

```
Terminal
+-----+-----+-----+-----+-----+-----+
| 15332 | 1961-12-29 | Aamer | Slutz | F | 1989-05-19 |
| 20678 | 1963-12-25 | Aamer | Parveen | F | 1987-03-25 |
| 22279 | 1959-01-30 | Aamer | Kornyak | M | 1985-02-25 |
| 23269 | 1952-02-15 | Aamer | Szmurlo | M | 1988-05-25 |
| 24404 | 1960-04-21 | Aamer | Tsukuda | M | 1998-12-25 |
| 28043 | 1957-07-13 | Aamer | Kroll | F | 1986-05-17 |
| 28162 | 1960-04-06 | Aamer | Parhami | F | 1996-02-15 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.24 sec)

mysql> SELECT MAX(hire_date) FROM employees;
+-----+
| MAX(hire_date) |
+-----+
| 2000-01-28 |
+-----+
1 row in set (0.14 sec)

mysql> SELECT MIN(hire_date) FROM employees;
+-----+
| MIN(hire_date) |
+-----+
| 1985-01-01 |
+-----+
1 row in set (0.10 sec)

mysql> █
```

SELECT DISTINCT(<nombre_campo>) FROM <nombre_tabla>

Muestra los los diferentes valores de un campo en la tabla. Ningún valor aparecerá repetido.

```
SELECT DISTINCT(gender) FROM employees;
```

SELECT <nombre_campo> FROM <nombre_tabla>

Lista campos específicos de la tabla. Si hay más de un campo, estos deben delimitarse por una ",".

```
SELECT first_name, last_name, gender FROM employees LIMIT 10;
```

```
Terminal
mysql> SELECT first_name, last_name, gender FROM employees LIMIT 10;
+-----+-----+-----+
| first_name | last_name | gender |
+-----+-----+-----+
| Georgi     | Facello   | M      |
| Bezalel    | Simmel    | F      |
| Parto      | Bamford   | M      |
| Chirstian  | Koblick   | M      |
| Kyoichi    | Maliniak  | M      |
| Anneke     | Preusig   | F      |
| Tzvetan    | Zielinski | F      |
| Saniya     | Kalloufi  | M      |
| Sumant     | Peac      | F      |
| Duangkaew | Piveteau  | F      |
+-----+-----+-----+
10 rows in set (0.00 sec)

mysql> SELECT SUM(salary) FROM salaries;
+-----+
| SUM(salary) |
+-----+
| 181480757419 |
+-----+
1 row in set (0.00 sec)

mysql>
```

SELECT SUM(<nombre_campo>) FROM <nombre_tabla>

Devuelve la sumatoria de campos numéricos.

```
SELECT SUM(salary) FROM salaries;
```

SELECT * FROM <nombre_tabla> WHERE <condiciones>

La sentencia **WHERE** nos sirve para filtrar registros por una serie de condiciones definidas por nosotros. Las condiciones deben cumplirse y pueden ser anidadas por los operadores lógicos **OR** y **AND**.

```
SELECT * FROM employees WHERE gender = 'M' AND last_name = 'Facello' LIMIT 10;
```

En este ejemplo estamos filtrando los registros donde el género sea **M** y además el apellido sea igual a **Facello**.

```
SELECT * FROM employees WHERE gender = 'M' AND (last_name = 'Facello' OR last_name = 'Simmel') LIMIT 10;
```

En el segundo ejemplo el campo género debe ser **M** y el apellido puede ser **Facello** o **Simmel**. Cuando se trabaja con diferentes operadores lógicos es necesario agruparlos por paréntesis para evitar malas interpretaciones del motor SQL.

Existen otros operadores condicionales como por ejemplo:

> mayor que

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> > <valor>;
```

< menor que

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> < <valor>;
```

>= mayor o igual que

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> >= <valor>;
```

<= menor o igual que

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> <= <valor>;
```

<> diferente a

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> <> <valor>;
```

!= no igual que

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> != <valor>;
```

IS NULL nulo

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> IS NULL;
```

IS NOT NULL no nulo

```
SELECT * FROM <nombre_tabla> WHERE <nombre_campo> IS NOT NULL;
```

Consultas avanzadas

Para extraer los datos y que se presenten en pantalla se utiliza la orden SELECT. Su completa constitución sería:

```
SELECT {NombreCampo, NombreCampo, ...}  
FROM NombreTablaWHERE Condiciones  
[ GROUP BY { NombreCampo | Expresión | Posición } ]  
[ HAVING Condiciones ]  
[ ORDER BY { NombreCampo | Expresión | Posición } ]  
[ ASC | DESC ]  
[ LIMIT { n }, { n } ]
```

Mediante la sentencia SELECT es posible hacer una proyección de una tabla, seleccionando las columnas de las que queremos obtener datos.

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

Es posible aplicar funciones sobre columnas de tablas, y usar esas columnas en expresiones para generar nuevas columnas. Por ejemplo:

```
SELECT idVenta, Fecha, DATEDIFF ( CURRENT_DATE(), Fecha ) / 30 FROM Ventas;
```

Es posible asignar un alias a cualquiera de las expresiones select. Esto se puede hacer usando la palabra AS, aunque esta palabra es opcional:

```
SELECT idVenta, Fecha, DATEDIFF(CURRENT_DATE(), Fecha ) / 30 Días FROM Ventas;
```