

## TEMA 8. LAS CONSULTAS SIMPLES

La sentencia SELECT es con mucho la más compleja y potente de las sentencias SQL. Empezaremos por ver las **consultas** más **simples**, **basadas** en **una** sola **tabla**.

Esta sentencia forma parte del DML (lenguaje de manipulación de datos), en este tema veremos cómo **seleccionar columnas** de una tabla, cómo **seleccionar filas** y cómo obtener las **filas ordenadas** por el criterio que queramos.

El resultado de la consulta es una **tabla lógica**, porque no se guarda en el disco sino que está en memoria y cada vez que ejecutamos la consulta se vuelve a calcular.

Cuando ejecutamos la consulta se visualiza el resultado en forma de tabla con columnas y filas, pues en la SELECT tenemos que indicar qué columnas queremos que tenga el resultado y qué filas queremos seleccionar de la tabla origen.

### Sintaxis de la sentencia SELECT (consultas simples)

Las consultas permiten seleccionar datos procedentes de varias tablas relacionadas o utilizar los datos de una única tabla, realizar cálculos sobre los datos.

Para ello usamos la sentencia **SELECT**.

```
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}]
```

SELECT lista\_selección # Columnas a seleccionar

FROM lista\_tablas # Tablas de donde seleccionamos las columnas

WHERE restricción\_principal #Condiciones que deben satisfacer las filas

GROUP BY agrupación\_columnas # como agrupar los resultados

ORDER BY orden\_columnas #como ordenar los resultados

HAVING restricción\_secundaria #condiciones secundarias que deben satisfacer las filas

LIMIT contador; # limitación del número de filas del resultado

## La lista de selección – SELECT

### Utilización del \*

```
SELECT * FROM nbtarla;
```

Se utiliza el asterisco \* en la lista de selección para indicar '**todas las columnas de la tabla**'.

Tiene dos **ventajas**:

- Evitar nombrar las columnas una a una (es más corto).
- Si añadimos una columna nueva en la tabla, esta nueva columna saldrá sin tener que modificar la consulta.

Se puede combinar el \* con el nombre de una tabla (ej. oficinas.\*), pero esto se utiliza más cuando el origen de la consulta son dos o más tablas.

### Ejemplo: Listar todos los datos de las oficinas

```
SELECT * FROM Oficinas;
```

```
SELECT Oficinas.* FROM Oficinas;
```

### Limitar las columnas: proyección

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

Mediante la sentencia SELECT es posible hacer una proyección de una tabla, seleccionando las columnas de las que queremos obtener datos. En la sintaxis que hemos mostrado, la selección de columnas corresponde con la parte "expresion\_select".

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

```
SELECT Nombre, LimiteCredito FROM Clientes;
```

```
SELECT Nombre, Categoria, FecNacimiento FROM Empleados;
```

Las columnas se pueden especificar mediante su nombre **simple** (nbcot) o su nombre **cualificado** (nbtarla.nbcot, el nombre de la columna precedido del nombre de la tabla que contiene la columna y separados por un punto).

El nombre cualificado se puede emplear siempre que queramos y es obligatorio en algunos casos, por ejemplo, cuando utilizamos dos columnas de distintas tablas con el mismo nombre.

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

```
SELECT SIN(3.1416/2), 3+5, 7*4;
```

Vemos que podemos usar funciones, en este ejemplo hemos usado la función **SIN** para calcular el seno de  $\pi/2$ .

## Utilizar funciones

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

```
SELECT DISTINCT FecContrato,  
DATEDIFF(CURRENT_DATE(), FecContrato) AS "Días contratado"  
FROM Empleados;
```

Lista el nombre, oficina, y fecha de contrato de todos los empleados.

```
SELECT Nombre, Oficina, fecContrato FROM Empleados;
```

Lista una tarifa de productos

```
SELECT IdFabricante, IdProducto, Descripcion, PrecioCompra  
FROM Productos;
```

## Valores de columnas calculadas

MySQL permite calcular valores de salida mediante los resultados de expresiones. Las expresiones pueden ser simples y complejas.

Además de las columnas que provienen directamente de la tabla origen, una consulta SQL puede incluir **columnas calculadas** cuyos valores se calculan a partir de los valores de los datos almacenados.

Para solicitar una columna calculada, se especifica en la lista de selección una **expresión** en vez de un nombre de columna. La expresión puede contener sumas, restas, multiplicaciones y divisiones, concatenación & , paréntesis y también funciones predefinidas).

## Ejemplos:

Lista la ciudad, región y el superavit de cada oficina

```
SELECT Region, (Ventas-Objetivo) AS superavit FROM Oficinas;
```

De cada producto obtiene su fabricante, idproducto, su descripción y el valor del inventario

```
SELECT IdFabricante, IdProducto, Descripcion, (Existencias *  
PrecioCompra) AS Valoración FROM Productos;
```

Lista el nombre, mes y año del contrato de cada empleado.

```
SELECT Nombre, MONTH(FecContrato), MONTHNAME(FecContrato),  
YEAR(FecContrato) FROM Empleados;
```

La función **MONTH()** devuelve el mes numérico de una fecha

La función **MONTHNAME()** devuelve el nombre del mes de una fecha

La función **YEAR()** devuelve el año de una fecha

Listar las ventas en cada oficina con el formato: 22 tiene ventas de 186,042.00 €

```
SELECT CodOficina, 'tiene ventas de ', Ventas FROM Oficinas;
```

## Se puede dar formato a los valores numéricos

```
SELECT 17, FORMAT(SQRT(3*3*4*4),0), FORMAT(SQRT(3*3*4*4),2);
```

### Consulta que concatena dos cadenas (nombre, titulo)

```
SELECT CONCAT(Nombre, ' ',Categoria) FROM Empleados;
```

*Cuando utilizamos una columna calculada, la expresión se convierte en el nombre del encabezado de la columna que se muestra en el resultado. Para cambiar esto podemos asignar un nombre diferente a la columna empleando un alias.*

```
SELECT Descripcion, PrecioCompra, PrecioCompra *0.21 AS IVA,  
PrecioCompra + PrecioCompra *0.21 AS Importe  
FROM Productos;
```

## Alias

Se puede asignar un alias a cualquiera de las expresiones **SELECT**. Esto se puede hacer usando la palabra **AS**, aunque esta palabra es opcional:

```
SELECT Nombre AS Cliente, CodRepresentante AS Representante FROM  
Clientes;
```

Se puede 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**. Si el alias lleva espacios en blanco hay que encerrarlo entre comillas.

```
SELECT Nombre Empleado, fecContrato "Fecha contrato" FROM Empleados;  
SELECT IdFabricante AS Fabricante, idproducto AS Producto,  
Descripcion FROM Productos;
```

Como título de la primera columna aparecerá fabricante en vez de idfab

Se pueden usar los alias en otras cláusulas como **ORDER BY**.

```
SELECT Nombre AS Cliente, CodRepresentante AS Representante FROM  
Clientes  
ORDER BY Cliente;
```

## 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, si ejecutamos la siguiente consulta vemos que hay títulos que se repiten: Representante, etc.

```
SELECT Categoria FROM Empleados;
```

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:

```
SELECT DISTINCT Categoria FROM Empleados;
```

Solo aparecen los títulos una vez.

### Mostrar filas formateadas. FORMAT(X, D)

Formatea el número según la plantilla '#,###,###.##', redondeando a D decimales, y devuelve el resultado como una cadena. Si D es 0, el resultado no tendrá punto decimal ni parte fraccionaria:

```
SELECT FORMAT(12332.123456, 4) AS Formato;
```

```
SELECT FORMAT(12332.1,4) AS Formato;
```

```
SELECT FORMAT(12332.2,0) AS Formato;
```

```
SELECT Descripcion, PrecioCompra, PrecioCompra *0.21 AS IVA,  
FORMAT(PrecioCompra + PrecioCompra *0.21, 1) AS 'Precio con IVA'  
FROM Productos;
```

### Ordenar resultados

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

ORDER BY nbcolumna   Númerocolumnas ASC   DESC, .....
---

Con esta cláusula se altera el orden de visualización de las filas de la tabla pero en ningún caso se modifica el orden de las filas dentro de la tabla. La tabla no se modifica.

Podemos indicar la columna por la que queremos ordenar utilizando su **nombre de columna** (nbcolumna) o utilizando su **número de orden** que ocupa en la lista de selección (Nºcolumna).

### Ejemplos:

```
SELECT Nombre, Oficina, fecContrato FROM Empleados ORDER BY Oficina;
```

es equivalente a

```
SELECT Nombre, Oficina, fecContrato FROM Empleados ORDER BY 2;
```

Por defecto el **orden** será **ascendente (ASC)** (de menor a mayor si el campo es numérico, por orden alfabético si el campo es de tipo texto, de anterior a posterior si el campo es de tipo fecha/hora, etc...

### Ejemplos:

#### Obtiene un listado alfabético de los empleados

```
SELECT Nombre, CodEmpleado, Oficina FROM Empleados ORDER BY nombre;
```

#### Obtiene un listado de los empleados por orden de antigüedad en la empresa (los de más antigüedad aparecen primero).

```
SELECT Nombre, CodEmpleado, Contrato FROM Empleados ORDER BY  
Contrato;
```

**Obtiene un listado de los empleados ordenados por volumen de ventas sacando los de menores ventas primero**

```
SELECT Nombre, CodEmpleado, Sueldo FROM Empleados ORDER BY Sueldo;
```

Si queremos podemos alterar ese orden utilizando la cláusula **DESC** (DESCendente), en este caso el orden será el inverso al ASC.

**Ejemplos:**

**Obtiene un listado de los empleados por orden de antigüedad en la empresa empezando por los más recientemente incorporados.**

```
SELECT Nombre, CodEmpleado, fecContrato FROM Empleados ORDER BY fecContrato DESC
```

**Obtiene un listado de los empleados ordenados por volumen de ventas sacando primero los de mayores ventas.**

```
SELECT Nombre, CodEmpleado, Sueldo FROM Empleados ORDER BY Sueldo DESC;
```

También podemos **ordenar** por **varias columnas**, en este caso se indican las columnas separadas por comas.

Se ordenan las filas por la primera columna de ordenación, para un mismo valor de la primera columna, se ordenan por la segunda columna, y así sucesivamente.

La cláusula **DESC** o **ASC** se puede indicar para cada columna y así utilizar una ordenación distinta para cada columna. Por ejemplo ascendente por la primera columna y dentro de la primera columna, descendente por la segunda columna.

**Ejemplos:**

**Muestra las ventas de cada oficina, ordenadas por orden alfabético de región y dentro de cada región por ciudad.**

```
SELECT Region, Ventas FROM Oficinas ORDER BY Region, Ventas;
```

**Lista las oficinas clasificadas por región y dentro de cada región por superavit de modo que las de mayor superavit aparezcan las primeras.**

```
SELECT Region, (Ventas - Objetivo) AS Superavit FROM Oficinas ORDER BY 2 DESC;
```

```
SELECT * FROM Clientes ORDER BY Nombre;
```

```
SELECT * FROM Clientes ORDER BY Nombre ASC;
```

Los valores NULL en una columna aparecen al principio cuando se especifica un orden ascendente o al final cuando el sentido es descendente.

**Obtener un listado de los empleados ordenado por la oficina en la que trabajan**

```
SELECT * FROM Empleados ORDER BY Oficina;
```

Los valores nulos aparecen al principio

```
SELECT * FROM Empleados ORDER BY Oficina DESC;
```

Los valores nulos aparecen al final

## Limitar el número de filas de salida

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

```
SELECT * FROM Empleados LIMIT 3;
```

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 cláusula **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:

```
SELECT * FROM Empleados LIMIT 0,2;
```

```
SELECT * FROM Empleados LIMIT 2,2;
```

```
SELECT * FROM Empleados LIMIT 4,2;
```

```
SELECT * FROM Empleados LIMIT 6,2;
```

### Consulta que selecciona las 5 empleados más antiguos en la empresa

```
SELECT * FROM Empleados ORDER BY fecContrato LIMIT 5;
```

Si utilizamos una ordenación descendente DESC, obtendremos los 5 empleados que llevan menos tiempo trabajando.

```
SELECT * FROM Empleados ORDER BY fecContrato DESC LIMIT 5;
```

Para obtener un número de registros seleccionados aleatoriamente en una tabla utilizamos ORDER BY RAND() en conjunción con LIMIT.

```
SELECT * FROM Empleados ORDER BY RAND() LIMIT 5;
```

```
SELECT * FROM Empleados ORDER BY RAND() LIMIT 1;
```

Las funciones de agregado se pueden combinar con ORDER BY y con LIMIT.

### Consulta que selecciona las 4 pedidos con menos valor

```
SELECT CodPedido, SUM(Cantidad * PrecioVenta) AS Total FROM  
LineasPedido GROUP BY CodPedido ORDER BY Total LIMIT 4;
```

## La tabla origen – FROM

Con la cláusula **FROM** indicamos **en qué tabla** tiene que **buscar la información**. En este capítulo de consultas simples el resultado se obtiene de una única tabla. La sintaxis de la cláusula es:

### **FROM especificación de tabla**

Una especificación de tabla puede ser el nombre de una consulta guardada (las que aparecen en la ventana de base de datos), o el nombre de una tabla.

FROM nbtTabla AS aliastabla
-----------------------------

**Aliastabla** es un nombre de **alias**, es como un **segundo nombre** que asignamos a la **tabla**, si en una consulta definimos un alias para la tabla, esta se deberá nombrar utilizando ese nombre y no su nombre real, además **ese nombre sólo** es **válido en la consulta** donde se define. El alias se suele emplear en consultas basadas en más de una tabla que veremos en el tema siguiente. La palabra **AS** que se puede poner delante del nombre de alias es opcional y es el valor por defecto por lo que no tienen ningún efecto.

### Ejemplo:

```
SELECT * FROM Oficinas Ofi;
```

es equivalente a

```
SELECT * FROM Oficinas AS Ofi;
```

esta sentencia me indica que se van a buscar los datos en la tabla *oficinas* que queda renombrada en esta consulta con *ofi*.

En una **SELECT** podemos utilizar tablas que no están definidas en la base de datos (siempre que tengamos los permisos adecuados), si la tabla no está en la base de datos activa, debemos indicar en qué base de datos se encuentra **utilizando el nombre cualificado**.

```
FROM nbbasededatos.nbtabla AS aliastabla
```

Supongamos que la tabla empleados estuviese en otra base de datos llamada otra en la carpeta c:\mis documentos\, habría que indicarlo así:

```
SELECT * FROM Otra.Empleados AS Ofi;
```

### La cláusula WHERE

```
WHERE condición de selección
```

La cláusula **WHERE** selecciona únicamente las **filas** que **cumplan la condición de selección** especificada.

En la consulta sólo aparecerán las filas para las cuales la condición es verdadera (TRUE), los valores nulos (NULL) no se incluyen por lo tanto en las filas del resultado. La **condición de selección** puede ser cualquier **condición válida** o **combinación de condiciones** utilizando los operadores **NOT** (no) **AND** (y) y **OR** (ó).

### Ejemplos:

Lista el nombre de los empleados de la oficina 12.

```
SELECT Nombre FROM Empleados WHERE Oficina = 12;
```

Lista el nombre de los empleados de la oficina 12 y que hayan sido contratados después de 1980

```
SELECT Nombre FROM Empleados WHERE Oficina = 12 AND  
YEAR(fecContrato) > 1980;
```



## Condiciones de selección

Las **condiciones de selección** son las condiciones que pueden aparecer en la cláusula **WHERE**.

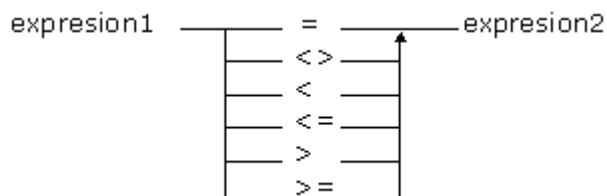
En SQL tenemos cinco condiciones básicas:

- el **test de comparación**
- el **test de rango**
- el **test de pertenencia a un conjunto**
- el **test de valor nulo**
- el **test de correspondencia con patrón**.

### El test de comparación.

Compara el valor de una expresión con el valor de otra.

La sintaxis es la siguiente:



### Lista las oficinas cuyas ventas superan su objetivo

```
SELECT codOficina, ciudad
FROM Oficinas
WHERE Ventas > Objetivo;
```

### Lista los empleados contratados antes del año 88 (cuya fecha de contrato sea anterior al 1 de enero de 1988).

```
SELECT CodEmpleado, Nombre, fecContrato
FROM Empleados
WHERE fecContrato < '1988/01/01';
```

### :: las fechas entre comillas simples o dobles deben estar con el formato año, mes, día.

Este ejemplo obtiene lo mismo que el anterior pero utiliza la función year(). Obtiene los empleados cuyo año de la fecha de contrato sea menor que 1988.

```
SELECT CodEmpleado, Nombre, fecContrato
FROM Empleados
WHERE YEAR(fecContrato) < 1988;
```

### Lista las oficinas cuyas ventas estén por debajo del 80% de su objetivo. Hay que utilizar siempre el punto decimal.

```
SELECT *
FROM Oficinas
WHERE Ventas < Objetivo * 0.8;
```


### Lista los productos del fabricante fea

```
SELECT *  
FROM Productos  
WHERE IdFabricante = "fea";
```

### Test de rango (BETWEEN).

Examina si el **valor** de la expresión está **comprendido entre** los **dos valores** definidos por exp1 y exp2.

Tiene la siguiente sintaxis:

expresion 

### Lista los empleados cuyos sueldos estén comprendidas entre 1.500 y 3.000 €

```
SELECT CodEmpleado, Nombre, Sueldo  
FROM Empleados  
WHERE Sueldo BETWEEN 1500 AND 3000;
```


### Obtenemos lo mismo que en el ejemplo anterior. Los paréntesis son opcionales.

```
SELECT CodEmpleado, Nombre, Sueldo  
FROM Empleados  
WHERE (Sueldo >= 1500) AND (Sueldo <= 3000);
```

### Test de pertenencia a conjunto (IN)

Examina si el **valor** de la expresión es uno de los valores **incluidos en la lista de valores**.

Tiene la siguiente sintaxis

Expresion 

### Lista los empleados de las oficinas 12, 14 y 16

```
SELECT CodEmpleado, Nombre, Oficina  
FROM Empleados  
WHERE Oficina IN ('12','14','16');
```

### Obtenemos lo mismo que en el ejemplo anterior. Los paréntesis son opcionales.

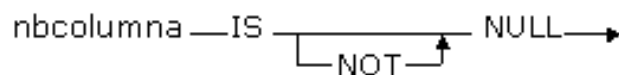
```
SELECT CodEmpleado, Nombre, Oficina  
FROM Empleados  
WHERE (Oficina = 12) OR (Oficina = 14) OR (Oficina = 16);
```

### Test de valor nulo (IS NULL)

Una condición de selección puede dar como resultado el valor verdadero TRUE, falso FALSE o nulo NULL.

Cuando una **columna** que interviene en una condición de selección **contiene** el **valor nulo**, el **resultado** de la condición no es verdadero ni falso, sino **nulo**, sea **cual sea el test** que se haya utilizado. Por eso si queremos listar las filas que tienen valor en una determinada columna, no podemos utilizar el test de comparación, la condición oficina = null devuelve el valor nulo sea cual sea el valor contenido en oficina. Si queremos preguntar si una columna contiene el valor nulo debemos utilizar un **test especial**, el test de valor nulo.

Tiene la siguiente sintaxis:



## Ejemplos:

Lista los empleados que no tienen una oficina asignada.

```

SELECT Nombre, Categoria, Oficina
FROM Empleados
WHERE Oficina IS NULL;
  
```

Lista los empleados asignados a alguna oficina (los que tienen un valor en la columna oficina).

```

SELECT Nombre, Categoria, Oficina
FROM Empleados
WHERE Oficina IS NOT NULL;
  
```

## Operador de igualdad con *NULL* seguro

El operador  $\lt=\gt$  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:

```
SELECT NULL = TRUE, NULL = 1, NULL = NULL;
```

NULL = TRUE	NULL = 1	NULL = NULL	
NULL	NULL	NULL	

El resultado de comparar NULL con un valor es NULL, y dos valores nulos también devuelven NULL, mientras que si utilizamos el operador de igualdad con NULL seguro el resultado es verdadero si comparamos dos valores NULL y falso si comparamos NULL con otro valor.

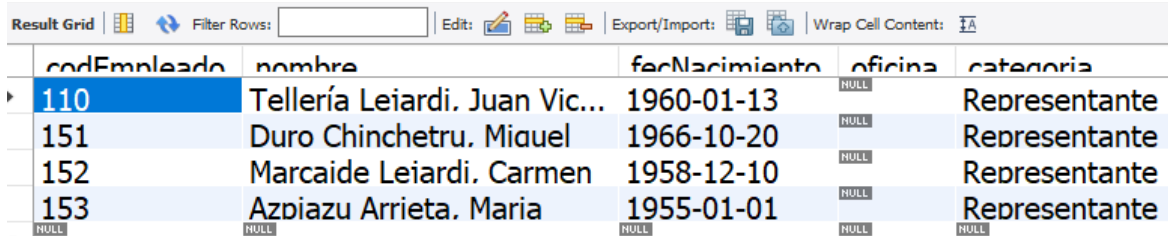
```
SELECT NULL <=> TRUE, NULL <=> 1, NULL <=> NULL;
```

NULL <=> TRUE	NULL <=> 1	NULL <=> NULL	
0	0	1	

## Ejemplo

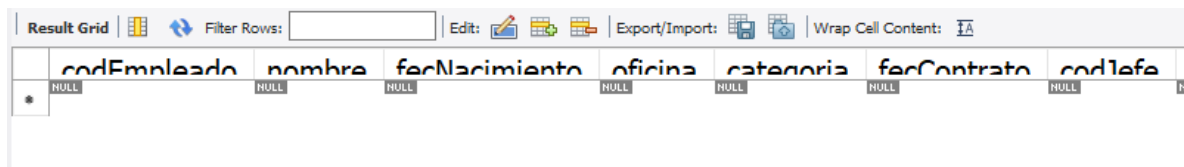
Listar los representantes que no tengan oficina asignada.

```
SELECT * FROM empleados e
WHERE Oficina IS NULL AND Categoria = 'Representante';
```



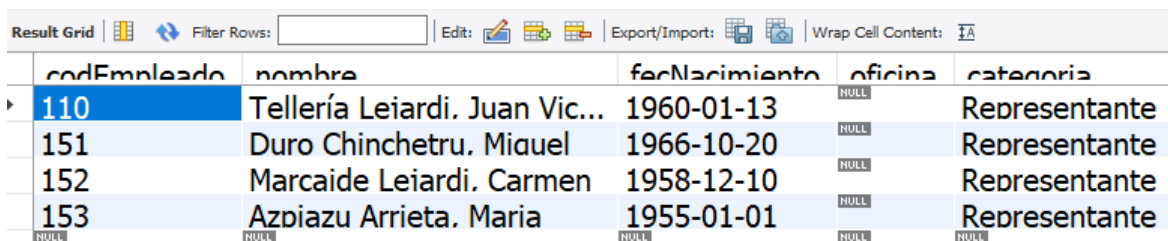
codEmpleado	nombre	fecNacimiento	oficina	categoria
110	Tellería Leiardi, Juan Vic...	1960-01-13	NULL	Representante
151	Duro Chinchetru, Miquel	1966-10-20	NULL	Representante
152	Marcaide Leiardi, Carmen	1958-12-10	NULL	Representante
153	Azpiazu Arrieta, Maria	1955-01-01	NULL	Representante

```
SELECT * FROM empleados e
WHERE Oficina = NULL AND Categoria = 'Representante';
```



codEmpleado	nombre	fecNacimiento	oficina	categoria	fecContrato	codIefe
110	Tellería Leiardi, Juan Vic...	1960-01-13	NULL	Representante		
151	Duro Chinchetru, Miquel	1966-10-20	NULL	Representante		
152	Marcaide Leiardi, Carmen	1958-12-10	NULL	Representante		
153	Azpiazu Arrieta, Maria	1955-01-01	NULL	Representante		

```
SELECT * FROM empleados e
WHERE Oficina <=> NULL AND Categoria = 'Representante';
```



codEmpleado	nombre	fecNacimiento	oficina	categoria
110	Tellería Leiardi, Juan Vic...	1960-01-13	NULL	Representante
151	Duro Chinchetru, Miquel	1966-10-20	NULL	Representante
152	Marcaide Leiardi, Carmen	1958-12-10	NULL	Representante
153	Azpiazu Arrieta, Maria	1955-01-01	NULL	Representante

## Test de correspondencia con patrón (LIKE)

Se utiliza cuando queremos **utilizar caracteres comodines** para formar el valor con el comparar.

Tiene la siguiente sintaxis:

nbcolumna NOT LIKE patron →

Los comodines más usados son los siguientes:

- **?** representa un carácter cualquiera
- **%** representa cero o más caracteres
- **\_** representa un dígito cualquiera (0-9)

## Ejemplos:

Listar los empleados cuyo nombre empiece por Luis (Luis seguido de cero o más caracteres).

```
SELECT CodEmpleado, Nombre
FROM Empleados
WHERE Nombre LIKE 'Luis%';
```

```
SELECT CodEmpleado, Nombre
FROM Empleados
WHERE Nombre LIKE 'Luis %';
```

**Lista los empleados cuyo nombre contiene Luis, en este caso también saldría los empleados José Luis (cero o más caracteres seguidos de LUIS y seguido de cero o más caracteres).**

```
SELECT CodEmpleado, Nombre
FROM Empleados
WHERE Nombre LIKE '%Luis%';
```

**Lista los empleados cuyo nombre contenga una a como tercera letra (dos caracteres, la letra a, y cero o más caracteres**

```
SELECT CodEmpleado, Nombre
FROM Empleados
WHERE nombre LIKE '__a%';
```

### **Operadores REGEXP y RLIKE**

La sintaxis es:

```
<expresión> RLIKE <patrón>
```

```
<expresión> REGEXP <patrón>
```

Al igual que LIKE el operador REGEXP (y su equivalente RLIKE), comparan una expresión con un patrón, pero en este caso, el patrón puede ser una expresión regular extendida.

El valor de retorno es verdadero (1) si la expresión coincide con el patrón, en caso contrario devuelve un valor falso (0). Tanto si la expresión como el patrón son nulos, el resultado es NULL.

El patrón no tiene que ser necesariamente una cadena, puede ser una expresión o una columna de una tabla.

**Seleccionar todos los empleados cuyo nombre este comprendido entre la D y la M.**

```
SELECT CodEmpleado, Nombre
FROM Empleados
WHERE Nombre REGEXP '^[D-M]';
```

```
SELECT CodEmpleado, Nombre
FROM Empleados
WHERE Nombre RLIKE '^[L-M]';
```

### **Operadores NOT REGEXP y NOT RLIKE**

La sintaxis es:

```
<expresión> NOT RLIKE <patrón>
```

```
<expresión> NOT REGEXP <patrón>
```

Que equivalen a:

NOT (<expresión> REGEXP <patrón>)
-----------------------------------

Seleccionar todos los empleados cuyo nombre no este comprendido entre la D y la M.

```
SELECT CodEmpleado, Nombre  
FROM Empleados  
WHERE Nombre NOT REGEXP '^[D-M]';
```

```
SELECT CodEmpleado, Nombre  
FROM Empleados  
WHERE Nombre NOT RLIKE '^[L-M]';
```