**Test de correspondencia con patrón LIKE**

Se utiliza cuando queremos comparar el valor de una columna con un patrón en el que se utilice caracteres comodines.

<expression>  [NOT] LIKE <patron>  [ESCAPE 'car\_escape']

Con expresión indicamos el valor a comparar (normalmente será el nombre de una columna) y patrón es la cadena que se busca. El patrón es de tipo texto y tiene que escribirse entre comillas simples. Dentro del patrón podemos utilizar los siguientes comodines:

%    representa cualquier cadena de cero o más caracteres.

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE ‘An%’;

|  |  |
| --- | --- |
| **numemp** | **nombre** |
| 101 | Antonio Viguer |
| 108 | Ana Bustamante |

Obtiene todos los nombres que empiecen por ***An***.

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE ‘%z’;

|  |  |
| --- | --- |
| **numemp** | **nombre** |
| 104 | José González |
| 107 | José González |

Obtiene los nombres que acaban en ***z***.

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE ‘%on%’;

|  |  |
| --- | --- |
| **numemp** | **nombre** |
| 101 | Antonio Viguer |
| 104 | José González |
| 106 | Luis Antonio |

Obtiene los nombres que contienen ***on***.

\_     representa cualquier carácter (sólo uno).

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE '\_\_a%';

|  |  |
| --- | --- |
| **numemp** | **nombre** |
| 103 | Juan Rovira |
| 108 | Ana Bustamante |
| 110 | Juan Victor |

Obtiene los nombres cuya tercera letra sea una ***a*** (en el patrón tenemos dos caracteres subrayado).

[  ] sirve para indicar un carácter cualquiera perteneciente al conjunto indicando.  
       El conjunto se indica enumerando los caracteres o indicando un intervalo.

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE '[a-d]%';

Obtiene los nombres que empiezan por cualquier letra de la ***a*** a la ***d***.

Es equivalente a escribir:

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE '[abcd]%';

[^] significa cualquier carácter individual que no se encuentre en el conjunto.

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE '[^abcd]%';

Y

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE '[^a-d]%';

Obtienen los nombres que no empiecen por ***a***, ***b***, ***c*** ni ***d***.

Es importante tener en cuenta que dentro del patrón el espacio en blanco es considerado como un carácter más, si colocamos dos espacios en el patrón, se buscarán dos espacios en el campo.

Si queremos incluir en el patrón uno de los caracteres comodines y que no sea interpretado como un comodín, sino como un carácter normal, lo tenemos que encerrar entre corchetes o utilizar un carácter de escape.

[ESCAPE 'car\_escape']

La cláusula ESCAPE es opcional y permite definir un carácter de escape.

Un carácter de escape es un carácter que se coloca delante de un carácter comodín para indicar que el comodín no debe interpretarse como tal, sino como un carácter normal.

Por ejemplo queremos buscar los nombres compuestos que incluyen un subrayado. En este caso tenemos que poner el carácter \_ como un carácter normal no como un comodín, así que lo escribiremos así:

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE '%[\_]%';

O bien,

SELECT numemp,nombre

FROM empleados

WHERE nombre LIKE '%!\_%' ESCAPE '!';

# **Obtener los 3 primeros registros en MySQL**

En este tutorial aprenderemos cómo obtener los tres primeros o últimos registros ordenados de una tabla mediante una consulta en MySQL.

## **Obtener los 3 primeros registros de mayor a menor**

**SELECT \* FROM ‘nombre\_tabla’ ORDER BY ‘campo\_ordenar’ DESC LIMIT 3**

Ejemplo:

Si tenemos una tabla llamada **datos\_usuarios** con los siguientes campos: **id, nombre, edad**

Si ejecutamos la consulta

**SELECT \* FROM datos\_usuarios ORDER BY edad DESC LIMIT 3**

obtenemos los tres usuarios más mayores ordenados de mayor a menor.

La consulta primero ordena los registros por el campo «**edad**» de mayor a menor **(desc)**,  y luego devuelve los tres primeros **(limit 3)**

## **Obtener los 3 primeros de menor a mayor**

**SELECT \* FROM ‘nombre\_tabla’ ORDER BY ‘campo\_ordenar’ ASC LIMIT 3**

## **Obtener los 5 primeros registros de mayor a menor**

**SELECT \* FROM ‘nombre\_tabla’ ORDER BY ‘campo\_ordenar’ DESC LIMIT 5**

# **¿Cómo funciona INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN?**

Aprende (o repasa) cómo funcionan las consultas J0IN y sus variantes en SQL.

Los conceptos que vamos a revisar hoy, aplican en general para las bases de datos relacionales.

Sin embargo, por esta vez, usaremos de forma específica SQL Server.

## **Antes de iniciar: ¿Qué son las consultas JOIN y por qué son importantes?**

Nosotros podemos consultar el contenido de una tabla muy puntualmente:

SELECT \* FROM myTable

Sin embargo, generalmente necesitamos combinar información de distintas tablas.

Obtener datos específicos de distintas tablas, y presentar la información de una mejor manera, resulta de ayuda para la toma de decisiones.

Cuando trabajamos con bases de datos relacionales:

* Una tabla fuerte representa una entidad independiente, y una tabla débil representa una entidad que depende de otra.
* Estas relaciones (dependencias entre tablas) consisten en columnas que tienen un mismo tipo de dato.

Entonces, la cláusula JOIN **nos permite asociar 2 o más tablas, en base a una columna** que tengan en común.

## **Ejemplo: Empleados y Departamentos**

A continuación vamos a analizar la cláusula JOIN y sus variantes.

Pero primero, empecemos definiendo 2 tablas, que serán la base para nuestros ejemplos posteriores.

Por un lado vamos a tener una tabla Empleados (que almacenará una lista de empleados y el id del departamento al que pertenecen):

| **Nombre** | **DepartamentoId** |
| --- | --- |
| Rafferty | 31 |
| Jones | 33 |
| Heisenberg | 33 |
| Robinson | 34 |
| Smith | 34 |
| Williams | NULL |

Y por otro lado, una tabla Departamentos (con la lista de departamentos que existen en la empresa).

| **Id** | **Nombre** |
| --- | --- |
| 31 | Sales |
| 33 | Engineering |
| 34 | Clerical |
| 35 | Marketing |

Entonces, si quieres seguir el tutorial, puedes ejecutar el siguiente script SQL para dar origen a las tablas y sus datos correspondientes:

CREATE DATABASE TestJoin

GO

USE TestJoin

GO

CREATE TABLE Departamentos

(

Id int,

Nombre varchar(20)

);

CREATE TABLE Empleados

(

Nombre varchar(20),

DepartamentoId int

);

INSERT INTO Departamentos VALUES(31, 'Sales');

INSERT INTO Departamentos VALUES(33, 'Engineering');

INSERT INTO Departamentos VALUES(34, 'Clerical');

INSERT INTO Departamentos VALUES(35, 'Marketing');

INSERT INTO Empleados VALUES('Rafferty', 31);

INSERT INTO Empleados VALUES('Jones', 33);

INSERT INTO Empleados VALUES('Heisenberg', 33);

INSERT INTO Empleados VALUES('Robinson', 34);

INSERT INTO Empleados VALUES('Smith', 34);

INSERT INTO Empleados VALUES('Williams', NULL);

En este script SQL, creamos una base de datos llamada TestJoin, y dentro de ella las 2 tablas mencionadas.

Ten en cuenta que podrías usar otro nombre para la base de datos, si así lo prefieres.

Hasta este punto, puedes confirmar que todo haya salido bien, consultando las 2 tablas correspondientes:

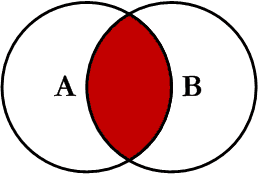
SELECT \* FROM Empleados;

SELECT \* FROM Departamentos;

### Cláusula INNER JOIN

Lo más usual, lo primero que se suele aprender, es el uso de INNER JOIN, o generalmente abreviado como JOIN.

Esta cláusula busca coincidencias entre 2 tablas, en función a una columna que tienen en común. De tal modo que **sólo la intersección se mostrará en los resultados**.



Por ejemplo, si queremos listar a los empleados e indicar el nombre del departamento al que pertenecen, podemos hacer lo siguiente:

SELECT \*

FROM Empleados E

JOIN Departamentos D

ON E.DepartamentoId = D.Id

Con esto, nuestro resultado será:

| **Nombre** | **DepartmentoId** | **Id** | **Nombre** |
| --- | --- | --- | --- |
| Rafferty | 31 | 31 | Sales |
| Jones | 33 | 33 | Engineering |
| Heisenberg | 33 | 33 | Engineering |
| Robinson | 34 | 34 | Clerical |
| Smith | 34 | 34 | Clerical |

Y apartir de aquí podemos notar lo siguiente:

* El empleado "Williams" no aparece en los resultados, ya que no pertenece a ningún departamento existente.
* El departamento "Marketing" tampoco aparece, ya que ningún empleado pertenece a dicho departamento.

¿Por qué ocurre esto? Porque JOIN muestra como resultado la intersección de ambas tablas.

También hay que tener en cuenta que, en los resultados vemos 4 columnas. Las 2 primeras se corresponden con la tabla Empleados y las últimas con Departamentos.

Esto ocurre porque estamos seleccionando todas las columnas con un \*.

Si queremos, podemos ser específicos y seleccionar sólo 2 columnas (además de asignarles un alias):

SELECT

E.Nombre as 'Empleado',

D.Nombre as 'Departamento'

FROM Empleados E

JOIN Departamentos D

ON E.DepartamentoId = D.Id

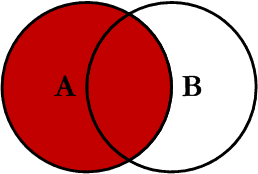
Y así obtener:

| **Empleado** | **Departamento** |
| --- | --- |
| Rafferty | Sales |
| Jones | Engineering |
| Heisenberg | Engineering |
| Robinson | Clerical |
| Smith | Clerical |

### Cláusula LEFT JOIN

A diferencia de un INNER JOIN, donde se busca una intersección respetada por ambas tablas, con LEFT JOIN damos prioridad a la tabla de la izquierda, y buscamos en la tabla derecha.

Si no existe ninguna coincidencia para alguna de las filas de la tabla de la izquierda, de igual forma **todos los resultados de la primera tabla se muestran**.



He aquí una consulta de ejemplo:

SELECT

E.Nombre as 'Empleado',

D.Nombre as 'Departamento'

FROM Empleados E

LEFT JOIN Departamentos D

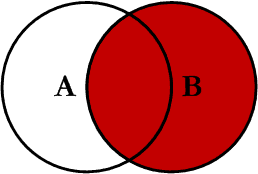
ON E.DepartamentoId = D.Id

* La tabla Empleados es la primera tabla en aparecer en la consulta (en el FROM), por lo tanto ésta es la tabla LEFT (izquierda), y todas sus filas se mostrarán en los resultados.
* La tabla Departamentos es la tabla de la derecha (aparece luego del LEFT JOIN). Por lo tanto, si se encuentran coincidencias, se mostrarán los valores correspondientes, pero sino, aparecerá NULL en los resultados.

| **Empleado** | **Departamento** |
| --- | --- |
| Rafferty | Sales |
| Jones | Engineering |
| Heisenberg | Engineering |
| Robinson | Clerical |
| Smith | Clerical |
| Williams | NULL |

### Cláusula RIGHT JOIN

En el caso de RIGHT JOIN la situación es muy similar, pero aquí se da prioridad a la tabla de la derecha.



De tal modo que si usamos la siguiente consulta, estaremos **mostrando todas las filas de la tabla de la derecha**:

SELECT

E.Nombre as 'Empleado',

D.Nombre as 'Departamento'

FROM Empleados E

RIGHT JOIN Departamentos D

ON E.DepartamentoId = D.Id

La tabla de la izquierda es Empleados, mientras que Departamentos es la tabla de la derecha.

La tabla asociada al FROM será siempre la tabla LEFT, y la tabla que viene después del JOIN será la tabla RIGHT.

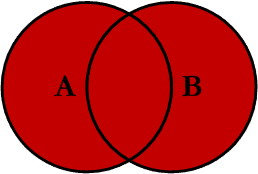
Entonces el resultado mostrará todos los departamentos al menos 1 vez.

Y si no hay ningún empleado trabajando en un departamento determinado, se mostrará NULL. Pero el departamento aparecerá de igual forma.

| **Empleado** | **Departamento** |
| --- | --- |
| Rafferty | Sales |
| Jones | Engineering |
| Heisenberg | Engineering |
| Robinson | Clerical |
| Smith | Clerical |
| NULL | Marketing |

### Cláusula FULL JOIN

Mientras que LEFT JOIN muestra todas las filas de la tabla izquierda, y RIGHT JOIN muestra todas las correspondientes a la tabla derecha, FULL OUTER JOIN (o simplemente FULL JOIN) se encarga de mostrar todas las filas de ambas tablas, sin importar que no existan coincidencias (usará NULL como un valor por defecto para dichos casos).



Para nuestro ejemplo, ocurre lo siguiente:

SELECT

E.Nombre as 'Empleado',

D.Nombre as 'Departamento'

FROM Empleados E

FULL JOIN Departamentos D

ON E.DepartamentoId = D.Id

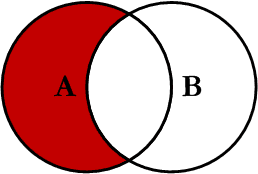
Se muestra el empleado "Williams" a pesasr que no está asignado a ningún departamento, y se muestra el departamento de "Marketing" a pesar que aún nadie está trabajando allí:

| **Empleado** | **Departamento** |
| --- | --- |
| Rafferty | Sales |
| Jones | Engineering |
| Heisenberg | Engineering |
| Robinson | Clerical |
| Smith | Clerical |
| Williams | NULL |
| NULL | Marketing |

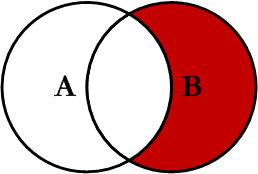
## **Más variantes**

Si prestamos atención a los diagramas de Venn, vamos a notar que es posible formar incluso más combinaciones, al momento de seleccionar datos.

Por ejemplo, tenemos el siguiente caso, conocido como Left Excluding JOIN:



Y de igual manera Right Excluding JOIN:



Estas combinaciones son posibles de lograr si añadimos algunas condiciones a nuestras consultas, haciendo uso de la cláusula WHERE.

Por ejemplo, siguiendo el ejemplo que estamos viendo (donde Empleados es la tabla izquierda y Departamentos la tabla derecha):

* Left Excluding JOIN nos permitirá obtener la lista de empleados que aún no han sido asignados a ningún departamento de trabajo.
* Mientras que Right Excluding JOIN nos mostrará la lista de departamentos que no tienen a ningún trabajador asociado.

**Las consultas SQL son las mismas que usamos antes, pero añadiendo condiciones adicionales.**

Si quieres aprender más sobre las consultas JOIN puedes también ver el siguiente [video práctico](https://www.youtube.com/watch?v=eNikUI0Y8y8):