QUERIES ML

DigitalHouse>
Coding School

Certified Tech Developer
The Ultimate Degree

SELECT - Cómo usarlo

Toda consulta a la base de datos va a empezar con la palabra **SELECT**.

Su funcionalidad es la de realizar consultas sobre **una** o **varias columnas** de una tabla.

Para especificar sobre qué tabla queremos realizar esa consulta usamos la palabra **FROM** seguida del nombre de la tabla.

```sql
SELECT nombre_columna, nombre_columna, ...
FROM nombre_tabla;
```

Ejemplo - Tabla Peliculas

id	título	rating	fecha_estreno	país
1001	Pulp Fiction	9.8	1995-02-16	Estados Unidos
1002	Kill Bill	9.5	2003-11-27	Estados Unidos

De esta tabla completa, para conocer solamente los títulos y ratings de las películas guardadas en la tabla **películas,** podríamos hacerlo ejecutando la siguiente consulta:

SQL

```sql
SELECT id, titulo, rating
FROM peliculas;
```

Where

La funcionalidad del **WHERE** es la de condicionar y filtrar las consultas **SELECT** que se realizan a una base de datos.

```SQL
SELECT nombre_columna_1, nombre_columna_2, ...
FROM nombre_tabla
WHERE condicion;
```

Teniendo una tabla **usuarios**, podríamos consultar nombre y edad, filtrando con un **WHERE** solamente los usuarios **mayores de 17 años** de la siguiente manera:

```SQL
SELECT nombre, edad
FROM usuarios
WHERE edad > 17;
```

Operadores

=	Igual a
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
<>	Diferente a
!=	Diferente a

Operadores

IS NULL	Es nulo
BETWEEN	Entre dos valores
IN	Lista de valores
LIKE	Se ajusta a...

Queries de ejemplo

```sql
SELECT nombre, edad
FROM usuarios
WHERE edad > 17;
```

```sql
SELECT *
FROM movies
WHERE title LIKE 'Avatar';
```

Queries de ejemplo

SQL

```sql
SELECT *
FROM movies
      WHERE awards >= 3
      AND awards < 8;
```

SQL

```sql
SELECT *
FROM movies
      WHERE awards = 2
      OR awards = 6;
```

Query de ejemplo

```sql
DELETE FROM usuarios
WHERE id = 2;
```

Si en esta query quitáramos el WHERE...
¡Borraríamos toda la tabla!

Order By

ORDER BY se utiliza para ordenar los resultados de una consulta **según el valor de la columna especificada**. Por defecto, se ordena de forma ascendente *(ASC)* según los valores de la columna. También se puede ordenar de manera descendente (DESC) aclarándolo en la consulta.

```sql
SELECT nombre_columna1, nombre_columna2
FROM tabla
WHERE condicion
ORDER BY nombre_columna1;
```

Query de ejemplo

Teniendo una tabla **usuarios**, podría consultar los nombres, filtrar con un **WHERE** sólo los usuarios **mayores de 21 años** y ordenarlos de forma descendente tomando como referencia la columna nombre.

SQL

```sql
SELECT nombre, edad
FROM usuarios
WHERE edad > 21
ORDER BY nombre DESC;
```

BETWEEN y LIKE

DigitalHouse>
Coding School

Certified Tech Developer
The Ultimate Degree

BETWEEN

Cuando necesitamos obtener valores **dentro de un rango**, usamos el operador BETWEEN.

- BETWEEN **incluye** los **extremos**.
- BETWEEN funciona con **números**, **textos** y **fechas**.
- Se usa como un filtro de un WHERE.

Por ejemplo, coloquialmente:

- Dados los números: 4, 7, 2, 9, 1

Si hiciéramos un BETWEEN entre 2 y 7 devolvería 4, 7, 2 *(excluye el 9 y el 1, e incluye el 2).*

Query de ejemplo

Con la siguiente consulta estaríamos seleccionando **nombre** y **edad** de la tabla **alumnos** sólo cuando las edades estén **entre** 6 y 12.

```sql
SELECT nombre, edad
FROM alumnos
WHERE edad BETWEEN 6 AND 12;
```

LIKE

Cuando hacemos un filtro con un **WHERE**, podemos especificar un patrón de búsqueda que nos permita especificar algo concreto que queremos encontrar en los registros. Eso lo logramos utilizando **comodines** *(wildcards)*.

Por ejemplo, podríamos querer buscar:

- Los nombres que tengan la letra 'a' como segundo carácter.
- Las direcciones postales que incluyan la calle 'Monroe'.
- Los clientes que empiecen con 'Los' y terminen con 's'.

COMODÍN %

Es un sustituto que representa **cero**, **uno**, o **varios** caracteres.

COMODÍN _

Es un sustituto para **un solo** carácter.

Queries de ejemplo

```sql
SELECT nombre
FROM usuarios
WHERE nombre LIKE '_a%';
```

Devuelve aquellos nombres que tengan la letra 'a' como segundo carácter.

```sql
SELECT nombre
FROM usuarios
WHERE direccion LIKE '%Monroe%';
```

Devuelve las direcciones de los usuarios que incluyan la calle 'Monroe'.

Queries de ejemplo

```sql
SELECT nombre
FROM clientes
WHERE nombre LIKE 'Los%s';
```

Devuelve los clientes que empiecen con 'Los' y terminen con 's'.

LIMIT y OFFSET

DigitalHouse>
Coding School

Certified Tech Developer
The Ultimate Degree

Limit

Su funcionalidad es la de **limitar el número de filas** (registros/resultados) devueltas en las consultas SELECT. También establece el **número máximo** de registros a eliminar con DELETE.

```SQL
SELECT nombre_columna1, nombre_columna2
FROM nombre_tabla
LIMIT cantidad_de_registros;
```

Query de ejemplo

Teniendo una tabla **peliculas**, podríamos armar un top 10 con las películas que tengan más de 4 premios usando un **LIMIT** en la siguiente consulta:

SQL

```sql
SELECT *
FROM peliculas
WHERE premios > 4
LIMIT 10;
```

Offset

- En un escenario en donde hacemos una consulta de todas las películas de la base de datos, la misma nos devolvería muchos registros. Usando un **LIMIT** podríamos aclarar un límite de 20.
- *¿Pero cómo haríamos si quisiéramos recuperar sólo 20 películas pero salteando las primeras 10 de la tabla?*
- **OFFSET** nos permite especificar a partir de qué fila comenzar la recuperación de los datos solicitados.

{código}

```
SELECT id, nombre, apellido
FROM alumnos
LIMIT 20
OFFSET 20;
```

{código}

```
SELECT id, nombre, apellido
FROM alumnos
LIMIT 20
OFFSET 20;
```

Seleccionamos las columnas id, nombre y apellido.

{código}

```
SELECT id, nombre, apellido
FROM alumnos
LIMIT 20
OFFSET 20;
```

de la tabla alumnos.

{código}

```
SELECT id, nombre, apellido
FROM alumnos
LIMIT 20
OFFSET 20;
```

Limitamos los registros de la tabla resultante a **20** registros.

{código}

```
SELECT id, nombre, apellido
FROM alumnos
LIMIT 20
OFFSET 20;
```

Desplazamos los resultados 20 posiciones para que se muestre desde la posición **21**.

Alias

DigitalHouse>
Coding School

Certified Tech Developer
The Ultimate Degree

Alias

Los **alias** se usan para darle un nombre temporal y más amigable a las **tablas**, **columnas** y **funciones**. Los **alias** se definen durante una consulta y persisten **solo** durante esa consulta.

Para definir un alias usamos las iniciales **AS** precediendo a la columna que estamos queriendo asignarle ese alias.

```sql
SELECT nombre_columna1 AS alias_nombre_columna1
FROM nombre_tabla;
```

Alias para una columna

```
SELECT razon_social_cliente AS nombre
FROM cliente
WHERE nombre LIKE 'a%';
```

Alias para una columna

```
SELECT razon_social_cliente AS nombre
FROM cliente
WHERE nombre LIKE 'a%';
```

Seleccionamos la **columna** *razon_social_cliente* y le asignamos el **alias** nombre.

Alias para una columna

```
SELECT razon_social_cliente AS nombre
FROM cliente
WHERE nombre LIKE 'a%';
```

En el **FROM** elegimos tabla cliente.
Con el **WHERE** filtramos los registros de la columna nombre que empiecen con la letra a.

Alias para una tabla

```sql
SELECT nombre, apellido, edad
FROM alumnos_comision_inicial AS alumnos;
```

Alias para una tabla

```
SELECT nombre, apellido, edad
FROM alumnos_comision_inicial AS alumnos;
```

Seleccionamos las columnas nombre, apellido y edad.

Alias para una tabla

```
SELECT nombre, apellido, edad
FROM alumnos_comision_inicial AS alumnos;
```

Hacemos la consulta sobre la tabla *alumnos_comision_inicial* y le **asignamos** el **alias** alumnos.

No es recomendable asignar más de una palabra dentro de un alias. En el caso de necesitarlo, utilizar "_".

De este modo, podemos darle alias a las **columnas** y **tablas** que vamos trayendo y hacer más legible la manipulación de datos, teniendo siempre presente que los alias **no modifican** los nombres originales en la base de datos.


DigitalHouse>
Coding School