MANEJO BASES DE DATOS SQL

Modulo 1 seleccionando columnas.

En SQL, los datos se guardan en tablas. A la hora de consultar los datos, podemos traer la información de todas las columnas o sólo de algunas que necesitemos. En este ejercicio, aprenderemos a seleccionar todas las columnas de una tabla en SQL utilizando el comodín \*.

La instrucción SELECT \* FROM nombre\_tabla; nos permite seleccionar todas las columnas de la tabla nombrada. Esto es útil cuando queremos obtener toda la información de una tabla sin filtrar ninguna columna en particular.

En SQL estas insrucciones se componen de *claúsulas*. En este caso nuestra consulta se compone de SELECT \* que selecciona todo y from tabla que indica desde qué tabla se seleccionará.

Ejercicio

Selecciona todas las columnas de la **tabla** llamada **productos**

Select \* from productos

*Seleccionando una columna de la tabla*

En el ejercicio anterior aprendimos que podemos seleccionar toda la información de una tabla utilizando el comodín \*, pero también es posible seleccionar una única columna de la tabla.

Por ejemplo, si la tabla usuarios tiene email, nombre y apellido, podríamos seleccionar solamente los emails con:

SELECT email FROM usuarios;

select nombre from usuarios

*Seleccionando múltiples columnas de una tabla*

En este ejercicio aprenderemos a seleccionar múltiples columnas de una tabla utilizando SELECT. Para lograrlo, simplemente tenemos que nombrar cada columna de la tabla por separado. Por ejemplo, si tenemos una tabla llamada alumnos, y en esta tabla las columnas nombre, apellido y edad, podemos seleccionar estas 3 columnas con:

SELECT nombre, apellido, edad FROM nombre\_tabla;

También, es importante destacar que SQL es un lenguaje insensible a las mayúsculas, es decir, podemos escribir la misma instrucción como:

select nombre, apellido, edad from nombre\_tabla;

Sólo los nombres de las palabras reservadas son insensibles a las mayúsculas. En el caso de los nombres de las columnas y las tablas debemos respetar el cómo fueron creados para mantener la consistencia.

select nombre, precio, proveedor from productos

*Asignando un alias a una columna con "AS"*

En SQL, podemos asignar un **alias** o nombre alternativo a una columna en el resultado de la consulta, utilizando la cláusula AS.

Por ejemplo, si tenemos una tabla con la columna 'col1', podemos asignarle un alias de la siguiente forma:

SELECT col1 AS col\_nombre1 FROM tabla;

select nombre AS cliente from usuarios

*Asignando alias a varias columnas con "AS"*

Se puede cambiar el nombre a múltiples columnas en la misma consulta. Por ejemplo, podemos cambiar el nombre de la columna 'col1' a 'col\_nombre1' y el nombre de la columna 'col2' a 'col\_nombre2' utilizando la siguiente consulta:

SELECT col1 AS col\_nombre1, col2 AS col\_nombre2 FROM tabla;

select nombre As nombre\_usuario, apellido As apellido\_usuario from usuarios

*Asignando un alias con AS y comillas dobles*

En SQL, podemos utilizar la cláusula AS junto con comillas dobles para cambiar el nombre de una columna en los resultados de una consulta. Esto es útil cuando queremos dar un nombre más descriptivo o cuando el nombre de la columna contiene espacios o tildes.

Por ejemplo, consideremos una tabla llamada 'empleados' con las columnas 'nombre\_completo' y 'sueldo'. Si deseamos cambiar el nombre de la columna 'sueldo' a 'Salario de empleados', podemos utilizar la siguiente consulta:

SELECT nombre\_completo, sueldo AS "Salario de Empleados" FROM empleados;

select nombre, email as "Correo electrónico" from usuarios

Módulos 2 seleccionando filas

*Utilizando el operador mayor que*

La cláusula *WHERE* en SQL se utiliza para filtrar los registros de una tabla según una condición específica.

Por ejemplo, si disponemos de una tabla llamada "productos" con la columna precio, podemos recuperar todas las filas en las que el precio sea mayor a 100.

SELECT \* FROM productos WHERE precio > 100;

Cuando se utiliza where, se utiliza en conjunto con un *operador* que nos ayuda a comparar los datos. En el ejemplo anterior se utilizó el operador mayor que (>)

Un detalle importante es que las claúsulas tienen un orden.

1. select,
2. from
3. where

Si cambiamos el orden de estas claúsulas obtendremos un error de sintaxis.

Ejercicio

Se tiene una **tabla** llamada **productos**, con las columnas id, nombre, precio y descuento. Selecciona todos los registros cuyo descuento sea mayor a 10.

select \* from productos where descuento >10

*Ejercitando el uso del operador mayor que*

Se tiene una **tabla** llamada **productos**, con las columnas id, nombre, precio y descuento.

Selecciona todos los registros cuyo precio sea mayor a 200.

select \* from productos where precio >200

*Utilizando el operador mayor o igual que*

El operador mayor o igual que (>=) se utiliza para seleccionar registros en los que el valor de una columna sea mayor o igual a un valor específico. Por ejemplo, podemos seleccionar todos los productos cuyo precio sea mayor o igual a $100 utilizando la siguiente consulta:

SELECT \* FROM productos WHERE precio >= 100;

Ejercicio

Selecciona todos los registros de la **tabla productos** en los que el valor de la columna 'precio' sea mayor o igual a 50.

select \* from productos where precio >=50

Si mostraras sólo los productos con precio a mayor a 50, se mostaría la Lámpara de escritorio? **NO**

*Utilizando el operador "menor que"*

El operador menor que (<) se utiliza para comparar valores y seleccionar filas donde el valor de una columna sea estrictamente menor que un valor específico. Este operador es útil cuando queremos filtrar registros y obtener aquellos que tienen un valor menor a un límite determinado.

Por ejemplo, si tenemos una tabla de productos con las columnas col1, col2 y col3, podemos utilizar la siguiente consulta para seleccionar todas las columnas donde el valor de col1 sea menor a 10:

SELECT \* FROM productos WHERE col1 < 10;

Ejercicio

Se tiene una **tabla usuarios** con las columnas id, nombre, apellido, email y telefono. Selecciona todos los registros de la tabla usuarios donde el valor de la columna id sea menor a 3.

select \* from usuarios where id <3

*Utilizando el operador "menor o igual que" en una condición*

Podemos utilizar el operador 'menor o igual que' (<=) en una condición para seleccionar registros en los que el valor de una columna sea menor o igual a un valor dado. Por ejemplo, si tenemos una tabla de productos con una columna 'precio', podemos seleccionar todos los productos cuyo precio sea menor o igual a x utilizando la consulta.

SELECT \* FROM productos WHERE precio <= x;

Ejercicio

Selecciona todos los registros de la **tabla productos** en los que el valor de la columna 'precio' sea menor o igual a 100.

select \* from productos where precio <= 100

*Seleccionando multiples filas bajo una condición*

En algunas situaciones seleccionaremos ciertas columnas y a la vez aplicaremos condiciones.

Por ejemplo, si tenemos una tabla llamada 'productos' con las columnas 'id', 'nombre', 'precio' y 'cantidad', podemos seleccionar el id y nombre de todos los productos cuyo precio sea mayor a 30 utilizando la siguiente consulta:

SELECT id, nombre FROM productos WHERE precio > 30;

Como vemos, este ejercicio se resuelve con las mismas reglas que hemos visto hasta ahora, aplicando la claúsulas en el siguiente orden:

1. SELECT,
2. FROM
3. WHERE

Ejercicio

Selecciona el nombre, precio y cantidad de la **tabla productos** cuya cantidad sea mayor a 6.

select nombre, precio, cantidad from productos where cantidad > 6

*Seleccionando filas bajo una condición de igualdad*

Para seleccionar un valor en específico utilizaremos el operador =

Ejemplo:

SELECT \* FROM productos WHERE precio = 100;

Ejercicio

Selecciona el nombre del usuario de la **tabla usuarios** con id igual a 2

select nombre from usuarios where id = 2

*Seleccionando filas bajo una condición de igualdad (tipo de dato string)*

En SQL, para comparar textos debemos utilizar comillas simples ('') o comillas dobles (""). Por ejemplo, si tenemos una tabla productos con un producto con nombre **Camiseta** podemos seleccionarlo utilizando:

SELECT \* FROM productos WHERE nombre = 'Camiseta';

Al comparar un string en una condición WHERE, debemos asegurarnos de encerrar el valor buscado entre comillas.

¿Por qué debemos envolver los textos en comillas?

Cuando envolvemos un texto entre comillas en SQL, estamos indicando que no se trata de una palabra clave ni de un nombre de tabla o columna, sino que es un valor que debe ser tomado literalmente.

Ejercicio

Selecciona todas las filas de la **tabla productos** donde el nombre del producto sea 'Pantalón'.

select \* from productos where nombre = 'Pantalón'

*Seleccionando filas bajo una condición de igualdad (tipo de dato string) parte 2*

Es importante recordar que al trabajar con strings, la comparación es sensible a mayúsculas y minúsculas. Por lo tanto, 'Camiseta' y 'camiseta' se considerarán diferentes valores en la comparación. Si deseamos realizar una comparación sin considerar la distinción entre mayúsculas y minúsculas, se pueden utilizar funciones o cláusulas específicas proporcionadas por el motor de base de datos.

Ejercicio

Selecciona todos los productos de la **tabla productos** que tengan el nombre 'Silla de Oficina'.

Puedes probar con 'c' y observar que no obtendrás ningún resultado.

select \* from productos where nombre = "Silla de Oficina"

*Seleccionando filas bajo una condición de igualdad (tipo de dato booleano true)*

Hasta el momento hemos trabajado con dos tipos de datos: números enteros, como el precio del producto, y strings, como 'Camiseta'. En este ejercicio introduciremos el tipo de dato *Boolean*, el cual puede guardar como valor verdadero o falso, *TRUE* o *FALSE*.

Supongamos que tenemos una tabla de productos con una columna 'destacado' de tipo booleano que indica si un producto está destacado o no. Para seleccionar todos los productos que están marcados como destacados, podemos usar la siguiente consulta:

SELECT \* FROM productos WHERE destacado = true;

Adicionalmente se pueden ocupar los valores 1 y 0 en lugar de las palabras reservadas true o false, por ejemplo la siguiente consulta es identica a la anterior.

SELECT \* FROM productos WHERE destacado = 1;

Ejercicio

Se tiene una tabla de usuarios con los campos id, nombre, apellido, email, teléfono y status. La columna status es de tipo booleano.

Selecciona todos los usuarios de la **tabla usuarios** cuyo status es activo.

select \* from usuarios where status = true

*Seleccionando filas bajo una condición de igualdad (tipo de dato booleano false)*

Supongamos que queremos seleccionar todos los usuarios cuyo status es inactivo en una tabla llamada 'usuarios'. Podemos hacer esto utilizando la siguiente consulta:

SELECT \* FROM usuarios WHERE status = false;

En esta consulta, estamos utilizando la cláusula WHERE para buscar todos los usuarios que tengan el valor 'false' en la columna 'status'.

Ejercicio

Selecciona todos los productos de la **tabla productos** que no están destacados.

select \* from productos where DESTACADO = false

*Utilizando dos condiciones con operador "and"*

La claúsula *WHERE* se puede combinar con el operador *AND* para juntar múltiples condiciones en una consulta SQL. Por ejemplo, si queremos seleccionar todos los usuarios cuyo nombre es 'Juan' y apellido es 'Pérez', podemos utilizar la siguiente consulta:

SELECT \* FROM usuarios WHERE nombre = 'Juan' AND apellido = 'Pérez';

Cuando se utiliza el operador AND se deben cumplir ambas condiciones, en este caso el nombre debe ser 'Juan' y el apellido debe ser Pérez. En caso de que cualquiera de ellos sea distinto, no se mostrará.

Ejercicio

Se tiene una **tabla de usuarios** con los campos id, nombre, apellido, email y teléfono.

Selecciona todos los usuarios cuyo nombre es 'María' y su email es '[mariagarcia@hotmail.com](mailto:mariagarcia@hotmail.com)' de la tabla de usuarios.

select \* from usuarios where nombre = 'María' and email = 'mariagarcia@hotmail.com'

*Utilizando dos condiciones con operador "and" parte 2*

Ejercicio

Se tiene una tabla llamada productos que tiene los campos id, nombre, agotado y precio. La columna precio es de tipo Integer mientras que la columna agotado es de tipo Boolean.

Selecciona los productos de la **tabla productos** que estén agotados y tengan un precio mayor a 100.

select \* from productos where agotado and precio > 100

*Utilizando operador "OR"*

El operador OR se utiliza para combinar múltiples condiciones en una cláusula WHERE en SQL. Cuando se usa el operador OR, al menos una de las condiciones debe ser verdadera para que el registro se incluya en el resultado.

Por ejemplo, si deseamos seleccionar todos los productos que sean de color 'Azul' o 'Verde', podemos usar la siguiente consulta:

SELECT \* FROM productos WHERE color = 'Azul' OR color = 'Verde';

Esto seleccionará todos los registros de la tabla 'productos' que tengan el color 'Azul' o el color 'Verde'.

Ejercicio

Se tiene una **tabla productos** con los campos id, nombre, precio y descuento. El campo precio y el campo descuento son de tipo integer.

Selecciona todos los productos cuyo precio sea mayor a 1000 o su descuento sea igual a 20.

select \* from productos where precio > 1000 or descuento = 20

*Utilizando dos condiciones con operador "or"*

Se tiene una tabla clientes con los campos id, nombre, ciudad y saldo. La ciudad es de tipo texto, el saldo es número entero.

Selecciona aquellos clientes de la **tabla clientes** que sean de la ciudad 'Madrid' o que su saldo sea negativo.

select \* from clientes where ciudad = 'Madrid' or saldo < 0

*Seleccionando una fecha*

Otro tipo de dato es el de fecha, *Date* en inglés. Por defecto, las fechas se guardan en un formato YYYY-MM-DD, indicando primero el año en 4 dígitos, luego el mes con dos dígitos y finalmente el día con dos dígitos. Un ejemplo de una fecha en este formato sería 2022-01-01

Sobre las fechas podemos hacer distinto tipo de operaciones, pero primero aprenderemos a utilizarlas para filtrar. Por ejemplo, podemos obtener todos los productos de una tabla cuya fecha sea mayor o igual al primero de enero de 2022:

SELECT \* FROM productos WHERE fecha\_de\_creación >= '2022-01-01';

Ejercicio

Se tiene una tabla de productos con los campos id, nombre, precio y fecha\_de\_creacion. El campo fecha\_de\_creacion es de tipo Date.

Selecciona todos los productos de la **tabla productos** que fueron creados después de '2021-05-01'.

select \* from productos where fecha\_de\_creacion > '2021-05-01'

*Seleccionando datos entre dos valores con "between"*

El operador BETWEEN se utiliza para seleccionar registros cuyos valores se encuentren dentro de un rango específico.

Por ejemplo, podemos seleccionar todos los productos cuyo stock se encuentre entre 10 y 50 utilizando la siguiente consulta:

SELECT \* FROM productos WHERE stock BETWEEN 10 AND 50;

Un detalle importante a mencionar es que el operador between es inclusivo. Es decir, en el ejemplo se incluirían los valores de 10 o 50.

Si quieres buscar con otro tipo de intervalo, por ejemplo que incluya el valor 10 y no el valor 50 puedes utilizar dos condiciones unidas con un operador and SELECT \* productos WHERE stock >= 10 and stock < 50

Selecciona todos los productos cuyo stock se encuentre entre 20 y 30.

select \* from productos where stock between 20 and 30

*Seleccionando filas con "like"*

Supongamos que queremos buscar todos los usuarios cuyo nombre empiece con la letra 'J' en la tabla de usuarios. Podemos hacer esto utilizando la siguiente consulta:

SELECT \* FROM usuarios WHERE nombre LIKE 'J%'

En esta consulta, estamos utilizando el operador LIKE para buscar todos los nombres de usuarios que comiencen con la letra 'J'.

El símbolo '%' es un comodín que representa cualquier cantidad de caracteres adicionales. En este caso, estamos utilizando '%' después de la letra 'J' para indicar que queremos buscar cualquier nombre que comience con 'J' y tenga cualquier número de caracteres adicionales después de ella.

Ejercicio

Se tiene una **tabla usuarios** con los campos id, nombre, apellido, email y teléfono. El campo nombre es de tipo texto.

Se pide seleccionar todos los usuarios cuyo apellido empiece con 'Ma'

select \* from usuarios where apellido like 'Ma%'

*Seleccionando con comodin al principio*

Supongamos que queremos buscar todos los usuarios cuyo nombre termine con la letra 's' en la tabla de usuarios. Podemos hacer esto utilizando la siguiente consulta:

SELECT \* FROM usuarios WHERE nombre LIKE '%s'

En esta consulta, estamos utilizando la cláusula LIKE para buscar todos los nombres de usuarios que terminen con la letra 's'. El símbolo '%' es un comodín que representa cualquier cantidad de caracteres adicionales. En este caso, estamos utilizando '%' antes de la letra 's' para indicar que queremos buscar cualquier nombre que termine con 's' y tenga cualquier número de caracteres adicionales antes de ella.

Ejercicio

Selecciona todos los usuarios de la **tabla usuarios** cuyo nombre termine con la letra 'o'

select \* from usuarios where nombre like '%o'

*Seleccionando registros sin valores nulos*

Algunos registros pueden tener valores **nulos** para algunos de sus campos. Por ejemplo, podríamos tener una tabla de usuarios con nombres y emails pero no tener todos los nombres de cada uno de los registros como ilustra la siguiente tabla.

| **ID** | **NOMBRE** | **EMAIL** |
| --- | --- | --- |
| 1 | Juan Pérez | [juan.perez@email.com](mailto:juan.perez@email.com) |
| 2 | María Gómez | [maria.gomez@email.com](mailto:maria.gomez@email.com) |
| 3 |  | [carlos.diaz@email.com](mailto:carlos.diaz@email.com) |
| 4 |  | [ana.torres@email.com](mailto:ana.torres@email.com) |
| 5 | Luis Méndez | [luis.mendez@email.com](mailto:luis.mendez@email.com) |

Para seleccionar todos los valores no nulos utilizaremos IS NOT NULL

Por ejemplo, en la tabla usuarios previamente mostrada podemos seleccionar todos los nombres no nulos utilizando SELECT \* FROM empleados WHERE nombre IS NOT NULL;

Esto nos devolverá todos los usuarios cuyo nombre no sea nulo.

Ejercicio

Se tiene una **tabla productos** con id, nombre, precio y descuento, siendo descuento de tipo integer.

Selecciona todos los registros de la tabla productos cuyo campo descuento no sea nulo.

select \* from productos where descuento is not null

*Seleccionando registros con valores nulos*

Así como podemos seleccionar valores no nulos, también podemos seleccionar valores nulos.

Por ejemplo, si queremos encontrar todos los usuarios que no tengan un número de teléfono registrado en la tabla de usuarios, podemos utilizar la siguiente consulta:

SELECT \* FROM usuarios WHERE telefono IS NULL;

Ejercicio

Se tiene una **tabla usuarios** con id, nombre, apellido, email y teléfono

Selecciona todos los usuarios que no tengan un email registrado en la tabla de usuarios.

select \* from usuarios where email is null

### MODULO 3 *Ordenando resultados (orden y limit)*

### ***Ordenando filas***

En este ejercicio, aprenderemos a ordenar las filas de una tabla en SQL, y para esto, estudiaremos una nueva cláusula llamada ORDER BY.

ORDER BY se utiliza para ordenar los resultados de una consulta según una o más columnas. Por defecto, se ordena de forma ascendente.

Por ejemplo, si tenemos una tabla de productos con los campos 'id', 'nombre' y 'precio', podemos ordenar los productos por precio del menor al mayor con:

SELECT \* FROM productos ORDER BY precio;

Es importante tener en cuenta que las claúsulas tienen que especificarse justo en este orden:

1. SELECT
2. FROM
3. ORDER BY

El orden de los resultados dependerá del tipo de dato: los números se ordenan de menor a mayor, los textos alfabéticamente y las fechas cronológicamente.

## **Ejercicio**

Ordena los registros de la **tabla usuarios** por el campo 'nombre'

select \* from usuarios order by nombre

### ***Ordenando filas asc explicito***

Con la claúsula ORDER BY podemos especificar la dirección de los resultados. Se puede ordenar en orden ascendente (ASC) o descendente (DESC).

En el ejercicio anterior aprendimos que implícitamente (si no lo indicamos en la consulta) los resultados se ordenan de menor a mayor, es decir, de forma ascendente. Para hacer nuestras consultas claras en su intención, indicaremos la dirección al momento de hacer la consulta:

SELECT \* FROM productos ORDER BY precio ASC;

Con esta consulta obtendremos el mismo resultado que utilizando:

SELECT \* FROM productos ORDER BY precio;

## **Ejercicio**

En este ejercicio se tiene una **tabla usuarios** con los campos id, nombre, apellido, email y teléfono. Se te pide ordenar los registros de la tabla 'usuarios' por el campo 'nombre' en orden ascendente.

select \* from usuarios order by nombre asc

### ***Ordenando filas desc***

La cláusula ORDER BY se utiliza para ordenar los resultados de una consulta. Por defecto el orden es ascendente, pero se puede especificar que sea descendente utilizando la palabra clave DESC. Por ejemplo:

SELECT \* FROM productos ORDER BY precio DESC;

## **Ejercicio**

Se tiene una **tabla productos** con los campos id, nombre, precio y stock. Selecciona sólo los precios de la tabla 'productos' ordenados de forma descendente.

select precio from productos order by precio desc

### ***Ordenando filas con valores nulos***

En SQL los registros nulos se consideran con el valor mas bajo, por lo que al ordenar ascendentemente los veremos al principio y al ordenar descendentemente los veremos al final.

En este ejercicio, no aprenderemos ninguna instrucción nueva, únicamente se te pedirá que ordenes una tabla por una columna específica y observes cómo se ordenan los valores nulos.

## **Ejercicio**

Ordena la **tabla empleados** por la columna 'salario' de manera ascendente.

select \* from empleados order by salario asc

### ***Ordenando con nulos al final***

A veces necesitamos que los valores nulos queden al principio o al final de la lista independiente de en cual direccion ordenemos. Para lograrlo utilizarmos ORDER BY junto con NULLS FIRST o NULLS LAST para especificar cómo queremos que se ordenen las filas con valores nulos.

Con NULLS FIRST se muestran los nulos primeros y con NULLS LAST se muestran al final

La consulta completa tendría la siguiente forma: SELECT \* FROM tabla ORDER BY campo NULLS FIRST

## **Ejercicio**

Dada una **tabla productos** con las columnas 'id', 'nombre' y 'precio' con los siguientes registos.

| **ID** | **NOMBRE** | **PRECIO** |
| --- | --- | --- |
| 1 | Producto 1 | 100 |
| 2 | Producto 2 | NULL |
| 3 | Producto 3 | 50 |
| 4 | Producto 4 | NULL |
| 5 | Producto 5 | 200 |

Ordena las filas de la tabla en función del precio de forma ascendente. Asegúrate de que las filas con valores nulos en la columna 'precio' aparezcan al final de la lista ordenada.

SELECT \* FROM productos ORDER BY PRECIO NULLS LAST

### ***Combinaciones de orden***

En algunas situaciones vamos a querer ordenar en función de múltiples columnas. Por ejemplo, si queremos obtener una lista de todos los productos ordenados por su stock y luego por su color, podemos seleccionar todos los campos de la tabla y ordenarlos primero por el campo stock y luego por el campo color de la siguiente manera:

SELECT \* FROM productos ORDER BY stock ASC, color ASC

## **Ejercicio**

Se tiene la **tabla empleados** con la siguiente información:

| **ID** | **NOMBRE** | **SALARIO** |
| --- | --- | --- |
| 1 | Juan Pérez | 4800 |
| 2 | María López | 5500 |
| 3 | Pedro García | 5500 |
| 4 | Ana Martínez | 5500 |
| 5 | Luis Rodríguez | 4800 |

Selecciona una lista de todos los empleados ordenados por su salario y por su nombre.

SELECT \* FROM empleados ORDER BY SALARIO ASC, NOMBRE ASC

### ***Combinaciones de orden asc y desc***

Supongamos que queremos obtener una lista de todos los productos cuyo precio sea mayor a $100 y ordenarlos primero por su precio de forma descendente y luego por su nombre de forma ascendente. Podemos hacer esto utilizando la siguiente consulta:

SELECT \* FROM productos WHERE precio > 100 ORDER BY precio DESC, nombre ASC;

## **Ejercicio**

Se tiene la **tabla productos** con la siguiente información:

| **ID** | **NOMBRE** | **STOCK** | **COLOR** |
| --- | --- | --- | --- |
| 1 | Silla | 10 | Rojo |
| 2 | Mesa | 5 | Verde |
| 3 | Lámpara | 15 | Azul |
| 4 | Escritorio | 8 | Blanco |
| 5 | Estantería | 12 | Negro |

Selecciona todos los registros de la tabla 'productos' y ordénalos primero por 'stock' de forma descendente y luego por 'color' de forma ascendente.

SELECT \* FROM productos ORDER BY STOCK DESC, COLOR ASC;