**OPERACIONES BASICAS FUNDAMENTALES (CRUD)**

* **INSERT**  
  La declaración INSERT INTO se utiliza para insertar nuevos registros en una tabla. Es posible escribir la INSERT INTO de dos maneras:

**1.** Especifique tanto los nombres de las columnas como los valores que se insertarán:

INSERT INTO nombre\_tabla(*column1*,*column2*,*column3*,...)  
VALUES (*value1*,*value2*,*value3*,...);

**2.** Si está agregando valores para todas las columnas de la tabla, no necesita especificar los nombres de las columnas en la consulta SQL. Sin embargo, asegúrese de que el orden de los valores sea el mismo que el de las columnas de la tabla. Aquí, la sintaxis de INSERT INTO sería la siguiente:

INSERT INTO nombre\_tabla VALUES (*value1*,*value2*,*value3*,...);

## SELECT La instrucción SELECT se utiliza para seleccionar datos de una base de datos. Los datos devueltos se almacenan en una tabla de resultados, denominada conjunto de resultados.

### Sintaxis

SELECT column1, column2,... FROM nombre\_tabla;

## UPDATE La declaración UPDATE se utiliza para modificar los registros existentes en una tabla.

### Sintaxis

UPDATE nombre\_tabla SET column1 = value1, column2 = value2,... WHERE condición;

**Nota:** Al momento de actualizar registros en una tabla se debe tener en cuenta que la cláusula WHERE especifica qué registros deben actualizarse. Si se omite esta cláusula (WHERE), se actualizarán todos los registros de la tabla.

Aquí, columna1, columna2, son los nombres de campo de la tabla de la que desea seleccionar datos. Si desea seleccionar todos los campos disponibles en la tabla, utilice la siguiente sintaxis:

SELECT \* FROM nombre\_tabla;

## Actualizar múltiples registros Es la cláusula WHERE que determina cuántos registros se actualizarán. La siguiente instrucción SQL actualizará ContactName a "Juan" para todos los registros donde el país sea "México":

### Ejemplo

UPDATE Clientes SET NombreContacto='Maria' WHERE Pais='Argentina';

## DELETE La instrucción DELETE se utiliza para eliminar registros existentes en una tabla.

### Sintaxis

DELETE FROM nombre\_tabla WHERE condición;

**Nota:** Si se desea realizar una eliminación, se debe tener en cuenta que la cláusula WHERE especifica qué registros deben eliminarse. Si se omite WHERE se eliminarán todos los registros de la tabla.

## Eliminar todos los registros Es posible eliminar todas las filas de una tabla sin eliminar la tabla. Esto significa que la estructura de la tabla, los atributos y los índices estarán intactos:

DELETE FROM nombre\_tabla;

## Sintaxis Básica del Lenguaje

## WHERE La WHERE cláusula se utiliza para filtrar registros. Se utiliza para extraer solo aquellos registros que cumplen una condición específica.

### Sintaxis SELECT column1, column2,... FROM nombre\_tabla WHERE condicion;

## Campos de texto frente a campos numéricos SQL requiere comillas simples alrededor de valores de texto (la mayoría de los sistemas de bases de datos también permitirán comillas dobles). Sin embargo, los campos numéricos no deben estar entre comillas:

### Ejemplo SELECT \* FROM Clientes WHERE ClientesID=1;

## Los siguientes operadores se pueden utilizar en la WHERE cláusula:

**=** igual  
**>** Mayor que  
**<** Menor que  
**>=** Mayor igual que  
**<=** Menor igual que  
**<>** Diferente  
**BETWEEN** Entre un cierto rango  
**LIKE** Buscar un patrón  
**IN** Para especificar múltiples valores posibles para una columna

## ORDER BY La ORDER BY palabra clave se utiliza para clasificar el conjunto de resultados en orden ascendente (ASC) o descendente (DESC).

### Sintaxis

SELECT column1, column2,... FROM nombre\_tabla ORDER BY column1, column2, ... ASC|DESC;

## La siguiente instrucción SQL selecciona todos los clientes de la tabla "Clientes", ordenados por la columna "País":

### Ejemplo SELECT \* FROM Clientes ORDER BY Pais;

## Ejemplo ORDEN POR DESC La siguiente instrucción SQL selecciona todos los clientes de la tabla "Clientes", ordenados DESCENDENTE por la columna "País":

SELECT \* FROM Clientes ORDER BY Pais DESC;

## Ejemplo 2 ordenar por Varias Columnas La siguiente instrucción SQL selecciona todos los clientes de la tabla "Clientes", ordenados de forma ascendente por el "País" y descendente por la columna "Nombre del cliente":

SELECT \* FROM Clientes ORDER BY Pais ASC, NombreCliente DESC;

* **JOIN**  
  Una cláusula JOIN se usa para combinar filas de dos o más tablas, en función de una columna relacionada entre ellas.

Veamos una selección de la tabla "pedido":

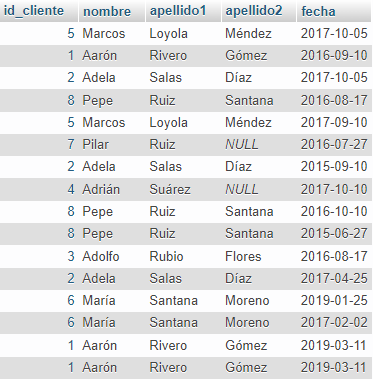


Luego, observe una selección de la tabla "Clientes":



Observe que la columna "id\_cliente" en la tabla "pedido" hace referencia al "id" en la tabla "cliente". Luego, podemos crear la siguiente instrucción SQL (que contiene un INNER JOIN), que selecciona registros que tienen valores coincidentes en ambas tablas:

### Ejemplo SELECT pedido.id\_cliente, cliente.nombre, cliente.apellido1, cliente.apellido2, pedido.fecha FROM pedido INNER JOIN cliente ON pedido.id\_cliente=cliente.id;



## Diferentes tipos de JOIN SQL Estos son los diferentes tipos de JOIN en SQL:

* **(INNER) JOIN:** Devuelve registros que tienen valores coincidentes en ambas tablas
* **LEFT (OUTER)** JOIN: Devuelve todos los registros de la tabla de la izquierda y los registros coincidentes de la tabla de la derecha.
* **RIGHT (OUTER)** JOIN: Devuelve todos los registros de la tabla de la derecha y los registros coincidentes de la tabla de la izquierda.
* **FULL (OUTER)** JOIN: Devuelve todos los registros cuando hay una coincidencia en la tabla izquierda o derecha



## INNER JOIN La palabra clave INNER JOIN selecciona registros que tienen valores coincidentes en ambas tablas.

### Sintaxis

SELECT nombre\_columna(s) FROM table1 INNER JOIN table2 ON table1.column\_name = table2.column\_name;



## LEFT JOIN Este devuelve todos los registros de la tabla de la izquierda (tabla1) y los registros coincidentes de la tabla de la derecha (tabla2). El resultado es 0 registros del lado derecho, si no hay coincidencia.

### Sintaxis

SELECT nombre\_columna(s) FROM table1 LEFT JOIN table2ON table1.column\_name = table2.column\_name;

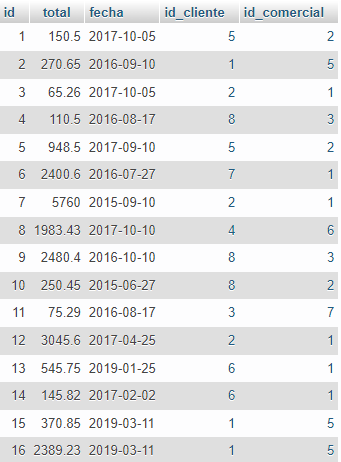
**Nota:** Se puede encontrar en algunas bases de datos como LEFT JOIN o LEFT OUTER JOIN.



## A continuación se muestra una selección de la tabla "cliente":



Y una selección de la tabla "pedido":



Ejemplo:

Devuelve un listado que solamente muestre los clientes que no han realizado ningún pedido.

SELECT DISTINCT cliente.apellido1, cliente.apellido2, cliente.nombre, pedido.id FROM cliente LEFT JOIN pedido ON cliente.id=pedido.id\_cliente WHERE pedido.id\_cliente IS NULL;



## RIGHT JOIN RIGHT JOIN devuelve todos los registros de la tabla de la derecha (tabla2) y los registros coincidentes de la tabla de la izquierda (tabla1). El resultado es 0 registros del lado izquierdo, si no hay ninguna coincidencia.

### Sintaxis SELECT nombre\_columna(s) FROM table1 RIGHT JOIN table2 ON table1.nombre\_columna = table2.nombre\_columna;

**Nota:** Podemos encontrar esta cláusula como RIGHT JOIN o RIGHT OUTER JOIN en algunas bases de datos.



## A continuación se muestra una selección de la tabla "pedido":

Y una selección de la tabla "cliente”:

**Ejemplo**La siguiente instrucción SQL devolverá todos los clientes y cualquier pedido que hayan realizado:

SELECT pedido.id, cliente.nombre, cliente.apellido1 FROM pedido RIGHT JOIN cliente ON pedido.id\_cliente=cliente.id;



**Nota:** RIGHT JOIN devuelve todos los registros de la tabla de la derecha (cliente), incluso si no hay coincidencias en la tabla de la izquierda (pedido).

## FULL OUTER JOIN La FULL OUTER JOIN devuelve todos los registros cuando hay una coincidencia en los registros de la tabla izquierda (tabla1) o derecha (tabla2).

### Sintaxis

SELECT nombre\_columna(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condicion;



**Nota:** FULL OUTER JOIN y FULL JOIN son lo mismo. Además FULL OUTER JOIN puede devolver conjuntos de resultados muy grandes. Esta cláusula en MySQL se trabaja como CROSS JOIN

**Sintaxis CROSS JOIN:**SELECT *nombre\_columna(s)*  
FROM table1  
CROSS JOIN table2;

Si agregamos WHERE (tabla1 y tabla2 tienen una relación), CROSS JOIN producirá el mismo resultado que la cláusula INNER JOIN:

**Ejemplo:**

SELECT \* FROM cliente CROSS JOIN pedido;

A continuación podemos ver el resultado de esta consulta, donde se observan todos los datos de ambas tablas, la tabla clientes y la tabla pedidos. Nos trae todos los datos de los clientes incluyendo los que no tienen pedidos.



**Nota:** CROSS JOIN devuelve todos los registros coincidentes de ambas tablas, ya sea que la otra tabla coincida o no. Por lo tanto, si hay filas en "Clientes" que no tienen coincidencias en "Pedidos", o si hay filas en "Pedidos" que no tienen coincidencias en "Clientes", esas filas también se mostrarán.

**Ejemplo**   
La siguiente instrucción SQL selecciona todos los clientes y todos los pedidos:

SELECT \* FROM cliente CROSS JOIN pedido ON cliente.id=pedido.id\_cliente ORDER BY cliente.id;

Una selección del conjunto de resultados puede verse así:



## JOIN en múltiples tablas La siguiente instrucción SQL devuelve un listado con todos los clientes junto con los datos de los pedidos que han realizado. Este listado también debe incluir los clientes que no han realizado ningún pedido. El listado debe estar ordenado alfabéticamente por el primer apellido, segundo apellido y nombre de los clientes.

Devuelve un listado con todos los clientes junto con los datos de los pedidos que han realizado. Este listado también debe incluir los clientes que no han realizado ningún pedido. El listado debe estar ordenado alfabéticamente por el primer apellido, segundo apellido y nombre de los clientes.

SELECT DISTINCT cliente.id, cliente.apellido1, cliente.apellido2, cliente.nombre, pedido.id FROM cliente LEFT JOIN pedido ON cliente.id=pedido.id\_cliente ORDER BY cliente.apellido1 ASC;



# DELETE y UPDATE con JOIN

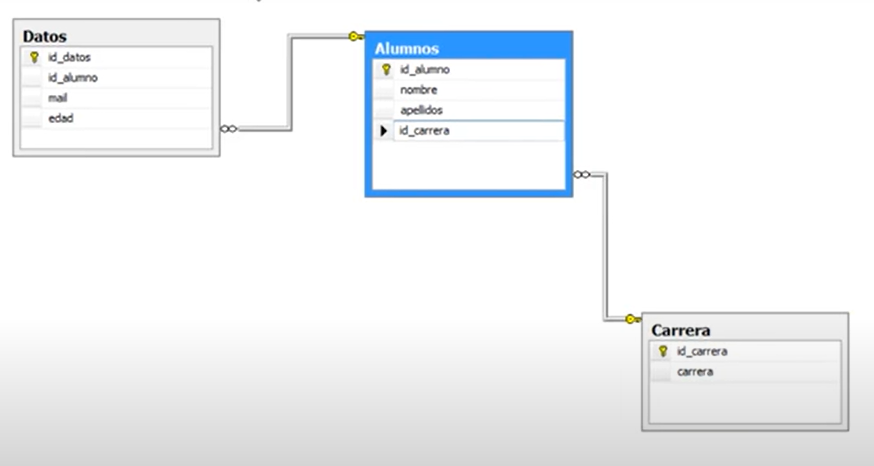
Para eliminar registros de una tabla siguiendo un criterio que tenga relación con otra tabla de la base de datos se puede establecer una join entre ambas tablas en la misma **instrucción SQL de DELETE**.

Es algo muy parecido al método para hacer un [**UPDATE** a partir de una join](http://www.dataprix.com/es/update-join-sqlserver) para actualizar registros en función de valores de otras tablas, pero en este caso, al tratarse de eliminar registros en lugar de actualizar, la sintaxis SQL cambia un poco.

Si queremos hacer un **DELETE de registros utilizando una join** lo único que hay que hacer es especificar en la primera línea de la sentencia la tabla o tablas en las que se quiere aplicar la eliminación de registros, y en la siguiente línea, comenzando con un FROM, la join de las tablas que necesitemos, y el criterio indicado con un WHERE.

## Ejemplo SQL de Delete y Update con Inner join

Un ejemplo sencillo de un delete y update con una join entre tablas. Se realizará una actualización y una eliminación de la siguiente base de datos. Donde se actualice el nombre de todos los alumnos cuya carrera sea Ingeniería de Sistemas.



UPDATE Alumnos *set* nombre =’Cambio’

FROM Alumnos *join* Carrera

ON Alumnos.id\_carrera = carrera.id\_carera

WHERE Carrera.id\_carrera=12

DELETE Alumnos

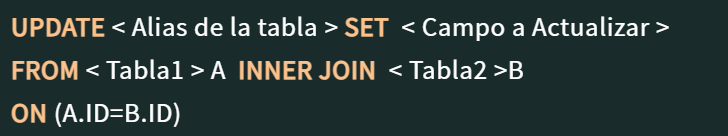
FROM Alumnos *join* Carrera

ON Alumnos.id\_carrera = carrera.id\_carera

**Como hacer Update con Inner Join**

Con la sentencia update podemos actualizar registros almacenados en una tabla de mi Base de datos pero en muchas ocasiones necesitamos actualizar cierto conjunto de registros a los cuales llegamos relacionando varias tablas por medio de un select, es ahí donde podemos usar la sentencia update con inner join para actualizar esos registros.

Sintaxis:



**Consideraciones a tener en cuenta al usar Update**

* Debemos tener en cuenta que estamos trabajando con la sentencia UPDATE y si nos equivocamos al momento de formar la consulta podemos actualizar registros que no deseamos que se actualicen.
* Crear primero la sentencia SELECT con todos los joins entre tablas para revisar el resultado de la consulta.
* Tener en cuenta que esta consulta podría afectar un poco el rendimiento de la base debido que aparte de actualizar los registros primero debe realizar la unión entre tablas.
* Tener en cuenta la cláusula where cuando trabajamos con la sentencia update.

# Subconsultas

El uso de Subconsultas es una técnica que permite utilizar el resultado de una tabla SELECT en otra consulta SELECT. Permite solucionar consultas complejas mediante el uso de resultados previos conseguidos a través de otra consulta.

El SELECT que se coloca en el interior de otro SELECT se conoce con el término de SUBSELECT. Ese SUBSELECT se puede colocar dentro de las cláusulas **WHERE**, **HAVING**, **FROM** o **JOIN**.

Reglas para crear una Subconsultas

* Ponga la Subconsulta entre paréntesis.
* En una Subconsulta, especifique sólo una columna o expresión a no ser que esté utilizando IN, ANY, ALL o EXISTS.
* Una Subconsulta no puede contener una cláusula BETWEEN ni LIKE.
* Una Subconsulta no puede contener una cláusula ORDER BY.
* Una Subconsulta de una sentencia UPDATE no puede recuperar datos de la misma tabla en la que deben actualizarse los datos.
* Una Subconsulta de una sentencia DELETE no puede recuperar datos de la misma tabla de la que deben suprimirse los datos.

## Uso de Subconsultas en instrucciones SELECT

### Uso de Subconsultas simples Las Subconsultas simples son aquellas que devuelven una única fila. Si además devuelven una única columna, se las llama ****Subconsultas escalares****, ya que devuelven un único valor.

L a sintaxis es:  
SELECT listaExpresiones   
FROM tabla  
WHERE expresión OPERADOR   
 (SELECT listaExpresiones  
 FROM tabla);

El operador puede ser **>,<,>=,<=,!=, =** o **IN**. Ejemplo:

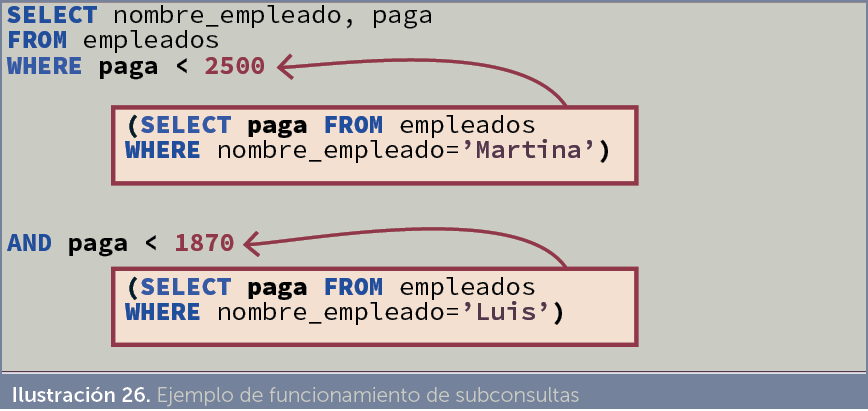
SELECT nombre\_empleado, paga  
FROM empleados  
WHERE paga <   
 (SELECT paga FROM empleados  
 WHERE nombre\_empleado=’Martina’);

Esa consulta muestra el nombre y paga de los empleados cuya paga es menor que la de la empleada Martina. Para que funcione esta consulta, la Subconsultas solo puede devolver un valor (solo puede haber una empleada que se llame Martina.

Se pueden usar Subconsultas las veces que haga falta:

SELECT nombre\_empleado, paga  
FROM empleados  
WHERE paga <   
 (SELECT paga FROM empleados  
 WHERE nombre\_empleado=’Martina’)  
AND paga >  
 (SELECT paga FROM empleado   
 WHERE nombre\_empleado=’Luis’);

En realidad lo primero que hace la base de datos es calcular el resultado de la Subconsultas:



La última consulta obtiene los empleados cuyas pagas estén entre lo que gana Luís (1870 euros) y lo que gana Martina (2500).

Las Subconsultas siempre se deben encerrar entre paréntesis y se deberían (aunque no es obligatorio, sí altamente recomendable) colocar a la derecha del operador relacional.

**Una Subconsultas que utilice los valores >,<,>=,... tiene que devolver un único valor**, de otro modo ocurre un error.

Además **tienen que devolver el mismo tipo y número de datos**para relacionar la Subconsultas con la consulta que la utiliza (no puede ocurrir que la Subconsultas tenga dos columnas y ese resultado se compare usando una sola columna en la consulta general).

### Uso de Subconsultas de múltiples filas En el apartado anterior se comentaba que las Subconsultas sólo pueden devolver una fila. Pero a veces se necesitan consultas del tipo: mostrar el sueldo y nombre de los empleados cuyo sueldo supera al de cualquier empleado del departamento de ventas.

La Subconsulta necesaria para ese resultado mostraría **todos** los sueldos del departamento de ventas. Pero no podremos utilizar un operador de comparación directamente ya que esa Subconsultas devuelve más de una fila. La solución a esto es utilizar instrucciones especiales entre el operador y la consulta, que permiten el uso de Subconsultas de varias filas.

Esas instrucciones son:

| **Instrucción** | **Significado** |
| --- | --- |
| **ANY**  o  **SOME** | Compara con cualquier registro de la Subconsultas. La instrucción es válida si hay un registro en la Subconsultas que permite que la comparación sea cierta.  Se suele utilizar la palabra ANY (SOME es un sinónimo) |
| **ALL** | Compara con todos los registros de la consulta. La instrucción resulta cierta si es cierta toda comparación con los registros de la Subconsultas |
| **IN** | No usa comparador, ya que sirve para comprobar si un valor se encuentra en el resultado de la Subconsultas |
| **NOT IN** | Comprueba si un valor no se encuentra en una Subconsultas |

Ejemplo:

SELECT nombre, sueldo   
FROM empleados  
WHERE sueldo >= ALL (SELECT sueldo FROM empleados);

La consulta anterior obtiene el empleado que más cobra. Otro ejemplo:

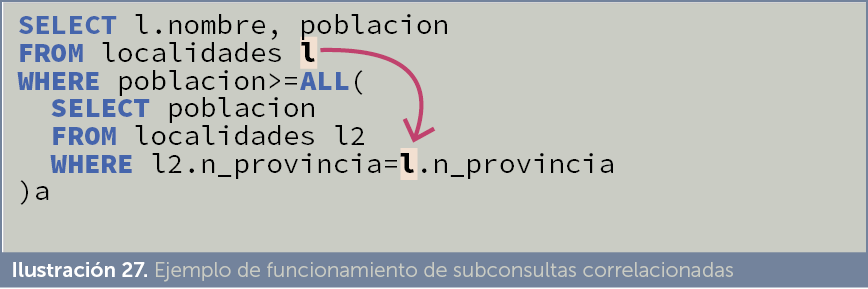
SELECT nombre FROM empleados   
WHERE dni IN (SELECT dni FROM directivos);

En ese caso se obtienen los nombres de los empleados cuyos dni están en la tabla de directivos.

### Consultas correlacionadas

También conocidas como Subconsultas Repetidas, son aquellas Subconsultas que su resultado depende de los valores que tenga la consulta externa. Esto implica que la Subconsulta podría ser ejecutada para cada fila d ela consulta externa. En las Subconsultas a veces se puede desear poder utilizar datos procedentes de la consulta principal. Eso es posible utilizando el alias de la tabla que queremos usar de la consulta principal.

Por ejemplo, supongamos que deseamos obtener de una base de datos geográfica, el nombre y la población de las localidades que sean las más pobladas de su provincia. Es decir, las localidades cuya población es la mayor de su provincia. Para ello necesitamos comparar la población de cada localidad con la de todas las localidades de su provincia. Supongamos que la tabla de las localidades almacena el nombre, población y el número de la provincia a la que pertenecen.

La consulta sería:

En el código anterior se observa que dentro de la Subconsultas usamos el alias l correspondiente a la tabla de localidades de la consulta principal (por eso se le ha puesto como alias l2 a la tabla localidades en la Subconsultas).

### Consultas EXISTS Este operador devuelve verdadero si la consulta que le sigue devuelve algún valor. Si no, devuelve falso. Se utiliza normalmente mediante consultas correlacionadas. Ejemplo:

### SELECT tipo,modelo, precio\_venta FROM piezas p WHERE EXISTS ( SELECT tipo,modelo FROM existencias WHERE tipo=p.tipo AND modelo=p.modelo);

Esta consulta devuelve las piezas que se encuentran en la tabla de existencias (es igual al ejemplo comentado en el apartado Subconsultas sobre múltiples valores).

La consulta contraria es:

SELECT tipo,modelo, precio\_venta  
FROM piezas p  
WHERE NOT EXISTS (  
 SELECT tipo,modelo FROM existencias  
 WHERE tipo=p.tipo AND modelo=p.modelo);

Normalmente las consultas EXISTS se pueden realizar de alguna otra forma con otros operadores.

**Consultas**

* **TOP:** Se utiliza para especificar el número de filas a mostrar en el resultado, es útil en tablas con muchos registros, para limitar el número de filas a mostrar en la consulta.

Cláusula SQL TOP para SQL SERVER

SELECT TOP número  
PERCENT nombre\_columna  
FROM nombre\_tabla

Cláusula SQL TOP para MySQL

SELECT columna(s) FROM tabla  
LIMIT númerofilas

Cláusula SQL TOP para ORACLE

SELECT columna(s) FROM tabla  
WHERE ROWNUM <= númerofilas

|  |  |  |
| --- | --- | --- |
| **nombre** | **apellido1** | **apellido2** |
| ANTONIO | PEREZ | GOMEZ |
| ANTONIO | GARCIA | RODRIGUEZ |
| PEDRO | RUIZ | GONZALEZ |

Respuesta consulta

|  |  |  |
| --- | --- | --- |
| **nombre** | **apellido1** | **apellido2** |
| ANTONIO | PEREZ | GOMEZ |
| ANTONIO | GARCIA | RODRIGUEZ |

* **OFFSET** y **FETCH**:   
  Son las opciones de la **ORDER BY** cláusula. Le permiten limitar el número de filas que devolverá una consulta.

El **OFFSET** indica el número de filas que se va a saltar antes de comenzar a devolver resultados de la consulta.

El **FETCH** indica el número de filas que devolverá después del **OFFSET**. (**FIRST** | **NEXT**).

**Ejemplo**:

SELEC product\_name, list\_price

FROM production.products

ORDER BY list\_price, product\_name

OFFSET 10 ROWS

FETCH NEXT 10 ROWS ONLY;

* **DISTINCT:** La SELECT DISTINCT es una declaración que se usa para devolver solo valores distintos (diferentes).

**Ejemplo:**

SELECT DISTINCT *column1*,*column2, ...*  
FROM *table\_name*;

**Ejemplo con numeración**:

SELECT COUNT(DISTINCT Country) FROM Customers;

**Funciones Básicas**

* **MIN()|MAX():**

**MIN:** Devuelve el valor más pequeño de la columna seleccionada.

**MAX:** Devuelve el valor más grande de la columna.

**Ejemplos:**

SELECT MIN(column\_name)

FROM table\_name;

SELECT MAX(column\_name)

FROM table\_name;

* **COUNT(), AVG() y SUM():**

**COUNT:** Función que devuelve el número de filas que coincide con un criterio específico.

**AVG:** Función que devuelve el valor promedio de una columna numérica.

**SUMA:** Función que encuentra la suma de los campos de una columna.

**Ejemplos:**

SELECT COUNT(column\_name) SELECT AVG(column\_name)

FROM table\_name; FROM table\_name;

SELECT SUM(Quantity)

FROM OrderDetails;

* **AND, OR, NOT:** Los operadores AND y OR se utilizan para filtrar registros en función de más de una condición:
* El AND operador muestra un registro si todas las condiciones separadas por AND son VERDADERAS.
* El OR operador muestra un registro si alguna de las condiciones separadas por OR es VERDADERA.
* El NOT operador muestra un registro si la(s) condición(es) NO ES VERDADERA.

**Ejemplos:**

SELECT \* FROM Customers  
WHERE Country='Germany' **AND** City='Berlin';

SELECT \* FROM Customers  
WHERE City='Berlin' **OR** City='München';

SELECT \* FROM Customers  
WHERE **NOT** Country='Germany';

**También se pueden combinar:**

SELECT \* FROM Customers  
WHERE Country='Germany' **AND** (City='Berlin' **OR** City='München');

SELECT \* FROM Customers  
WHERE **NOT** Country='Germany' **AND** **NOT** Country='USA';

* **Case:** La CASE declaración pasa por las condiciones y devuelve un valor cuando se cumple la primera condición (como una declaración if-then-else). Si ninguna se cumple entonces devuelve un NULL.

**Ejemplos:**

SELECT OrderID, Quantity,  
CASE  
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'  
    WHEN Quantity = 30 THEN 'The quantity is 30'  
    ELSE 'The quantity is under 30'  
END AS QuantityText  
FROM OrderDetails;

**También podemos ordenarlos:**

SELECT CustomerName, City, Country  
FROM Customers  
ORDER BY  
(CASE  
    WHEN City IS NULL THEN Country  
    ELSE City  
END);

* **BETWEEN:** El BETWEEN(Entre) operador selecciona valores dentro de un rango dado. Los valores pueden ser números, texto o fechas.

El BETWEEN operador es inclusivo.

**Ejemplo:**

SELECT \* FROM Products  
WHERE Price **BETWEEN** 10 AND 20;

* **IN:** El IN operador le permite especificar múltiples valores en una WHERE cláusula.

El IN operador es una forma abreviada de múltiples OR condiciones**.**

**Ejemplo:**

SELECT \* FROM Customers  
WHERE Country **IN** ('Germany', 'France', 'UK');

* **LIKE:** Este operador se usa en una WHERE para buscar un patrón específico en una columna.

**Ejemplos:**

**Busca palabras o datos que inician con la letra** **a.**

SELECT \* FROM Customers  
WHERE CustomerName **LIKE** 'a%';

**Busca palabras o datos que finalizan con la letra a.**

SELECT \* FROM Customers  
WHERE CustomerName **LIKE** '%a';

**Busca palabras o datos que tienen la r en la segunda posición.**

SELECT \* FROM Customers  
WHERE CustomerName LIKE '\_r%';

* **IS NULL:** Le permite devolver un valor alternativo si una expresión es NULL:

**Ejemplo:**

SELECT ProductName, UnitPrice \* (UnitsInStock + **ISNULL**(UnitsOnOrder, 0))  
FROM Products;

* **HAVING:** Se agregó a SQL porque WHERE no se puede usar con funciones agregadas.

**Ejemplo:**

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

* **GROUP BY:** Esta declaración agrupa filas que tienen los mismos valores en filas de resumen.

**Ejemplo:**

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;

* **UNION:** Esteoperador se utiliza para combinar el conjunto de resultados de dos o más SELECT.

(**UNION**|**UNION ALL**)

**Ejemplo:**

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

* **INTERSECT:** Se utiliza para devolver los resultados de 2 o más instrucciones SELECT. Sin embargo, sólo devuelve las filas seleccionadas por todas las consultas o conjuntos de datos. Si un registro existe en una consulta y no en el otro, se omite de los resultados.

**Ejemplo:**

SELECT expression1, expression2, ... expression\_n

FROM tables

[WHERE conditions]

**INTERSECT**

SELECT expression1, expression2, ... expression\_n

FROM tables

[WHERE conditions];

**REFERENCIAS:**

* <https://www.w3schools.com/sql/sql_intro.asp>
* <https://www.ibm.com/docs/es/qmf/12.1.0?topic=ddfmtuss-creating-subquery-retrieve-data-from-more-than-one-table>
* <https://www.youtube.com/watch?v=nXmsP-YKnfo>