SQL sentencias

Seleccionar

```
SELECT column1, column2, ...
FROM table_name;
```

Aquí, column1, column2, ... son los nombres de campo de la tabla de la que desea seleccionar datos. Si desea seleccionar todos los campos disponibles en la tabla, use la siguiente sintaxis:

```
SELECT * FROM table_name;
```

La instrucción SELECT DISTINCT se usa para devolver solo valores distintos (diferentes).

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

La siguiente instrucción SQL selecciona solo los valores DISTINCT de la columna "País" en la tabla "Clientes":

```
SELECT DISTINCT Country FROM Customers;
```

La cláusula WHERE se usa para extraer solo aquellos registros que cumplen una condición específica.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

La siguiente instrucción SQL selecciona todos los clientes del país "México", en la tabla "Clientes":

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

SQL requiere comillas simples alrededor de valores de texto (la mayoría de los sistemas de bases de datos también permitirán dobles comillas).

Sin embargo, los campos numéricos no deben estar entre comillas:

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

La cláusula WHERE se puede combinar con los operadores AND, OR y NOT.

Los operadores AND y OR se utilizan para filtrar registros según más de una condición:

- **El operador Y muestra un registro si todas las condiciones separadas por Y son VERDADERAS.**
- **El operador OR muestra un registro si alguna de las condiciones separadas por OR es VERDADERA.**

El operador NOT muestra un registro si la condición (es) NO es VERDADERA.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

La siguiente instrucción SQL selecciona todos los campos de "Clientes" donde el país es "Alemania" Y la ciudad es "Berlín":

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

La siguiente instrucción SQL selecciona todos los campos de "Clientes" donde la ciudad es "Berlín" O "München":

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

La siguiente instrucción SQL selecciona todos los campos de "Clientes" donde el país es "Alemania" O "España":

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

La siguiente instrucción SQL selecciona todos los campos de "Clientes" donde el país NO es "Alemania":

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

También puede combinar los operadores AND, OR y NOT.

La siguiente instrucción SQL selecciona todos los campos de "Clientes" donde el país es "Alemania" Y la ciudad debe ser "Berlín" O "München" (use paréntesis para formar expresiones complejas):

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

La siguiente instrucción SQL selecciona todos los campos de "Clientes" donde el país NO es "Alemania" y NO "EE. UU.":

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

La palabra clave ORDER BY se utiliza para ordenar el conjunto de resultados en orden ascendente o descendente.

La palabra clave ORDER BY ordena los registros en orden ascendente de forma predeterminada. Para ordenar los registros en orden descendente, use la palabra clave DESC.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

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

```
SELECT * FROM Customers
ORDER BY Country;
```

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

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

La siguiente instrucción SQL selecciona todos los clientes de la tabla "Clientes", ordenados por la columna "País" y "Nombre del cliente". Esto significa que ordena por país, pero si algunas filas tienen el mismo país, las ordena por nombre de cliente:

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

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 de cliente":

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

La instrucción INSERT INTO se utiliza para insertar nuevos registros en una tabla.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Si agrega valores para todas las columnas de la tabla, no es necesario que especifique los nombres de las columnas en la consulta SQL. Sin embargo, asegúrese de que el orden de los valores esté en el mismo orden que las columnas de la tabla. La sintaxis INSERT INTO sería la siguiente:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

La siguiente instrucción SQL inserta un nuevo registro en la tabla "Clientes":

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

También es posible insertar solo datos en columnas específicas.

La siguiente instrucción SQL insertará un nuevo registro, pero solo insertará datos en las columnas "CustomerName", "City" y "Country" (CustomerID se actualizará automáticamente):

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

Un campo con un valor NULL es un campo sin valor.

No es posible probar valores NULL con operadores de comparación, como =, <o <>.

Tendremos que usar los operadores IS NULL y IS NOT NULL en su lugar.

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

El operador IS NULL se utiliza para probar valores vacíos (valores NULL).

El siguiente SQL enumera todos los clientes con un valor NULL en el campo "Dirección":

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

El operador IS NOT NULL se utiliza para probar valores no vacíos (valores NOT NULL).

El siguiente SQL enumera todos los clientes con un valor en el campo "Dirección":

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

La instrucción UPDATE se usa para modificar los registros existentes en una tabla.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

La siguiente instrucción SQL actualizará el nombre de contacto a "Juan" para todos los registros donde el país es "México":

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

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

```
DELETE FROM table_name WHERE condition;
```

Nota: ¡Tenga cuidado al eliminar registros en una tabla! Observe la cláusula WHERE en la instrucción DELETE. La cláusula WHERE especifica qué registro (s) deben eliminarse. Si omite la cláusula WHERE, se eliminarán todos los registros de la tabla.

La siguiente instrucción SQL elimina el cliente "Alfreds Futterkiste" de la tabla "Clientes":

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

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 table_name;
```

La siguiente instrucción SQL elimina todas las filas de la tabla "Clientes", sin eliminar la tabla:

```
DELETE FROM Customers;
```

La cláusula SELECT TOP se utiliza para especificar el número de registros a devolver.

La cláusula SELECT TOP es útil en tablas grandes con miles de registros. Devolver una gran cantidad de registros puede afectar el rendimiento.

Nota: No todos los sistemas de bases de datos admiten la cláusula SELECT TOP. MySQL admite la cláusula LIMIT para seleccionar un número limitado de registros, mientras que Oracle usa ROWNUM.

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

Sintaxis de MySQL:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

La siguiente instrucción SQL selecciona los primeros tres registros de la tabla "Clientes" (para SQL Server / MS Access):

```
SELECT TOP 3 * FROM Customers;
```

La siguiente declaración SQL muestra el ejemplo equivalente usando la cláusula LIMIT (para MySQL):

```
SELECT * FROM Customers
LIMIT 3;
```

La siguiente instrucción SQL selecciona el primer 50% de los registros de la tabla "Clientes" (para SQL Server / MS Access):

```
SELECT TOP 50 PERCENT * FROM Customers;
```

La siguiente instrucción SQL selecciona los primeros tres registros de la tabla "Clientes", donde el país es "Alemania" (para SQL Server / MS Access):

```
SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```

La siguiente declaración SQL muestra el ejemplo equivalente usando la cláusula LIMIT (para MySQL):

```sql
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

La función MIN () devuelve el valor más pequeño de la columna seleccionada.

```sql
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

La siguiente instrucción SQL encuentra el precio del producto más barato:

```sql
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

La función MAX () devuelve el valor más grande de la columna seleccionada.

```sql
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

La siguiente instrucción SQL encuentra el precio del producto más caro:

```sql
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

Las funciones SQL COUNT (), AVG () y SUM ()

La función COUNT () devuelve el número de filas que coincide con un criterio especificado.

```sql
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

La siguiente instrucción SQL encuentra el número de productos:

```
SELECT COUNT(ProductID)
FROM Products;
```

La función AVG () devuelve el valor promedio de una columna numérica.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

La siguiente instrucción SQL encuentra el precio promedio de todos los productos:

```
SELECT AVG(Price)
FROM Products;
```

La función SUM () devuelve la suma total de una columna numérica.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

La siguiente instrucción SQL busca la suma de los campos "Cantidad" en la tabla "Detalles del pedido":

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

El operador SQL LIKE

El operador LIKE se usa en una cláusula WHERE para buscar un patrón específico en una columna.

Hay dos comodines que se utilizan a menudo junto con el operador LIKE:

- %: El signo de porcentaje representa cero, uno o varios caracteres
- _ - El guión bajo representa un solo carácter

Nota: MS Access usa un asterisco (*) en lugar del signo de porcentaje (%) y un signo de interrogación (?) En lugar del guión bajo (_).

```sql
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

La siguiente instrucción SQL selecciona a todos los clientes con un CustomerName que comienza con "a":

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

La siguiente instrucción SQL selecciona todos los clientes con un CustomerName que termina con "a":

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

La siguiente declaración SQL selecciona todos los clientes con un CustomerName que tienen "o" en cualquier posición:

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

La siguiente declaración SQL selecciona todos los clientes con un CustomerName que tienen "r" en la segunda posición:

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

La siguiente instrucción SQL selecciona a todos los clientes con un CustomerName que comienza con "a" y tiene al menos 3 caracteres de longitud:

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

La siguiente instrucción SQL selecciona a todos los clientes con un ContactName que comienza con "a" y termina con "o":

```sql
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

La siguiente instrucción SQL selecciona a todos los clientes con un CustomerName que NO comienza con "a":

```sql
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Caracteres comodín en MS Access

Symbol	Description	Example
*	Represents zero or more characters	bl* finds bl, black, blue, and blob
?	Represents a single character	h?t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
!	Represents any character not in the brackets	h[!oa]t finds hit, but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

#	Represents any single numeric character	2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

Caracteres comodín en SQL Server

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

¡Todos los comodines también se pueden usar en combinaciones!

El operador IN le permite especificar varios valores en una cláusula WHERE.

El operador IN es una abreviatura de múltiples condiciones de OR.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

o:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

La siguiente instrucción SQL selecciona todos los clientes que se encuentran en "Alemania", "Francia" o "Reino Unido":

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

La siguiente instrucción SQL selecciona todos los clientes que NO se encuentran en "Alemania", "Francia" o "Reino Unido":

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

La siguiente declaración SQL selecciona todos los clientes que son de los mismos países que los proveedores:

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

El operador BETWEEN selecciona valores dentro de un rango determinado. Los valores pueden ser números, texto o fechas.

El operador BETWEEN es inclusivo: se incluyen los valores inicial y final.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

La siguiente instrucción SQL selecciona todos los productos con un precio ENTRE 10 y 20:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

La siguiente instrucción SQL selecciona todos los productos con un ProductName ENTRE Carnarvon Tigers y Mozzarella di Giovanni:

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di
Giovanni'
ORDER BY ProductName;
```

Para mostrar los productos fuera del rango del ejemplo anterior, use NOT BETWEEN:

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di
Giovanni'
ORDER BY ProductName;
```

La siguiente instrucción SQL selecciona todos los pedidos con un OrderDate ENTRE '01 -Julio-1996 'y '31 -Julio-1996':

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #01/07/1996# AND #31/07/1996#;
```

O

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

Los alias SQL se utilizan para dar un nombre temporal a una tabla, o una columna de una tabla.

Los alias se utilizan a menudo para hacer que los nombres de las columnas sean más legibles.

Un alias solo existe mientras dura la consulta.

```
SELECT column_name AS alias_name
FROM table_name;
```

La siguiente instrucción SQL crea dos alias, uno para la columna CustomerID y otro para la columna CustomerName:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Sintaxis de la tabla de alias

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

La siguiente instrucción SQL crea dos alias, uno para la columna CustomerName y otro para la columna ContactName. **Nota:** Requiere comillas dobles o corchetes si el nombre de alias contiene espacios:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

La siguiente instrucción SQL crea un alias llamado "Dirección" que combina cuatro columnas (Dirección, Código Postal, Ciudad y País):

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ',
' + Country AS Address
FROM Customers;
```

Para que la declaración SQL anterior funcione en MySQL, use lo siguiente:

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,',
',Country) AS Address
FROM Customers;
```

La siguiente instrucción SQL selecciona todos los pedidos del cliente con CustomerID = 4 (Around the Horn). Usamos las tablas "Clientes" y "Pedidos", y les damos los alias de tabla de "c" y "o" respectivamente (aquí usamos alias para hacer el SQL más corto):

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

La siguiente instrucción SQL es la misma que la anterior, pero sin alias:

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the
Horn' AND Customers.CustomerID=Orders.CustomerID;
```

SQL JOIN

Una cláusula JOIN se utiliza para combinar filas de dos o más tablas, según una columna relacionada entre ellas.

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

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 izquierda y los registros coincidentes de la tabla derecha
- **RIGHT (OUTER) JOIN**: Devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla 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.

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

La siguiente declaración SQL selecciona todos los pedidos con información del cliente:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

La siguiente declaración SQL selecciona todos los pedidos con información del cliente y del remitente:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

La palabra clave FULL OUTER JOIN devuelve todos los registros cuando hay una coincidencia en los registros de la tabla izquierda (tabla1) o derecha (tabla2). FULL OUTER JOIN y FULL JOIN son lo mismo.

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

La siguiente declaración SQL selecciona todos los clientes y todos los pedidos:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Nota: La palabra clave FULL OUTER JOIN devuelve todos los registros coincidentes de ambas tablas, independientemente de 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 enumerarán.

Un self JOIN es una combinación normal, pero la tabla se une a sí misma.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 y *T2* son alias de tabla diferentes para la misma tabla.

La siguiente declaración SQL coincide con los clientes que son de la misma ciudad:

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

El operador UNION se utiliza para combinar el conjunto de resultados de dos o más sentencias SELECT.

- Cada instrucción SELECT dentro de UNION debe tener el mismo número de columnas
- Las columnas también deben tener tipos de datos similares
- Las columnas de cada instrucción SELECT también deben estar en el mismo orden

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

La siguiente instrucción SQL devuelve las ciudades (solo valores distintos) de las tablas "Clientes" y "Proveedores":

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

Nota: si algunos clientes o proveedores tienen la misma ciudad, cada ciudad solo aparecerá una vez, porque UNION selecciona solo valores distintos. Utilice UNION ALL para seleccionar también valores duplicados.

La siguiente instrucción SQL devuelve las ciudades alemanas (solo valores distintos) de las tablas "Clientes" y "Proveedores":

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

El operador UNION selecciona solo valores distintos de forma predeterminada. Para permitir valores duplicados, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Nota: Los nombres de columna en el conjunto de resultados suelen ser iguales a los nombres de columna en la primera instrucción SELECT en UNION.

La siguiente instrucción SQL devuelve las ciudades alemanas (también valores duplicados) de la tabla "Clientes" y "Proveedores":

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

La siguiente declaración SQL enumera todos los clientes y proveedores:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

La instrucción GROUP BY agrupa las filas que tienen los mismos valores en filas de resumen, como "encontrar el número de clientes en cada país".

La instrucción GROUP BY se usa a menudo con funciones agregadas (COUNT, MAX, MIN, SUM, AVG) para agrupar el conjunto de resultados por una o más columnas.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

La siguiente declaración SQL enumera el número de clientes en cada país:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

La siguiente declaración SQL enumera el número de clientes en cada país, ordenados de mayor a menor:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

La siguiente declaración SQL enumera el número de pedidos enviados por cada remitente:

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

La cláusula HAVING se agregó a SQL porque la palabra clave WHERE no se pudo usar con funciones agregadas.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

La siguiente declaración SQL enumera el número de clientes en cada país. Incluya solo países con más de 5 clientes:

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

La siguiente declaración SQL enumera el número de clientes en cada país, ordenados de mayor a menor (solo incluye países con más de 5 clientes):

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

La siguiente declaración SQL enumera los empleados que han registrado más de 10 pedidos:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

La siguiente declaración SQL enumera si los empleados "Davolio" o "Fuller" han registrado más de 25 pedidos:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

El operador EXISTS se utiliza para probar la existencia de cualquier registro en una subconsulta.

El operador EXISTS devuelve verdadero si la subconsulta devuelve uno o más registros.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

La siguiente declaración SQL devuelve VERDADERO y enumera los proveedores con un precio de producto inferior a 20:

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.Supplie
rID = Suppliers.supplierID AND Price < 20);
```

Los operadores ANY y ALL se utilizan con una cláusula WHERE o HAVING.

El operador ANY devuelve verdadero si alguno de los valores de la subconsulta cumple la condición.

El operador ALL devuelve verdadero si todos los valores de la subconsulta cumplen la condición.

Cualquiera

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
(SELECT column_name FROM table_name WHERE condition);
```

todas

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
(SELECT column_name FROM table_name WHERE condition);
```

Nota: El *operador* debe ser un operador de comparación estándar (=, <>,! =,>,> =, <O <=).

El operador ANY devuelve TRUE si alguno de los valores de la subconsulta cumple la condición.

La siguiente instrucción SQL devuelve VERDADERO y enumera los nombres de los productos si encuentra CUALQUIER registro en la tabla Detalles del pedido que cantidad = 10:

```
SELECT ProductName
FROM Products
WHERE ProductID
= ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
```

La instrucción SELECT INTO copia datos de una tabla a una nueva tabla.

Copie todas las columnas en una nueva tabla:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Copie solo algunas columnas en una nueva tabla:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

La siguiente declaración SQL crea una copia de seguridad de los clientes:

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

La siguiente instrucción SQL usa la cláusula IN para copiar la tabla en una nueva tabla en otra base de datos:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

La siguiente instrucción SQL copia solo algunas columnas en una nueva tabla:

```sql
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

La siguiente instrucción SQL copia solo los clientes alemanes en una nueva tabla:

```sql
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

La siguiente instrucción SQL copia datos de más de una tabla en una nueva tabla:

```sql
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Sugerencia: SELECT INTO también se puede usar para crear una tabla nueva y vacía usando el esquema de otra. Simplemente agregue una cláusula WHERE que haga que la consulta no devuelva datos:

```sql
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

La instrucción INSERT INTO SELECT copia los datos de una tabla y los inserta en otra tabla.

- INSERT INTO SELECT requiere que los tipos de datos en las tablas de origen y destino coincidan
- Los registros existentes en la tabla de destino no se ven afectados

Copie todas las columnas de una tabla a otra tabla:

```sql
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copie solo algunas columnas de una tabla a otra tabla:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

La siguiente instrucción SQL copia "Proveedores" en "Clientes" (las columnas que no están llenas de datos contendrán NULL):

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

La siguiente instrucción SQL copia "Proveedores" en "Clientes" (complete todas las columnas):

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
SELECT SupplierName, ContactName, Address, City,
PostalCode, Country FROM Suppliers;
```

La siguiente instrucción SQL copia solo los proveedores alemanes en "Clientes":

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
```

La sentencia CASE pasa por condiciones y devuelve un valor cuando se cumple la primera condición (como una sentencia IF-THEN-ELSE). Entonces, una vez que una condición es verdadera, dejará de leer y devolverá el resultado. Si no se cumple ninguna condición, devuelve el valor de la cláusula ELSE.

Si no hay una parte ELSE y ninguna condición es verdadera, devuelve NULL.

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

El siguiente SQL pasa por condiciones y devuelve un valor cuando se cumple la primera condición:

```
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;
```

El siguiente SQL ordenará a los clientes por ciudad. Sin embargo, si la ciudad es NULL, ordene por país:

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

Un procedimiento almacenado es un código SQL preparado que puede guardar, por lo que el código se puede reutilizar una y otra vez.

Entonces, si tiene una consulta SQL que escribe una y otra vez, guárdela como un procedimiento almacenado y luego llámela para ejecutarla.

También puede pasar parámetros a un procedimiento almacenado, de modo que el procedimiento almacenado pueda actuar en función de los valores de parámetro que se pasan.

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Ejecutar un procedimiento almacenado

```
EXEC procedure_name;
```

La siguiente instrucción SQL crea un procedimiento almacenado llamado "SelectAllCustomers" que selecciona todos los registros de la tabla "Clientes":

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Ejecute el procedimiento almacenado anterior de la siguiente manera:

```
EXEC SelectAllCustomers;
```

La siguiente instrucción SQL crea un procedimiento almacenado que selecciona Clientes de una Ciudad en particular de la tabla "Clientes":

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;


EXEC SelectAllCustomers @City = 'London';
```

Configurar múltiples parámetros es muy fácil. Simplemente enumere cada parámetro y el tipo de datos separados por una coma como se muestra a continuación.

La siguiente instrucción SQL crea un procedimiento almacenado que selecciona Clientes de una Ciudad en particular con un Código Postal particular de la tabla "Clientes":

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode
nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode =
@PostalCode
GO;
```

```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```

El siguiente SQL crea una CLAVE EXTRANJERA en la columna "PersonID" cuando se crea la tabla "Pedidos":

MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Acceso a SQL Server / Oracle / MS:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

Para permitir el nombramiento de una restricción FOREIGN KEY y para definir una restricción FOREIGN KEY en varias columnas, utilice la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

Para crear una restricción FOREIGN KEY en la columna "PersonID" cuando la tabla "Pedidos" ya está creada, use el siguiente SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Para permitir el nombramiento de una restricción FOREIGN KEY y para definir una restricción FOREIGN KEY en varias columnas, utilice la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Para eliminar una restricción FOREIGN KEY, use el siguiente SQL:

MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

Acceso a SQL Server / Oracle / MS:

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

SQL Server viene con los siguientes tipos de datos para almacenar una fecha o un valor de fecha / hora en la base de datos:

- FECHA - formato AAAA-MM-DD
- DATETIME - formato: AAAA-MM-DD HH: MI: SS
- SMALLDATETIME - formato: AAAA-MM-DD HH: MI: SS
- TIMESTAMP - formato: un número único

Nota: i Los tipos de fecha se eligen para una columna cuando crea una nueva tabla en su base de datos!

En SQL, una vista es una tabla virtual basada en el conjunto de resultados de una declaración SQL.

Una vista contiene filas y columnas, como una tabla real. Los campos de una vista son campos de una o más tablas reales de la base de datos.

Puede agregar funciones SQL, WHERE y declaraciones JOIN a una vista y presentar los datos como si vinieran de una sola tabla.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Nota: i Una vista siempre muestra datos actualizados! El motor de la base de datos recrea los datos, utilizando la declaración SQL de la vista, cada vez que un usuario consulta una vista.

El siguiente SQL crea una vista que muestra todos los clientes de Brasil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

Podemos consultar la vista anterior de la siguiente manera:

```
SELECT * FROM [Brazil Customers];
```

El siguiente SQL crea una vista que selecciona todos los productos de la tabla "Productos" con un precio superior al precio medio:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

```
SELECT * FROM [Products Above Average Price];
```

Una vista se puede actualizar con el comando CREAR O REEMPLAZAR VISTA.

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

El siguiente SQL agrega la columna "Ciudad" a la vista "Clientes de Brasil":

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

Una vista se elimina con el comando DROP VIEW.

```
DROP VIEW view_name;
```

La instrucción BACKUP DATABASE se utiliza en SQL Server para crear una copia de seguridad completa de una base de datos SQL existente.

```
BACKUP DATABASE databasename
TO DISK = 'filepath';
```

Una copia de seguridad diferencial solo hace una copia de seguridad de las partes de la base de datos que han cambiado desde la última copia de seguridad completa de la base de datos.

```
BACKUP DATABASE databasename
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

La siguiente declaración SQL crea una copia de seguridad completa de la base de datos existente "testDB" en el disco D:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```

La siguiente instrucción SQL crea una copia de seguridad diferencial de la base de datos "testDB":

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
WITH DIFFERENTIAL;
```

La instrucción CREATE TABLE se utiliza para crear una nueva tabla en una base de datos.

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Los parámetros de columna especifican los nombres de las columnas de la tabla.

El parámetro de tipo de datos especifica el tipo de datos que la columna puede contener (por ejemplo, varchar, integer, date, etc.).

El siguiente ejemplo crea una tabla llamada "Personas" que contiene cinco columnas: PersonID, LastName, FirstName, Address y City:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

También se puede crear una copia de una tabla existente usando CREATE TABLE.

La nueva tabla obtiene las mismas definiciones de columna. Se pueden seleccionar todas las columnas o columnas específicas.

Si crea una nueva tabla utilizando una tabla existente, la nueva tabla se completará con los valores existentes de la tabla anterior.

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

El siguiente SQL crea una nueva tabla llamada "TestTables" (que es una copia de la tabla "Clientes"):

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

La instrucción DROP TABLE se utiliza para eliminar una tabla existente en una base de datos.

```
DROP TABLE table_name;
```

Nota : tenga cuidado antes de borrar una tabla. ¡Eliminar una tabla resultará en la pérdida de la información completa almacenada en la tabla!

La siguiente instrucción SQL elimina la tabla existente "Transportistas":

```
DROP TABLE Shippers;
```

La instrucción ALTER TABLE se usa para agregar, eliminar o modificar columnas en una tabla existente.

La instrucción ALTER TABLE también se usa para agregar y eliminar varias restricciones en una tabla existente.

Para agregar una columna en una tabla, use la siguiente sintaxis:

```
ALTER TABLE table_name
ADD column_name datatype;
```

El siguiente SQL agrega una columna "Correo electrónico" a la tabla "Clientes":

```
ALTER TABLE Customers
ADD Email varchar(255);
```

Para eliminar una columna de una tabla, utilice la siguiente sintaxis (observe que algunos sistemas de bases de datos no permiten eliminar una columna):

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

El siguiente SQL elimina la columna "Correo electrónico" de la tabla "Clientes":

```
ALTER TABLE Customers
DROP COLUMN Email;
```

Para cambiar el tipo de datos de una columna en una tabla, use la siguiente sintaxis:

Acceso a SQL Server / MS:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

Ahora queremos agregar una columna llamada "DateOfBirth" en la tabla "Personas".

Usamos la siguiente declaración SQL:

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

Ahora queremos cambiar el tipo de datos de la columna denominada "DateOfBirth" en la tabla "Personas".

Usamos la siguiente declaración SQL:

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

A continuación, queremos eliminar la columna llamada "DateOfBirth" en la tabla "Personas".

Usamos la siguiente declaración SQL:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```