The SELECT DISTINCT statement is used to return only distinct (different) values.

SELECT DISTINCT Country FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

SELECT Count(\*) AS DistinctCountries  
FROM (SELECT DISTINCT Country FROM Customers);

WHERE clause is used to filter records.

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

SELECT \* FROM Customers  
WHERE CustomerID=1;

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

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

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

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

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

SELECT \* FROM Customers  
ORDER BY Country;

SELECT \* FROM Customers  
ORDER BY Country DESC;

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

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

The INSERT INTO statement is used to insert new records in a table.

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

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

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

The UPDATE statement is used to modify the existing records in a table.

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

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

The DELETE statement is used to delete existing records in a table.

DELETE FROM table\_name WHERE condition;

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

SELECT TOP 3 \* FROM Customers;

SELECT \* FROM Customers  
LIMIT 3;

SELECT \* FROM Customers  
WHERE ROWNUM <= 3;

SELECT TOP 50 PERCENT \* FROM Customers;

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

SELECT \* FROM Customers  
WHERE Country='Germany' AND ROWNUM <= 3;

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

SELECT MAX(Price) AS LargestPrice  
FROM Products;

The COUNT() function returns the number of rows that matches a specified criterion.

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

SELECT COUNT(ProductID)  
FROM Products;

SELECT AVG(Price)  
FROM Products;

SELECT SUM(Quantity)  
FROM OrderDetails;

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

|  |  |
| --- | --- |
| **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" |

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

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

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

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

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

SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the [SQL LIKE](https://www.w3schools.com/sql/sql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Wildcard Characters in 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 |

Wildcard Characters in 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 |

All the wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends 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 starts with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a" and ends with "o" |

SELECT \* FROM Customers  
WHERE City LIKE 'ber%';

SELECT \* FROM Customers  
WHERE City LIKE '%es%';

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

Example

SELECT \* FROM Customers  
WHERE City LIKE 'L\_n\_on';

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

Example

SELECT \* FROM Customers  
WHERE City LIKE '[bsp]%';

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

Example

SELECT \* FROM Customers  
WHERE City LIKE '[a-c]%';

The two following SQL statements select all customers with a City NOT starting with "b", "s", or "p":

Example

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

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

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

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

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

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

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20  
AND CategoryID NOT IN (1,2,3);

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

SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

### Example

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

Here are the different types of the JOINs in SQL:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

The INNER JOIN keyword selects records that have matching values in both tables.

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;

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

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);

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

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

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
ORDER BY Orders.OrderID;

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

**Note:** FULL OUTER JOIN can potentially return very large result-sets!

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

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

A self JOIN is a regular join, but the table is joined with itself.

### Self JOIN Syntax

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

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;

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Each SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in each SELECT statement must also be in the same order

### UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

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

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)ORDER BY column\_name(s);

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

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

### HAVING Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

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

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;