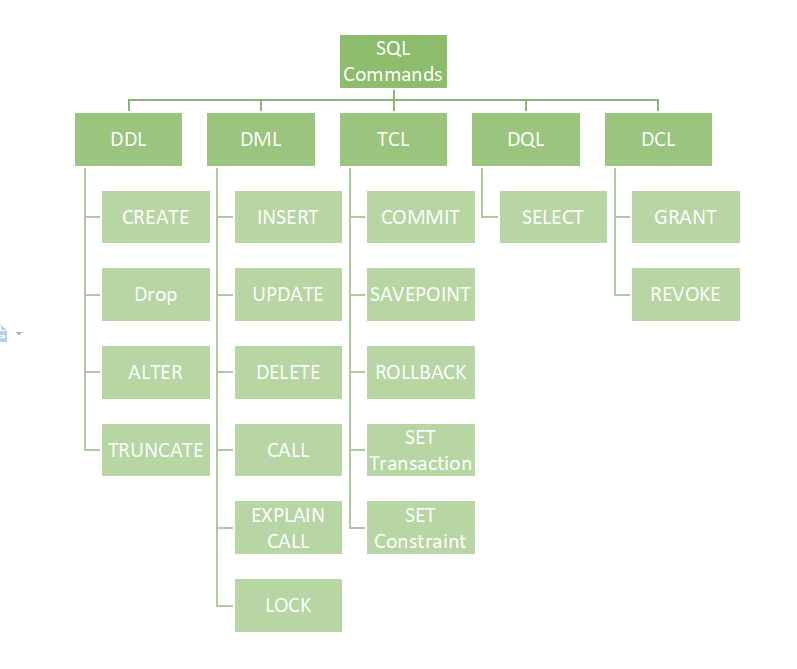
These [SQL](https://www.geeksforgeeks.org/sql-concepts-and-queries/)commands are mainly categorized into five categories:

1. DDL – Data Definition Language
   * Create
   * Drop
   * Alter
   * Truncate
2. DQL – Data Query Language
   * Select
3. DML – Data Manipulation Language
   * Insert
   * Update
   * Delete
   * Lock
4. DCL – Data Control Language
   * Grant
   * Revoke
5. TCL – Transaction Control Language
   * Commit
   * Save point
   * Rollback
   * Set Transaction
   * Set Constraint



The SQL SELECT DISTINCT Statement

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

## Count Distinct

By using the DISTINCT keyword in a function called COUNT, we can return the number of different countries.

SELECT COUNT(DISTINCT Country) FROM Customers;

**Note:**If we are not use distinct keyword in this query, will get wrong answer due to entry of same country name

## The SQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

### **Example**

Select all customers with a CustomerID greater than 80:

SELECT \* FROM Customers  
WHERE CustomerID > 80;

## The SQL ORDER BY

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

## ORDER BY Several Columns

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

### **Example**

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

## Using Both ASC and DESC

The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:

### **Example**

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

## The SQL AND Operator

The WHERE clause can contain one or many AND operators.

The AND operator is used to filter records based on more than one condition, like if you want to return all customers from Spain that starts with the letter 'G':

### **Example**[**Get your own SQL Server**](https://www.w3schools.com/spaces/)

Select all customers from Spain that starts with the letter 'G':

SELECT \*  
FROM Customers  
WHERE Country = 'Spain' AND CustomerName LIKE 'G%';

## AND vs OR

The AND operator displays a record if all the conditions are TRUE.

The OR operator displays a record if any of the conditions are TRUE.

## All Conditions Must Be True

The following SQL statement selects all fields from Customers where Country is "Germany" AND City is "Berlin" AND PostalCode is higher than 12000:

SELECT \* FROM Customers  
WHERE Country = 'Germany'  
AND City = 'Berlin'  
AND PostalCode > 12000;

## Combining AND and OR

You can combine the AND and OR operators.

The following SQL statement selects all customers from Spain that starts with a "G" or an "R".

Make sure you use parenthesis to get the correct result.

### **Example**

Select all Spanish customers that starts with either "G" or "R":

SELECT \* FROM Customers  
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');

### **Example**

Select all customers that either:  
are from Spain and starts with either "G", or  
starts with the letter "R":

SELECT \* FROM Customers  
WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';

**Note**: used to refer to a situation in which there is a choice between two different plans of action, but both together are not possible so we can use above Query

## At Least One Condition Must Be True

The following SQL statement selects all fields from Customers where either City is "Berlin", CustomerName starts with the letter "G" or Country is "Norway":

### **Example**

SELECT \* FROM Customers  
WHERE City = 'Berlin' OR CustomerName LIKE 'G%' OR Country = 'Norway';

## The NOT Operator

The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.

### **Example**

Select only the customers that are NOT from Spain:

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

## NOT LIKE

### **Example**

Select customers that does not start with the letter 'A':

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

## NOT BETWEEN

### **Example**

Select customers with a customerID not between 10 and 60:

SELECT \* FROM Customers  
WHERE CustomerID NOT BETWEEN 10 AND 60;

## NOT IN

### **Example**

Select customers that are not from Paris or London:

SELECT \* FROM Customers  
WHERE City NOT IN ('Paris', 'London');

## NOT Greater Than

### **Example**

Select customers with a CustomerId not greater than 50:

SELECT \* FROM Customers  
WHERE NOT CustomerID > 50;

**Note:** There is a not-greater-then operator: !> that would give you the same result.

## The SQL INSERT INTO Statement

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

### **INSERT INTO Syntax**

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO table\_name  
VALUES (value1, value2, value3, ...);

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');

## Insert Multiple Rows

It is also possible to insert multiple rows in one statement.

To insert multiple rows of data, we use the same INSERT INTO statement, but with multiple values:

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
VALUES  
('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),  
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),  
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');

**Note:** Make sure you separate each set of values with a comma ,

## What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

**Note:** A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

## How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

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

# **SQL TOP, LIMIT, FETCH FIRST or ROWNUM Clause**

## The SQL SELECT TOP Clause

**Note:** Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.

**SQL Server / MS Access Syntax:**

SELECT TOP number|*percent* column\_name(s)  
FROM table\_nameWHERE condition;

**MySQL Syntax:**

SELECT column\_name(s)  
FROM table\_nameWHERE condition  
LIMIT number;

**SQL Server / MS Access Syntax:**

SELECT TOP number|*percent* column\_name(s)  
FROM table\_nameWHERE condition;

**MySQL Syntax:**

SELECT column\_name(s)  
FROM table\_nameWHERE condition  
LIMIT number;

**Oracle 12 Syntax:**

SELECT column\_name(s)  
FROM table\_nameORDER BY column\_name(s)  
FETCH FIRST number ROWS ONLY;

**Older Oracle Syntax:**

SELECT column\_name(s)  
FROM table\_name  
WHERE ROWNUM <= number;

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany" (for SQL Server/MS Access):

### **Example**

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

The following SQL statement shows the equivalent example for MySQL:

### **Example**

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

The following SQL statement shows the equivalent example for Oracle:

### **Example**

SELECT \* FROM Customers  
WHERE Country='Germany'  
FETCH FIRST 3 ROWS ONLY;

## The SQL LIKE Operator

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 sign \_ represents one, single character

### **Example**

Return all customers from a city that contains the letter 'L':

SELECT \* FROM Customers  
WHERE city LIKE '%L%';

### **Example**

Return all customers that starts with 'a' or starts with 'b':

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

Return all customers that starts with "b" and ends with "s":

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'b%s';

### **Example**

Return all customers that contains the phrase 'or'

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

### **Example**

Return all customers that starts with "a" and are at least 3 characters in length:

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

Return all customers that have "r" in the second position:

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

## Wildcard Characters

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| % | Represents zero or more characters |
| \_ | Represents a single character |
| [] | Represents any single character within the brackets \* |
| ^ | Represents any character not in the brackets \* |
| - | Represents any single character within the specified range \* |
| {} | Represents any escaped character \*\* |

\* Not supported in PostgreSQL and MySQL databases.

\*\* Supported only in Oracle databases.

Return all customers starting with either "b", "s", or "p":

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

### **Example**

Return all customers starting with "a", "b", "c", "d", "e" or "f":

SELECT \* FROM Customers  
WHERE CustomerName LIKE '[a-f]%';

## SQL JOIN

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

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

Different Types of SQL JOINs

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

**Note:** The INNER JOIN keyword returns only rows with a match in both tables. Which means that if you have a product with no CategoryID, or with a CategoryID that is not present in the Categories table, that record would not be returned in the result.

## JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

### **Example**

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);

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

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

## SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

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

## SQL FULL OUTER JOIN Keyword

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

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

**Note:** The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

## SQL Self Join

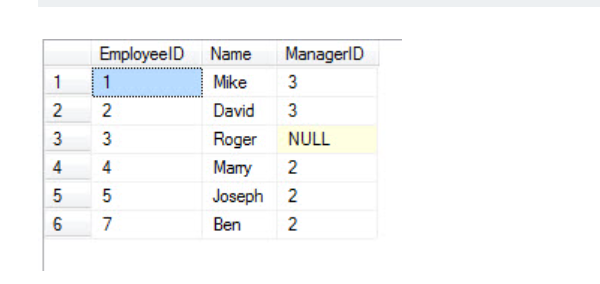
A self join is a regular join, but the table is joined with itself.

### **Self Join Syntax**

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

T1 and T2 are different table aliases for the same table.

SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City  
FROM Customers A, Customers B  
WHERE  A.City = B.City  
ORDER BY A.City;



-- Inner Join

SELECT e1.Name EmployeeName, e2.name AS ManagerName

FROM Employee e1

INNER JOIN Employee e2

ON e1.ManagerID = e2.EmployeeID

GO

## The SQL UNION Operator

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

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

## The SQL GROUP BY Statement

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.

The following SQL statement lists the number of customers in each country, sorted high to low:

### **Example**

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

## GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

### **Example**

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

## The SQL HAVING Clause

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

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

### **Example**

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

The following SQL statement lists the employees that have registered more than 10 orders:

### **Example**

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;

## The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

The following SQL statement returns TRUE and lists the suppliers with a product price equal to 22:

### **Example**

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

## The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

## The SQL ANY Operator

The ANY operator:

* returns a boolean value as a result
* returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 99 (this will return TRUE because the Quantity column has some values larger than 99):

### **Example**

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY  
  (SELECT ProductID  
  FROM OrderDetails  
  WHERE Quantity > 99);

## SQL ALL Examples

The following SQL statement lists ALL the product names:

The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10):

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

## The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

SQL SELECT INTO Examples

The following SQL statement creates a backup copy of Customers:

SELECT \* INTO CustomersBackup2017  
FROM Customers;

The following SQL statement uses the IN clause to copy the table into a new table in another database:

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

The following SQL statement copies only a few columns into a new table:

SELECT CustomerName, ContactName INTO CustomersBackup2017  
FROM Customers;

The following SQL statement copies only the German customers into a new table:

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

The following SQL statement copies data from more than one table into a new table:

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

**Tip:** SELECT INTO can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

SELECT \* INTO *newtable*  
FROM *oldtable*  
WHERE 1 = 0;

## The SQL CASE Expression

The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

The following SQL goes through conditions and returns a value when the first condition is met:

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;

The following SQL will order the customers by City. However, if City is NULL, then order by Country:

### **Example**

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

## What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

### **Example**

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

## SQL Coalesce Function

The SQL server's Coalesce function is used to handle the Null values. The null values are replaced with user-defined values during the expression evaluation process. This function evaluates arguments in a particular order from the provided arguments list and always returns the first non-null value.

COALESCE(value\_1, value\_2, ...., value\_n)

The COALESCE() function takes in at least one value (value\_1). It will return the first value in the list that is non-null.

For example, it will first check if value\_1 is null. If not, then it returns value\_1. Otherwise, it checks if value\_2 is null. The process goes on until the list is complete.