श्री राधा 🌹

SQL

LinkedIn : [Mr. Gresh Sehrawat]

SELECT Syntax

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

```
SELECT * FROM table_name;
```

SELECT DISTINCT Syntax

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

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

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

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

WHERE Syntax

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

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

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

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern

IN	To specify multiple possible values for a column

AND Syntax

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

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

OR Syntax

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

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

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

NOT Syntax

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

NOT Example

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

Combining AND, OR and NOT

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

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

The SQL ORDER BY Keyword

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

```
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;
```

INSERT INTO Syntax

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

```
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');
```

IS NULL Syntax

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

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

IS NOT NULL Syntax

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

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

UPDATE Syntax

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

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

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

Update Warning!

Be careful when updating records. If you omit the `WHERE` clause, ALL records will be updated!

Example

```sql
UPDATE Customers
SET ContactName='Juan';
```

DELETE Syntax

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

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

Delete All Records

```sql
DELETE FROM table_name;
```

```sql
DELETE FROM Customers;
```

SQL TOP, LIMIT and FETCH FIRST Examples

```
SELECT TOP 3 * FROM Customers;

SELECT * FROM Customers LIMIT 3;

SELECT * FROM Customers FETCH FIRST 3 ROWS ONLY;

SELECT TOP 50 PERCENT * FROM Customers;

SELECT * FROM Customers FETCH FIRST 50 PERCENT ROWS ONLY;


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

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


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

MIN() Syntax

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


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

MAX() Syntax

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


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

COUNT() Syntax

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

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

AVG() Syntax

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

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

SUM() Syntax

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

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

Note: NULL values are ignored.

The SQL LIKE Operator

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

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 NOT LIKE 'a%';
```

IN Syntax

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

or:

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

```
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);
```

BETWEEN Syntax

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

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

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

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

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

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

Alias Column Syntax

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

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

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

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

Alias Table Syntax

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

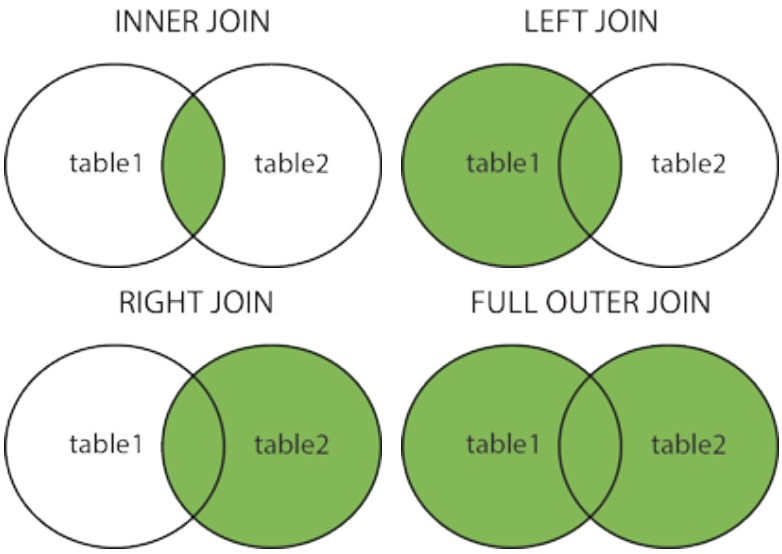
```sql
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

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



INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

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

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON 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;
```

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON 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;
```

FULL OUTER JOIN Syntax

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

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

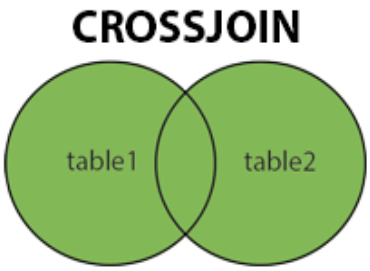
Self Join Syntax

```
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;
```

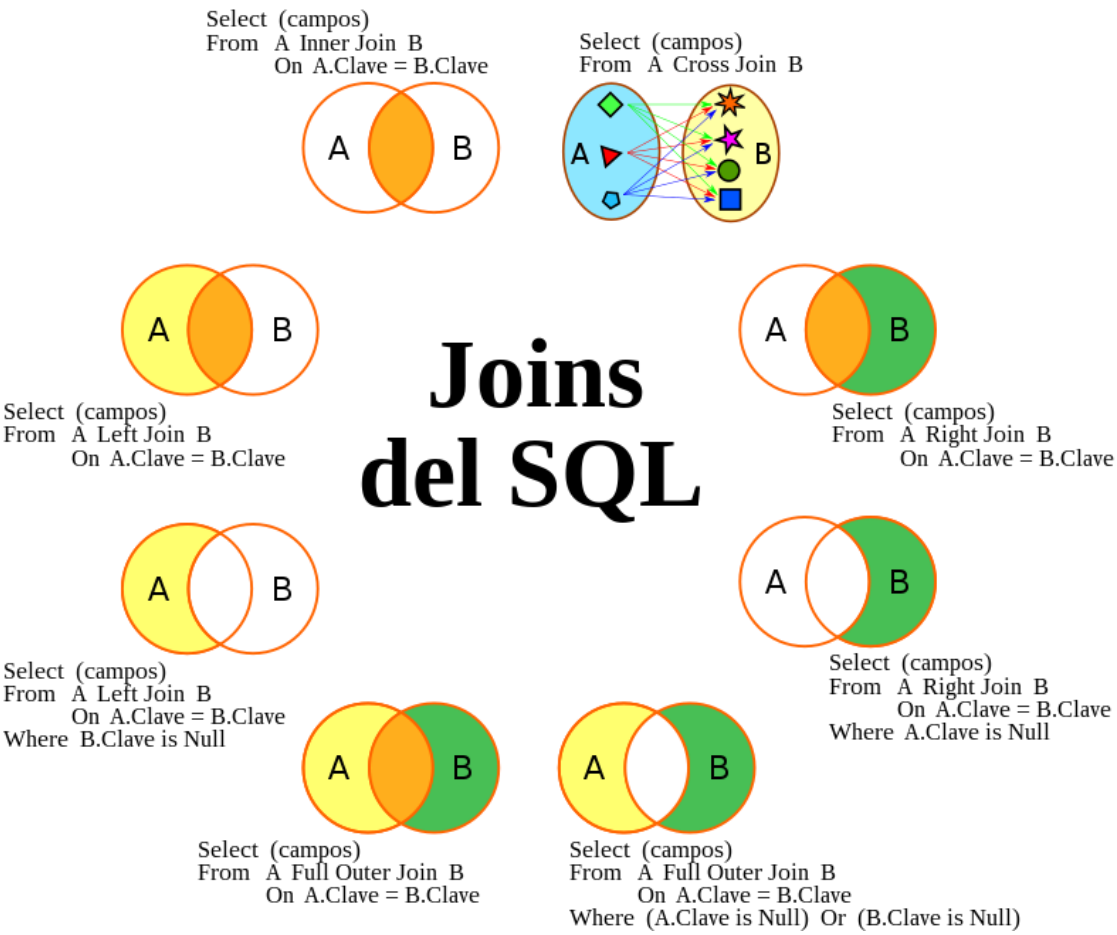
Natural Join Syntax

```
SELECT * FROM table1 NATURAL JOIN table2;
```



CROSS JOIN Syntax

```sql
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```



Subqueries can be categorized into two types: A noncorrelated (simple) subquery obtains its results independently of its containing (outer) statement. **A correlated subquery requires values from its outer query in order to execute**.

Uncorrelated Sub-query Example

Let us execute a uncorrelated sub-query which retrieves records of all the students who belong to “Computer” department.

```
USE schooldb;

SELECT * FROM
  student
  WHERE dep_id =
  (
    SELECT id from department WHERE name = 'Computer'
  );
```

The output of the above SQL will be:

id	name	gender	age	dep_id
4	Laura	Female	18	2
6	Kate	Female	22	2
7	Joseph	Male	18	2
10	Elis	Female	27	2

You can see that there are two queries. The inner query retrieves id of the “Computer” department while the outer query retrieves student records with that id value in the dep_id column.

We know that in the case of uncorrelated sub-queries the inner query can be executed as standalone query and it will still work. Let’s check if this is true in this case. Execute the following query on the server.

```
SELECT id from department WHERE name = 'Computer';
```

The above query will execute successfully and will return 2 i.e. the of the “Computer” department. This is a uncorrelated sub-query.

Correlated Sub-query Example

We know that in case of correlated sub-queries, the inner query depends upon the outer query and cannot be executed as a standalone query.

Lets execute a correlated sub-query that retrieves results of all the students with age greater than average age within their department as discussed above.

```
USE schooldb;

SELECT   name, gender, age
  FROM     student Greater
  WHERE    age >
  (SELECT   AVG (age)
     FROM     student average
     WHERE      greater.dep_id = average.dep_id) ;
```

The output of the above query will be:

name	gender	age
Kate	Female	22
Elis	Female	27
Jon	Male	22
Sara	Female	25

We know that in the case of a correlated sub-query, the inner query cannot be executed as standalone query. You can verify this by executing the following inner query on it's own:

```
1 SELECT   AVG (age)
2   FROM     student average
3   WHERE      greater.dep_id = average.dep_id
```

The above query will throw an error.

Other small differences between correlated and uncorrelated sub-queries are:

1. The outer query executes before the inner query in the case of a correlated sub-query. On the other hand in case of a uncorrelated sub-query the inner query executes before the outer query.
2. Correlated sub-queries are slower. They take M x N steps to execute a query where M is the records retrieved by outer query and N is the number of iteration of inner query. Uncorrelated sub-queries complete execution in M + N steps.

SQL UNION Operator

UNION Syntax

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

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

UNION ALL Syntax

```
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 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.

GROUP BY Syntax

```
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;
```

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

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

```
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;
```

```
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;
```

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.

EXISTS Syntax

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

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

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

The SQL HAVING Clause

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

HAVING Syntax

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

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

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

Reference : W3Schools

Thank You !