**SELECT Statement in SQL**

The SELECT statement in SQL is used to fetch or retrieve data from a database. It allows users to access the data and retrieve specific data based on specific conditions.

We can fetch either the entire table or according to some specified rules. The data returned is stored in a result table. This result table is also called the **result set.**With the SELECT clause of a SELECT command statement, we specify the columns that we want to be displayed in the query result and, optionally, which column headings we prefer to see above the result table.

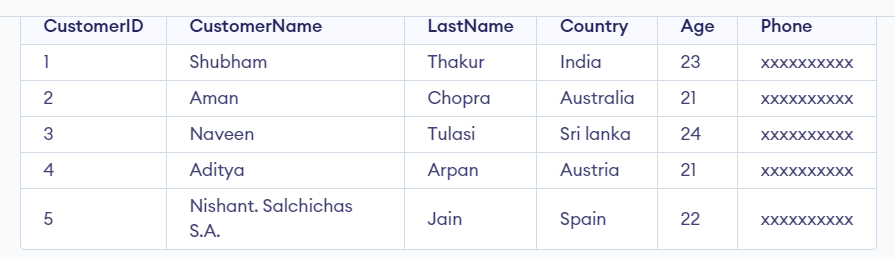
The SELECT clause is the first clause and is one of the last clauses of the select statement that the database server evaluates. The reason for this is that before we can determine what to include in the final result set, we need to know all of the possible columns that could be included in the final result set.

**Syntax**

CREATE TABLE Customer(  
 CustomerID INT PRIMARY KEY,  
 CustomerName VARCHAR(50),  
 LastName VARCHAR(50),  
 Country VARCHAR(50),  
 Age int(2),  
 Phone int(10)  
);

-- Insert some sample data into the Customers table  
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)  
VALUES (1, 'Shubham', 'Thakur', 'India','23','xxxxxxxxxx'),  
 (2, 'Aman ', 'Chopra', 'Australia','21','xxxxxxxxxx'),  
 (3, 'Naveen', 'Tulasi', 'Sri lanka','24','xxxxxxxxxx'),  
 (4, 'Aditya', 'Arpan', 'Austria','21','xxxxxxxxxx'),  
 (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain','22','xxxxxxxxxx');

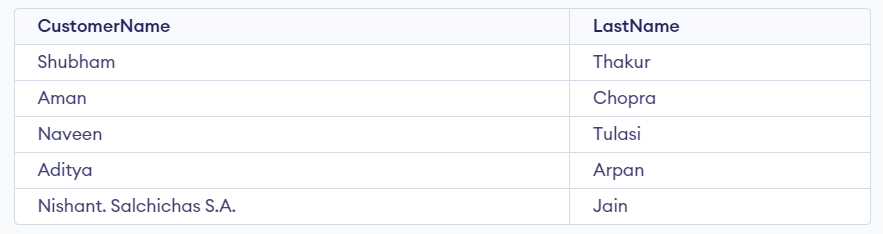
**Output:**



**Retrieve Data using SELECT Query**

In this example, we will fetch CustomerName, LastName from the table Customer:

**SELECT** CustomerName, LastName **FROM** Customer;

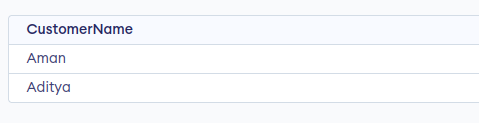


**SELECT Statement with WHERE Clause**

Suppose we want to see table values with specific conditions then [**WHERE Clause**](https://www.geeksforgeeks.org/sql-where-clause) is used with select statement.

**Query:**

SELECT CustomerName FROM Customer where Age = '21';

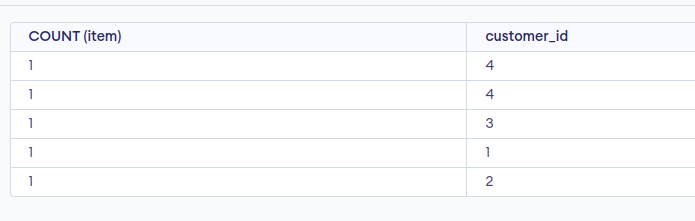


**SQL SELECT Statement with GROUP BY Clause**

In this example, we will use SELECT statement with [**GROUP BY**](https://www.geeksforgeeks.org/sql-group-by) Clause

**Query:**

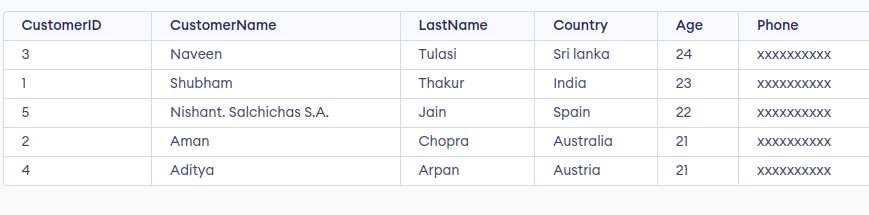
**SELECT COUNT** (item), Customer\_id **FROM** Orders **GROUP BY** order\_id;



**SELECT Statement with ORDER BY clause in SQL**

In this example, we will use SELECT Statement with [**ORDER BY**](https://www.geeksforgeeks.org/sql-order-by) clause

**SELECT** \* **FROM** Customer **ORDER BY** Age **DESC**;



The project SQL operation allows users of the relational model to retrieve column-specific data from a table. This data is then used to create a new table that is dedicated to the information that the user would like to see.

So, if you had a relational model consisting of nine different columns but you only need the name and the date of birth for each individual in the table, you would use a project operation to retrieve this data.

**Project Operation Structure**

**Select** column\_name **from** table\_name

The project operation has a pretty straightforward structure, consisting of exactly four parts.

* The **Select** keyword, which should always begin with a capital letter.
* The column name/s, if there is more than one each should be separated from the other with a comma.
* The **from** keyword, which is all lower case.
* The table name.

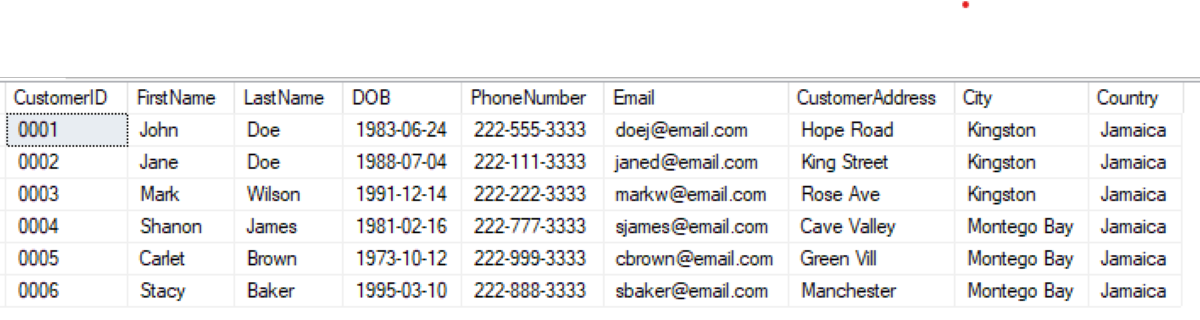
**Using the Project Operation on a Table**

Imagine a furniture store that has a relational database management system.

In this database, a customer table that stores all the data we have on each customer. In the customer table are nine fields:

1. CustomerID
2. FirstName
3. LastName
4. DOB
5. PhoneNumber
6. Email
7. CustomerAddress
8. City
9. Country

**Customer Table Example**



One day the customer relations officer comes up with a brilliant idea that is aimed at improving customer relationship.

The idea is to get the software developer to create a simple automated program that will email each customer on their birthday.

So now you need exactly four fields of data from our customer table: FirstName and LastName, to personalize the email; DOB, to know the date to schedule the email on; and Email.

One day the customer relations officer comes up with a brilliant idea that is aimed at improving customer relationship.

The idea is to get the software developer to create a simple automated program that will email each customer on their birthday.

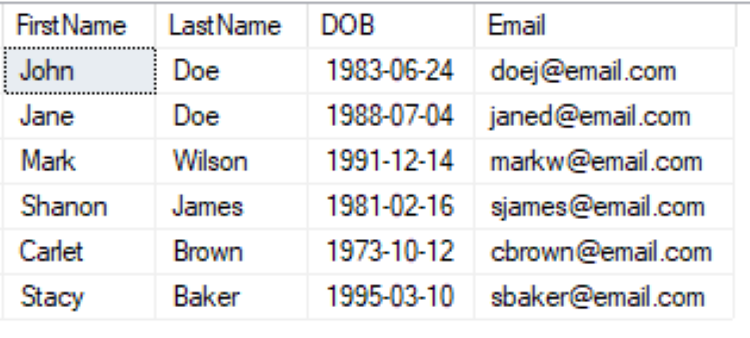
So now you need exactly four fields of data from our customer table: FirstName and LastName, to personalize the email; DOB, to know the date to schedule the email on; and Email.

**Using the Project Operation Example**

**Select** FirstName, LastName, DOB, Email **from** Customer

The code above will effectively generate a new table that can be used to create a simple program. The table that was generated can be seen below.

**Customers Birthday Table Example**



**SQL Arithmetic Operators**

In Structured Query Language, the arithmetic operators are used to perform mathematical operations on the numerical values stored in the database tables.

We can use these operators with the SELECT statement in SQL. We can also use the WHERE clause in the SELECT statement for performing operations on particular rows.

These types of operators are used between two numerical operands for performing addition, subtraction,

**The arithmetic operators in SQL are categorized into the following five types:**multiplication, and division operations.

1. SQL Addition Operator (+)
2. SQL Subtraction Operator (-)
3. SQL Multiplication Operator (\*)
4. SQL Division Operator (/)
5. SQL Modulus Operator (%)

### **SQL Addition Operator (+)**

The SQL Addition Operator performs the addition on the numerical columns in the table.

If you want to add the values of two numerical columns in the table, then you have to specify both columns as the first and second operand. You can also add the new integer value in the value of the integer column.

**Syntax of SQL Addition Operator:**

1. **SELECT** Column\_Name\_1 Addition\_Operator Column\_Name2 **FROM** Table\_Name;

Addition Operator with WHERE Clause

The addition operator can also be used with the WHERE clause in the SQL SELECT query.

**The syntax for using the WHERE clause with the addition operator is given below:**

1. **SELECT** Column\_Name\_1 Addition\_Operator Column\_Name2 **FROM** Table\_Name **WHERE** Condition;

**Implementation of Addition operator in SQL:**

The following CREATE query creates the Employee table with five fields:

1. **CREATE** **TABLE** Employee
2. (
3. Employee\_ID **INT** AUTO\_INCREMENT **PRIMARY** **KEY**,
4. Emp\_Name **VARCHAR** (50),
5. Emp\_City **VARCHAR** (20),
6. Emp\_Salary **INT** NOT NULL,
7. Emp\_Bonus **INT** NOT NULL
8. ) ;

The following INSERT query inserts the record of employees into the Employee table:

1. INSERT **INTO** Employee (Employee\_ID, Emp\_Name, Emp\_City, Emp\_Salary, Emp\_Bonus) **VALUES** (101, Anuj, Ghaziabad, 25000, 2000),
2. (102, Tushar, Lucknow, 29000, 1000),
3. (103, Vivek, Kolkata, 35000, 2500),
4. (104, Shivam, Goa, 22000, 3000);

The following SELECT query shows the data of **the Employee** table:

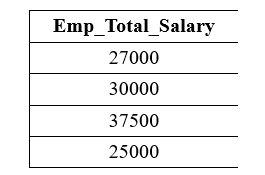
1. SELECT \* **FROM** Employee;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Employee\_Id** | **Emp\_Name** | **Emp\_City** | **Emp\_Salary** | **Emp\_bonus** |
| 101 | Anuj | Ghaziabad | 25000 | 2000 |
| 102 | Tushar | Lucknow | 29000 | 1000 |
| 103 | Vivek | Kolkata | 35000 | 2500 |
| 104 | Shivam | Goa | 22000 | 3000 |

The following query adds the Emp\_Salary and Emp\_Bonus of each employee of the **Employee** table using the addition operator:

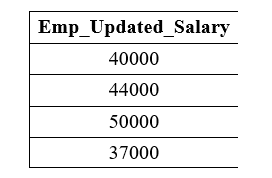
1. **SELECT** Emp\_Salary + Emp\_Bonus **AS** Emp\_Total\_Salary **FROM** Employee;

**Output:**



The following query adds 15000 to the salary of each employee in the Emp\_Salary column of **the Employee** table:

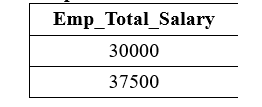
1. **SELECT** Emp\_Salary + 15000 **AS** Emp\_Updated\_Salary **FROM** Employee;



The following query performs the addition operation on the above **Employee** table with the WHERE clause:

1. **SELECT** Emp\_Salary + Emp\_Bonus **AS** Emp\_Total\_Salary **FROM** Employee **WHERE** Emp\_Salary > 25000;

It shows only records of those employees whose Emp\_Salary is greater than 25000:



**Syntax of SQL Subtraction Operator:**

1. **SELECT** Column\_Name\_1 Subtraction\_Operator Column\_Name2 **FROM** Table\_Name;

Subtraction Operator with WHERE Clause

The subtraction operator can also be used with the WHERE clause in the SELECT query.

**The syntax for using the WHERE clause with the subtraction operator is given below:**

1. **SELECT**  Column\_Name\_1 Subtraction\_Operator Column\_Name2 **FROM** Table\_Name **WHERE** Condition;

**Implementation of Subtraction operator in SQL:**

The following CREATE query creates the Employee table with five fields:

1. **CREATE** **TABLE** Employee
2. (
3. Employee\_ID **INT** AUTO\_INCREMENT **PRIMARY** **KEY**,
4. Emp\_Name **VARCHAR** (50),
5. Emp\_City **VARCHAR** (20),
6. Emp\_Salary **INT** NOT NULL,
7. Emp\_Panelty **INT** NOT NULL
8. ) ;

The following INSERT query inserts the record of employees into the Employee table:

1. **INSERT** **INTO** Employee (Employee\_ID, Emp\_Name, Emp\_City, Emp\_Salary, Emp\_Bonus) **VALUES** (101, Anuj, Ghaziabad, 25000, 500),
2. (102, Tushar, Lucknow, 29000, 1000),
3. (103, Vivek, Kolkata, 35000, 700),
4. (104, Shivam, Goa, 22000, 500);

The following SELECT query shows the data of **the Employee** table:

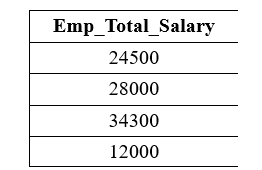
1. **SELECT** \* **FROM** Employee;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Employee\_Id** | **Emp\_Name** | **Emp\_City** | **Emp\_Salary** | **Emp\_Panelty** |
| 101 | Anuj | Ghaziabad | 25000 | 500 |
| 102 | Tushar | Lucknow | 29000 | 1000 |
| 103 | Vivek | Kolkata | 35000 | 700 |
| 104 | Shivam | Goa | 22000 | 500 |

The following query subtracts the values of the Emp\_Panelty column from the Emp\_Salary column of the **Employee** table using the subtraction operator:

1. **SELECT** Emp\_Salary - Emp\_Panelty **AS** Emp\_Total\_Salary **FROM** Employee;

**Output:**

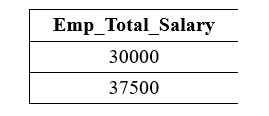


The following query performs the subtraction operation on the above **Employee** table with the WHERE clause:

1. **SELECT** Emp\_Panelty - Emp\_Salary **AS** Emp\_Total\_Salary **FROM** Employee **WHERE** Employee\_ID = 104;

It shows only records of those employees whose Employee\_ID is 103:

**Output:**



A [**nested query**](https://www.tutorialspoint.com/explain-about-nested-queries-in-dbms) is a query that has another query embedded within it. The embedded query is called a subquery.

A subquery typically appears within the [**WHERE clause**](https://www.tutorialspoint.com/sql/sql-where-clause.htm) of a query. It can sometimes appear in the FROM clause or [**HAVING clause**](https://www.tutorialspoint.com/sql/sql-having-clause.htm).

Let’s learn about nested queries with the help of an example.

Find the names of employee who have regno=103

The query is as follows −

select E.ename from employee E where E.eid IN (select S.eid from salary S where S.regno=103);

Student table

The student table is created as follows −

create table student(id number(10), name varchar2(20),classID number(10), marks varchar2(20));

Insert into student values(1,'pinky',3,2.4);

Insert into student values(2,'bob',3,1.44);

Insert into student values(3,'Jam',1,3.24);

Insert into student values(4,'lucky',2,2.67);

Insert into student values(5,'ram',2,4.56);

select \* from student;

[Explore our](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)**[latest online courses](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)**[and learn new skills at your own pace. Enroll and become a certified expert to boost your career.](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)

Output

You will get the following output −

| **Id** | **Name** | **classID** | **Marks** |
| --- | --- | --- | --- |
| 1 | Pinky | 3 | 2.4 |
| 2 | Bob | 3 | 1.44 |
| 3 | Jam | 1 | 3.24 |
| 4 | Lucky | 2 | 2.67 |
| 5 | Ram | 2 | 4.56 |

Teacher table

The teacher table is created as follows −

Example

Create table teacher(id number(10), name varchar(20), subject varchar2(10), classID number(10), salary number(30));

Insert into teacher values(1,’bhanu’,’computer’,3,5000);

Insert into teacher values(2,'rekha','science',1,5000);

Insert into teacher values(3,'siri','social',NULL,4500);

Insert into teacher values(4,'kittu','mathsr',2,5500);

select \* from teacher;

Output

You will get the following output –

| **Id** | **Name** | **Subject** | **classID** | **Salary** |
| --- | --- | --- | --- | --- |
| 1 | Bhanu | Computer | 3 | 5000 |
| 2 | Rekha | Science | 1 | 5000 |
| 3 | Siri | Social | NULL | 4500 |
| 4 | Kittu | Maths | 2 | 5500 |

Create table class(id number(10), grade number(10), teacherID number(10), noofstudents number(10));

insert into class values(1,8,2,20);

insert into class values(2,9,3,40);

insert into class values(3,10,1,38);

select \* from class;

Output

You will get the following output −

| **Id** | **Grade** | **teacherID** | **No.ofstudents** |
| --- | --- | --- | --- |
| 1 | 8 | 2 | 20 |
| 2 | 9 | 3 | 40 |
| 3 | 10 | 1 | 38 |

Now let’s work on nested queries

Example 1

Select AVG(noofstudents) from class where teacherID IN(

Select id from teacher

Where subject=’science’ OR subject=’maths’);

Output

You will get the following output −

20.0

Nested query is one of the most useful functionalities of SQL. Nested queries are useful when we want to write complex queries where one query uses the result from another query. Nested queries will have multiple SELECT statements nested together. A subquery is a SELECT statement nested within another SELECT statement.

**What is a Nested Query in SQL?**

A nested query in SQL contains a query inside another query. The outer query will use the result of the inner query. For instance, a nested query can have two **SELECT** statements, one on the inner query and the other on the outer query.

**What are the Types of Nested Queries in SQL?**

Nested queries in SQL can be classified into two different types:

* Independent Nested Queries
* Co-related Nested Queries

**Independent Nested Queries**

In independent nested queries, the execution order is from the innermost query to the outer query. An outer query won't be executed until its inner query completes its execution. The outer query uses the result of the inner query. Operators such as **IN**, **NOT IN**, **ALL**, and **ANY** are used to write independent nested queries.

* The **IN** operator checks if a column value in the outer query's result is **present** in the inner query's result. The final result will have rows that satisfy the **IN** condition.
* The **NOT IN** operator checks if a column value in the outer query's result is **not present** in the inner
* The **ALL** operator compares a value of the outer query's result with **all the values** of the inner query's result and returns the row if it matches all the values.
* The **ANY** operator compares a value of the outer query's result with all the inner query's result values and returns the row if there is a match with **any value**.

**Co-related Nested Queries**

In co-related nested queries, the inner query uses the values from the outer query to execute the inner query for every row processed by the outer query. The co-related nested queries run slowly because the inner query is executed for every row of the outer query's result.

## How to Write Nested Query in SQL?

We can write a nested query in SQL by nesting a **SELECT** statement within another **SELECT** statement. The outer **SELECT** statement uses the result of the inner **SELECT** statement for processing.

The general syntax of nested queries will be:

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE]

)

The **SELECT** query inside the brackets **()** is the inner query, and the **SELECT** query outside the brackets is the outer query. The outer query uses the result of the inner query.

## Examples of Nested Query in SQL

We will use the **Employees** and **Awards** table below to understand independent and co-related nested queries. We will be using **Oracle SQL** syntax in our queries.

Let's create the **Employees** and **Awards** tables:

**SQL JOIN:**

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN





Both these tables are connected by one common key (column) i.e **ROLL\_NO**.

We can perform a JOIN operation using the given SQL query:

**SELECT** s.roll\_no, s.name, s.address, s.phone, s.age, sc.course\_id  
**FROM** Student s  
**JOIN** StudentCourse sc **ON** s.roll\_no = sc.roll\_no;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ROLL\_NO | NAME | ADDRESS | PHONE | AGE | COURSE\_ID |
| 1 | HARSH | DELHI | XXXXXXXXXX | 18 | 1 |
| 2 | PRATIK | BIHAR | XXXXXXXXXX | 19 | 2 |
| 3 | RIYANKA | SILGURI | XXXXXXXXXX | 20 | 2 |
| 4 | DEEP | RAMNAGAR | XXXXXXXXXX | 18 | 3 |
| 5 | SAPTARHI | KOLKATA | XXXXXXXXXX | 19 | 1 |

**Types of JOIN in SQL**

There are many types of Joins in SQL. Depending on the use case, you can use different type of SQL JOIN clause.

Here are the frequently used SQL JOIN types:

**Table of Content**

* [INNER JOIN](https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/#sql-inner-join)
* [LEFT JOIN](https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/#sql-left-join)
* [RIGHT JOIN](https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/#sql-right-join)
* [FULL JOIN](https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/#sql-full-join)
* [Natural join](https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/#sql-natural-join-)

**SQL INNER JOIN**

The [**INNER JOIN**](https://www.geeksforgeeks.org/sql-inner-join) keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

**Syntax:**

The syntax for SQL INNER JOIN is:

**SELECT** table1.column1,table1.column2,table2.column1,....  
**FROM** table1   
**INNER JOIN** table2  
**ON** table1.matching\_column = table2.matching\_column;

Here,

* **table1**: First table.
* **table2**: Second table
* **matching\_column**: Column common to both the tables.



**INNER JOIN Example**

Let’s look at the example of INNER JOIN clause, and understand it’s working.

This query will show the names and age of students enrolled in different courses.

**SELECT** StudentCourse.COURSE\_ID, Student.NAME, Student.AGE **FROM** Student  
**INNER JOIN** StudentCourse  
**ON** Student.ROLL\_NO = StudentCourse.ROLL\_NO;



**SQL LEFT JOIN**

LEFT JOIN returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

**Syntax:**

The syntax of LEFT JOIN in SQL is**:**

**SELECT** table1.column1,table1.column2,table2.column1,....  
**FROM** table1   
**LEFT JOIN** table2  
**ON** table1.matching\_column = table2.matching\_column;



**SQL RIGHT JOIN**

[**RIGHT JOIN**](https://www.geeksforgeeks.org/sql-right-join) returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.It is very similar to LEFT JOIN For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

**Syntax:**

The syntax of RIGHT JOIN in SQL is:

**SELECT** table1.column1,table1.column2,table2.column1,....  
**FROM** table1   
**RIGHT JOIN** table2  
**ON** table1.matching\_column = table2.matching\_column;

## Lightbox

## ****SQL FULL JOIN****

[**FULL JOIN**](https://www.geeksforgeeks.org/sql-full-join) creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.

The syntax of SQL FULL JOIN is:

**SELECT** table1.column1,table1.column2,table2.column1,....  
**FROM** table1   
**FULL JOIN** table2  
**ON** table1.matching\_column = table2.matching\_column;

| **NAME** | **COURSE\_ID** |
| --- | --- |
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| DHANRAJ | NULL |
| ROHIT | NULL |
| NIRAJ | NULL |
| NULL | 4 |
| NULL | 5 |
| NULL | 4 |