**RDBMS**

**RDBMS** stands for Relational Database Management Systems..

All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS.

It is called Relational Data Base Management System (RDBMS) because it is based on relational model introduced by E.F. Codd.

**Database**

 database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a [database management system (DBMS)](https://www.oracle.com/in/database/what-is-database/#WhatIsDBMS). The data can then be easily accessed, managed, modified, updated, controlled, and organized.

**Database objects**

A **database object** is any defined object in a database that is used to store or reference data.Anything which we make from **create command**is known as Database Object.It can be used to hold and manipulate the data.Some of the examples of database objects are : view, sequence, indexes, etc.

**Keys**

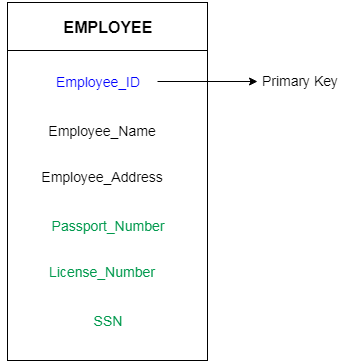
* Keys play an important role in the relational database.
* It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

**For example:** In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport\_number, license\_number, SSN are keys since they are unique for each person.

### DBMS Keys

### **1. Primary key**

* It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
* In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License\_Number and Passport\_Number as primary key since they are also unique.
* For each entity, selection of the primary key is based on requirement and developers.



### **A) PRIMARY KEY Constraint using create table**

In a create table statement, you can define primary key for a table.

**Syntax:**

CREATE TABLE <table\_name>

(

Column\_name1 datatype(),

Column\_name2 datatype(),

PRIMARY KEY (Column\_name1)

);

**For Example:** The following table definition has student\_id which is PRIMARY KEY.

CREATE TABLE Student\_details

(

     student\_id INT AUTO\_INCREMENT,

     student\_name varchar(50),

     student\_address varchar(200) PRIMARY KEY(student\_id)

);

### **B)  PRIMARY KEY Constraint Using Alter Table**

In MySQL, we can also add a primary key on the column of the existing table.

**Syntax:**

ALTER TABLE <table\_name>

ADD PRIMARY KEY (column\_name);

**For Example:** First of all we have to create a table with the specified fields.

**Create a table:**

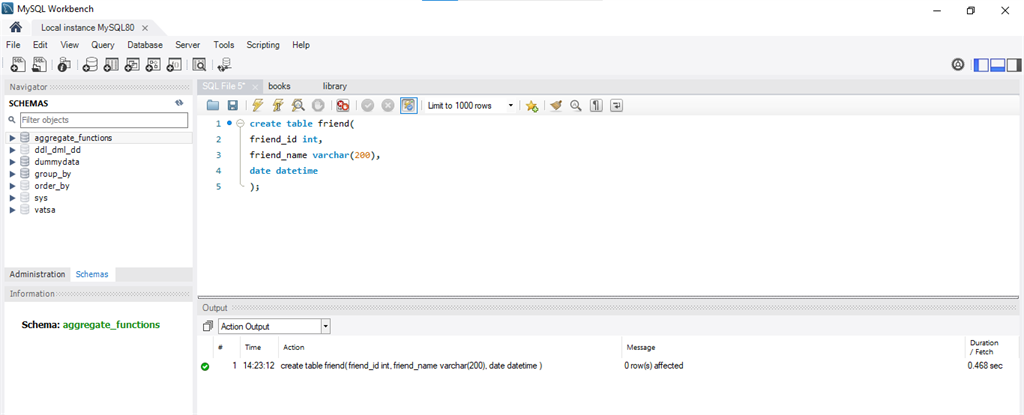
CREATE TABLE friend(

friend\_id int,

friend\_name varchar(200),

date DATETIME

);



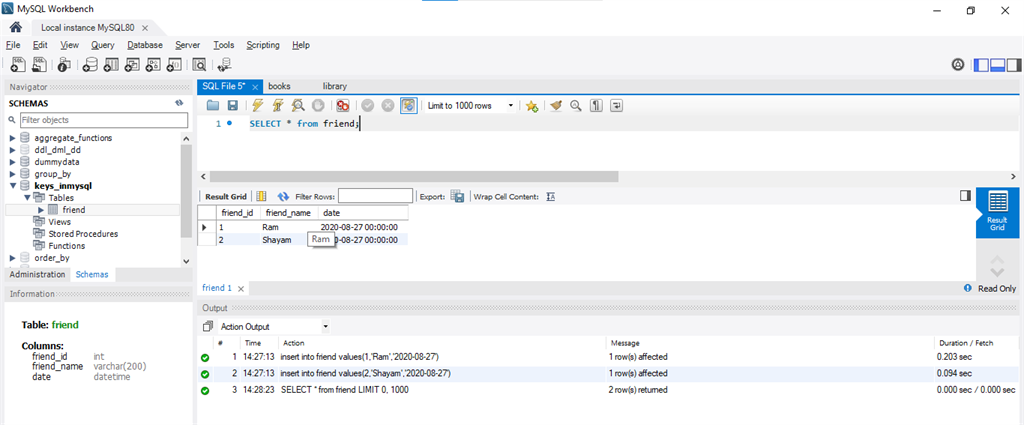
**Insert the values into the table:**

INSERT INTO friend VALUES(1,'Ram','2020-08-27');

INSERT INTO friend VALUES(2,'Shayam','2020-08-27');

**Result**:

SELECT \* from friend;

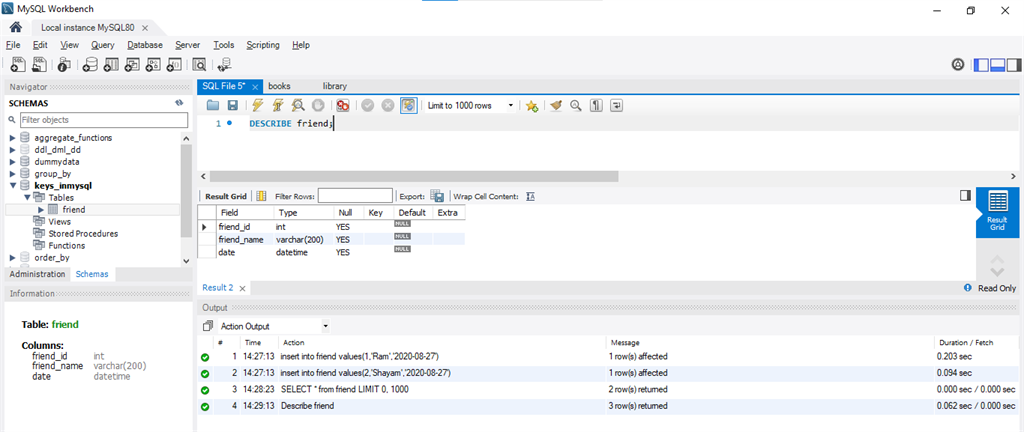


#### **Command to view data type of Table named employees**

The describe employees return the table attribute like fieldname,Type,Null,Key,Default,Extra etc.

**Syntax:** DESCRIBE friend;

**Output:**



#### **Command to add primary key into the Table name friend**

The Alter Query is used to redefine the table 'friend' and add keyword is used to add primary key on column 'friend\_id'.

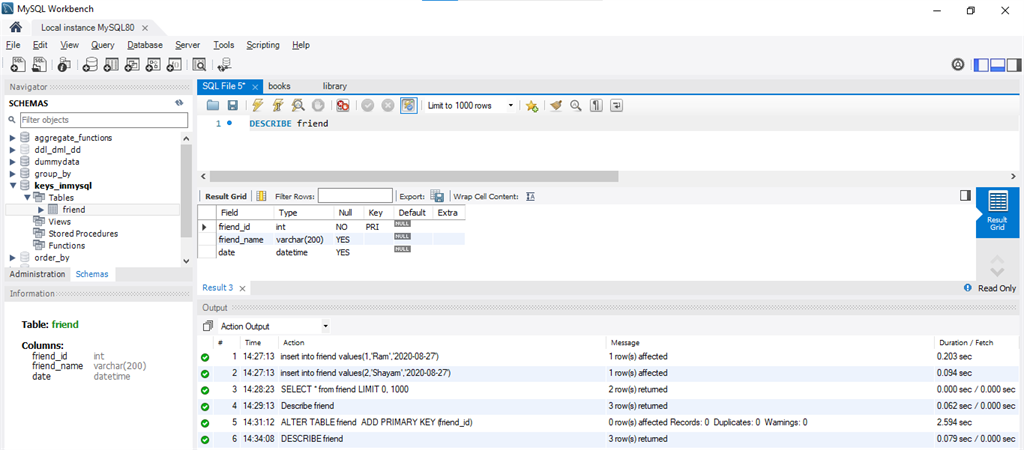
**Query:**

ALTER TABLE friend

ADD PRIMARY KEY (friend\_id);

Again type, DESCRIBE friend;

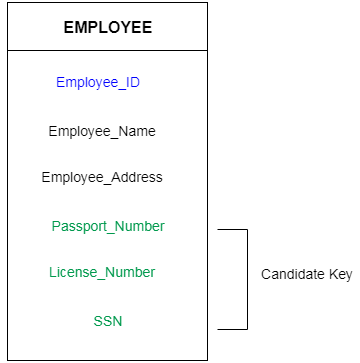
**OUTPUT:**



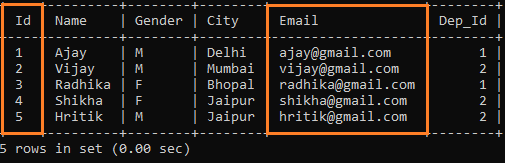
### **2. Candidate key**

* A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
* The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

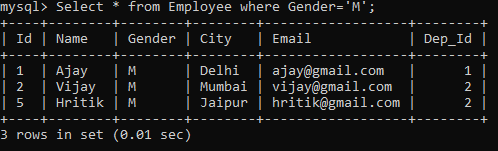
**For example:** In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport\_Number, and License\_Number, etc. are considered as a candidate key.



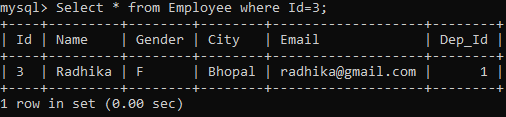
In the example that we took earlier, both *Id*and *Email*can act as a Candidate for the table as they contain unique and non-null values.



On the other hand, we cannot use the attributes like *City*or *Gender* to retrieve tuples from the table as they have no unique values.



Whereas on querying the table on the *Id* attribute will help us to retrieve unique tuples.



### **3.Super Key**

Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

**For example:** In the above EMPLOYEE table, for(EMPLOEE\_ID, EMPLOYEE\_NAME) the name of two employees can be the same, but their EMPLYEE\_ID can't be the same. Hence, this combination can also be a key.

In this example we will illustrate super keys with the help of the “student\_list” table.

**Code:**

CREATE TABLE student\_list(  
student\_name VARCHAR(50),  
class VARCHAR(50),  
Subject VARCHAR(50)  
);

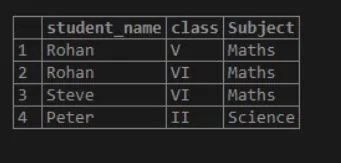
INSERT INTO student\_list(  
student\_name, class, subject)  
VALUES ('Rohan','V','Maths'),  
('Rohan','VI','Maths'),  
('Steve','VI','Maths'),  
('Peter','II','Science');

The data in student\_list table looks something as follows :

**Code:**

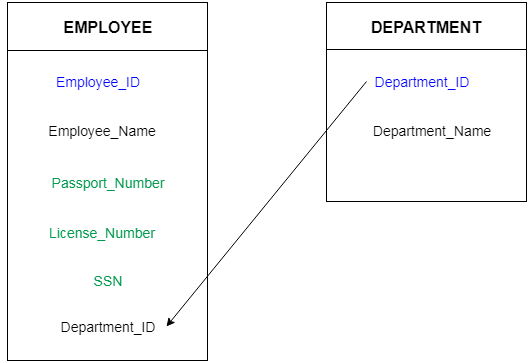
SELECT \* FROM student\_list;

**Output:**



### **4. Foreign key**

* Foreign keys are the column of the table which is used to point to the primary key of another table.
* In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
* We add the primary key of the DEPARTMENT table, Department\_Id as a new attribute in the EMPLOYEE table.
* Now in the EMPLOYEE table, Department\_Id is the foreign key, and both the tables are related.



**create a table**

CREATE TABLE species (

id int NOT NULL AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

PRIMARY KEY(id)) ;

**Insert some records into it.**

INSERT INTO species VALUES

(1, 'birds'),

(2, 'dog'),

(3, 'cat'),

(4, 'monkey');

**And, create another table whose name is zoo.**

CREATE table zoo (

id int(4) NOT NULL,

name VARCHAR NOT NULL,

FK\_species int NOT NULL,

INDEX (FK\_species),

FOREIGN KEY (FK\_species) REFERENCES species (id),

PRIMARY KEY(id)

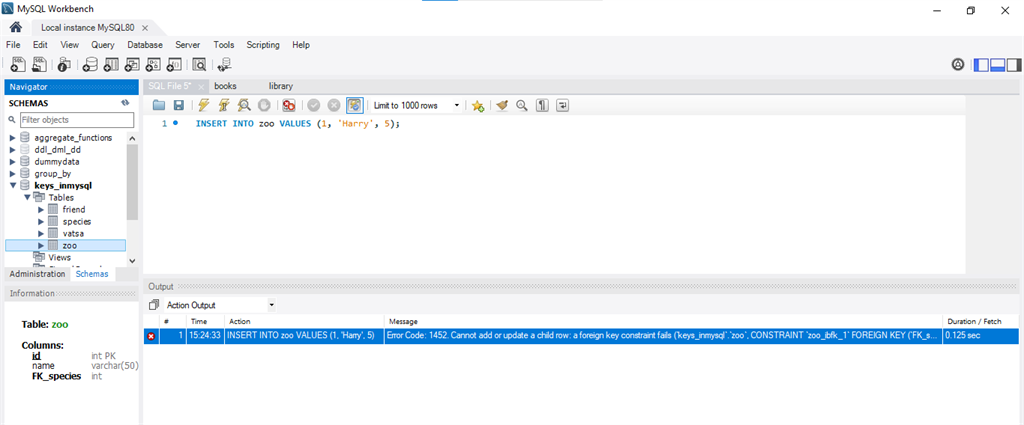
);

As in the above command,  relationship in a foreign key now exists between the fields zoo.species and species.id. An entry in the zoo table will be permitted only if the corresponding zoo.species field matches a value in the species id field. What happens when we have attempted to enter a record for Harry cat with an invalid species command:

**Query**

INSERT INTO zoo VALUES (1, 'Harry', 5);

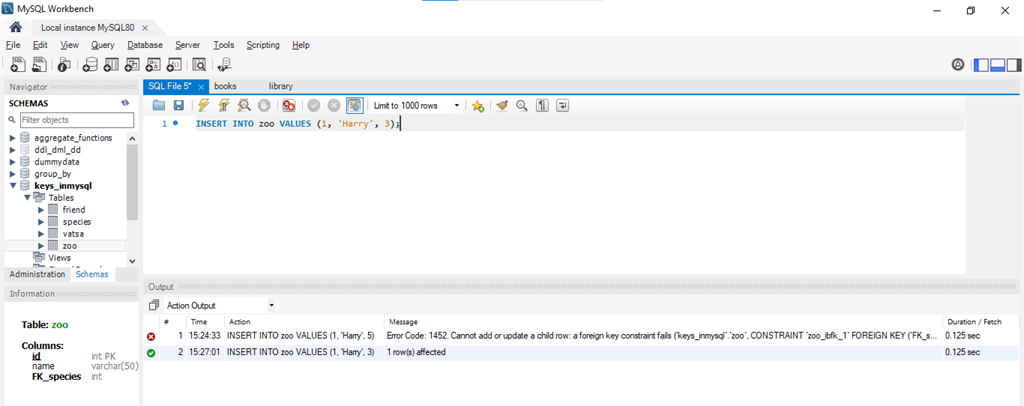
**OUTPUT**



MySQL defines the species table to find if the species query exists and, finding that it does not, rejects the record. Contrast this with what happens when you enter the same record with a valid species code (one that already exists in the species table):

**Query**

INSERT INTO zoo VALUES (1, 'Harry', 3);



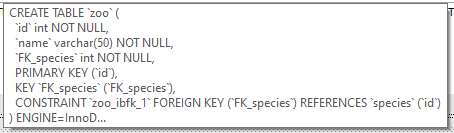
Here, MySQL findout the species table to see if the species query exists and, finding that it does, permits the record to be saved to the zoo table.

## **DELETE A FOREIGN KEY CONSTRAINT**

To delete a foreign key relationship, first use the SHOW CREATE TABLE command to find out InnoDB's internal label for the field.

**QUERY**

SHOW create TABLE zoo



And then use the command of the ALTER TABLE with the DROP FOREIGN KEY given as below:

**Query**

ALTER TABLE zoo

 DROP FOREIGN KEY zoo\_ibfk\_1;

## **ADD A FOREIGN KEY CONSTRAINT**

To add a foreign key to an existing table, use the ALTER TABLE command with an ADD FOREIGN KEY clause to define the appropriate field as a foreign key:

**Query**

ALTER TABLE zoo

ADD FOREIGN KEY

(FK\_species) REFERENCES species (id);

**DDL Commands**  
In this section, We will cover the following DDL commands as follows.

1. Create
2. Alter
3. truncate
4. drop

**Command-1:**  
**CREATE:**  
This command is used to create a new table in SQL. The user has to give information like table name, column names, and their datatypes.

**Syntax –**

CREATE TABLE table\_name

(

column\_1 datatype,

column\_2 datatype,

column\_3 datatype,

....

);

**Example –**  
We need to create a table for storing Student information of a particular College. Create syntax would be as below.

CREATE TABLE Student\_info

(

College\_Id number(2),

College\_name varchar(30),

Branch varchar(10)

);

**Command-2:**  
**ALTER:**  
This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can do add, delete or modify tasks easily.

**Syntax –**  
Syntax to add a column to an existing table.

ALTER TABLE table\_name

ADD column\_name datatype;

**Example –**  
In our Student\_info table, we want to add a new column for CGPA. The syntax would be as below as follows.

ALTER TABLE Student\_info

ADD CGPA number;

**Command-3:**  
**TRUNCATE:**  
This command is used to remove all rows from the table, but the structure of the table still exists.

**Syntax –**

Syntax to remove an existing table.

TRUNCATE TABLE table\_name;

**Example –**  
The College Authority wants to remove the details of all students for new batches but wants to keep the table structure. The command they can use is as follows.

TRUNCATE TABLE Student\_info;

**Command-4:**  
**DROP:**  
This command is used to remove an existing table along with its structure from the Database.

**Syntax –**  
Syntax to drop an existing table.

DROP TABLE table\_name;

**Example –**  
If the College Authority wants to change their Database by deleting the Student\_info Table.

DROP TABLE Student\_info;

**DQL Commands**

Data Query Language (DQL) is used to fetch the data from the database. It uses only one command:

### SELECT:

This command helps you to select the attribute based on the condition described by the WHERE clause.

**Syntax:**

SELECT expressions

FROM TABLES

WHERE conditions;

**For example:**

SELECT FirstName

FROM Student

WHERE RollNo > 15;

# SQL Operators

Every database administrator and user uses SQL queries for manipulating and accessing the data of database tables and views.

The manipulation and retrieving of the data are performed with the help of reserved words and characters, which are used to perform arithmetic operations, logical operations, comparison operations, compound operations, etc.

## **Types of Operator**

SQL operators are categorized in the following categories:

1. SQL Arithmetic Operators
2. SQL Comparison Operators
3. SQL Logical Operators
4. SQL Set Operators
5. SQL Bit-wise Operators
6. SQL Unary Operators

## **SQL Arithmetic Operators**

The **Arithmetic Operators** perform the mathematical operation on the numerical data of the SQL tables. These operators perform addition, subtraction, multiplication, and division operations on the numerical operands.

**Following are the various arithmetic operators performed on the SQL data:**

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

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

The **Addition Operator** in SQL performs the addition on the numerical data of the database table. In SQL, we can easily add the numerical values of two columns of the same table by specifying both the column names as the first and second operand. We can also add the numbers to the existing numbers of the specific column.

**Syntax of SQL Addition Operator:**

SELECT operand1 + operand2;

**Let's understand the below example which explains how to execute Addition Operator in SQL query:**

This example consists of an **Employee\_details** table, which has four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_Monthlybonus.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp Monthlybonus** |
| 101 | Tushar | 25000 | 4000 |
| 102 | Anuj | 30000 | 200 |

Suppose, we want to add **20,000** to the salary of each employee specified in the table. Then, we have to write the following query in the SQL:

SELECT Emp\_Salary + 20000 as Emp\_New\_Salary FROM Employee\_details;

In this query, we have performed the SQL addition operation on the single column of the given table.

Suppose, we want to add the Salary and monthly bonus columns of the above table, then we have to write the following query in SQL:

SELECT Emp\_Salary + Emp\_Monthlybonus as Emp\_Total\_Salary FROM Employee\_details;

In this query, we have added two columns with each other of the above table.

## **SQL Subtraction Operator (-)**

The Subtraction Operator in SQL performs the subtraction on the numerical data of the database table. In SQL, we can easily subtract the numerical values of two columns of the same table by specifying both the column names as the first and second operand. We can also subtract the number from the existing number of the specific table column.

**Syntax of SQL Subtraction Operator:**

SELECT operand1 - operand2;

**Let's understand the below example which explains how to execute Subtraction Operator in SQL query:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Penalty** |
| 201 | Abhay | 25000 | 200 |
| 202 | Sumit | 30000 | 500 |

This example consists of an **Employee\_details** table, which has four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_Monthlybonus.**

Suppose we want to subtract 5,000 from the salary of each employee given in the **Employee\_details** table. Then, we have to write the following query in the SQL:

SELECT Emp\_Salary - 5000 as Emp\_New\_Salary FROM Employee\_details;

In this query, we have performed the SQL subtraction operation on the single column of the given table.

If we want to subtract the penalty from the salary of each employee, then we have to write the following query in SQL:

SELECT Emp\_Salary - Penalty as Emp\_Total\_Salary FROM Employee\_details;

## **SQL Multiplication Operator (\*)**

The Multiplication Operator in SQL performs the Multiplication on the numerical data of the database table. In SQL, we can easily multiply the numerical values of two columns of the same table by specifying both the column names as the first and second operand.

**Syntax of SQL Multiplication Operator:**

1. SELECT operand1 \* operand2;

**Let's understand the below example which explains how to execute Multiplication Operator in SQL query:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Penalty** |
| 201 | Abhay | 25000 | 200 |
| 202 | Sumit | 30000 | 500 |

This example consists of an **Employee\_details** table, which has four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_Monthlybonus.**

Suppose, we want to double the salary of each employee given in the **Employee\_details** table. Then, we have to write the following query in the SQL:

SELECT Emp\_Salary \* 2 as Emp\_New\_Salary FROM Employee\_details;

In this query, we have performed the SQL multiplication operation on the single column of the given table.

If we want to multiply **the Emp\_Id** column to **Emp\_Salary** column of that employee whose **Emp\_Id** is **202,** then we have to write the following query in SQL:

SELECT Emp\_Id \* Emp\_Salary as Emp\_Id \* Emp\_Salary FROM Employee\_details WHERE Emp\_Id = 202;

In this query, we have multiplied the values of two columns by using the WHERE clause.

## **SQL Division Operator (/)**

The Division Operator in SQL divides the operand on the left side by the operand on the right side.

**Syntax of SQL Division Operator:**

SELECT operand1 / operand2;

In SQL, we can also divide the numerical values of one column by another column of the same table by specifying both column names as the first and second operand.

We can also perform the division operation on the stored numbers in the column of the SQL table.

**Let's understand the below example which explains how to execute Division Operator in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, and Emp\_Salary.**

|  |  |  |
| --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** |
| 201 | Abhay | 25000 |
| 202 | Sumit | 30000 |

Suppose, we want to half the salary of each employee given in the Employee\_details table. For this operation, we have to write the following query in the SQL:

SELECT Emp\_Salary / 2 as Emp\_New\_Salary FROM Employee\_details;

In this query, we have performed the SQL division operation on the single column of the given table.

## **SQL Modulus Operator (%)**

The Modulus Operator in SQL provides the remainder when the operand on the left side is divided by the operand on the right side.

**Syntax of SQL Modulus Operator:**

SELECT operand1 % operand2;

**Let's understand the below example which explains how to execute Modulus Operator in SQL query:**

This example consists of a **Division** table, which has three columns **Number, First\_operand, and Second\_operand.**

|  |  |  |
| --- | --- | --- |
| **Number** | **First operand** | **Second operand** |
| 1 | 56 | 4 |
| 2 | 32 | 8 |
| 3 | 89 | 9 |
| 4 | 18 | 10 |
| 5 | 10 | 5 |

If we want to get the remainder by dividing the numbers of First\_operand column by the numbers of Second\_operand column, then we have to write the following query in SQL:

SELECT First\_operand % Second\_operand as Remainder FROM Employee\_details;

## **SQL Comparison Operators**

The **Comparison Operators** in SQL compare two different data of SQL table and check whether they are the same, greater, and lesser. The SQL comparison operators are used with the WHERE clause in the SQL queries

**Following are the various comparison operators which are performed on the data stored in the SQL database tables:**

1. SQL Equal Operator (=)
2. SQL Not Equal Operator (!=)
3. SQL Greater Than Operator (>)
4. SQL Greater Than Equals to Operator (>=)
5. SQL Less Than Operator (<)\
6. SQL Less Than Equals to Operator (<=)

### **SQL Equal Operator (=)**

This operator is highly used in SQL queries. The **Equal Operator** in SQL shows only data that matches the specified value in the query.

This operator returns TRUE records from the database table if the value of both operands specified in the query is matched.

**Let's understand the below example which explains how to execute Equal Operator in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, and Emp\_Salary.**

|  |  |  |
| --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** |
| 201 | Abhay | 30000 |
| 202 | Ankit | 40000 |
| 203 | Bheem | 30000 |
| 204 | Ram | 29000 |
| 205 | Sumit | 30000 |

Suppose, we want to access all the records of those employees from the **Employee\_details** table whose salary is 30000. Then, we have to write the following query in the SQL database:

SELECT \* FROM Employee\_details WHERE Emp\_Salary = 30000;

In this example, we used the SQL equal operator with WHERE clause for getting the records of those employees whose salary is 30000.

### **SQL Equal Not Operator (!=)**

The **Equal Not Operator** in SQL shows only those data that do not match the query's specified value.

This operator returns those records or rows from the database views and tables if the value of both operands specified in the query is not matched with each other.

**Let's understand the below example which explains how to execute Equal Not Operator in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, and Emp\_Salary.**

|  |  |  |
| --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** |
| 201 | Abhay | 45000 |
| 202 | Ankit | 45000 |
| 203 | Bheem | 30000 |
| 204 | Ram | 29000 |
| 205 | Sumit | 29000 |

Suppose, we want to access all the records of those employees from the **Employee\_details** table whose salary is not 45000. Then, we have to write the following query in the SQL database:

SELECT \* FROM Employee\_details WHERE Emp\_Salary != 45000;

In this example, we used the SQL equal not operator with WHERE clause for getting the records of those employees whose salary is not 45000.

### **SQL Greater Than Operator (>)**

The **Greater Than Operator** in SQL shows only those data which are greater than the value of the right-hand operand.

**Let's understand the below example which explains how to execute Greater ThanOperator (>) in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, and Emp\_Salary.**

|  |  |  |
| --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** |
| 201 | Abhay | 45000 |
| 202 | Ankit | 45000 |
| 203 | Bheem | 30000 |
| 204 | Ram | 29000 |
| 205 | Sumit | 29000 |

Suppose, we want to access all the records of those employees from the **Employee\_details** table whose employee id is greater than 202. Then, we have to write the following query in the SQL database:

SELECT \* FROM Employee\_details WHERE Emp\_Id **>** 202;

Here, SQL greater than operator displays the records of those employees from the above table whose Employee Id is greater than 202.

### **SQL Greater Than Equals to Operator (>=)**

The **Greater Than Equals to Operator** in SQL shows those data from the table which are greater than and equal to the value of the right-hand operand.

**Let's understand the below example which explains how to execute greater than equals to the operator (>=) in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, and Emp\_Salary.**

|  |  |  |
| --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** |
| 201 | Abhay | 45000 |
| 202 | Ankit | 45000 |
| 203 | Bheem | 30000 |
| 204 | Ram | 29000 |
| 205 | Sumit | 29000 |

Suppose, we want to access all the records of those employees from the **Employee\_details** table whose employee id is greater than and equals to 202. For this, we have to write the following query in the SQL database:

SELECT \* FROM Employee\_details WHERE Emp\_Id **>**= 202;

Here,**'SQL greater than equals to operator'** with WHERE clause displays the rows of those employees from the table whose Employee Id is greater than and equals to 202.

### **SQL Less Than Operator (<)**

The **Less Than Operator** in SQL shows only those data from the database tables which are less than the value of the right-side operand.

This comparison operator checks that the left side operand is lesser than the right side operand. If the condition becomes true, then this operator in SQL displays the data which is less than the value of the right-side operand.

**Let's understand the below example which explains how to execute less than operator (<) in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, and Emp\_Salary.**

|  |  |  |
| --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** |
| 201 | Abhay | 45000 |
| 202 | Ankit | 45000 |
| 203 | Bheem | 30000 |
| 204 | Ram | 29000 |
| 205 | Sumit | 29000 |

Suppose, we want to access all the records of those employees from the **Employee\_details** table whose employee id is less than 204. For this, we have to write the following query in the SQL database:

SELECT \* FROM Employee\_details WHERE Emp\_Id **<** **204**;

Here,**SQL less than operator** with WHERE clause displays the records of those employees from the above table whose Employee Id is less than 204.

### **SQL Less Than Equals to Operator (<=)**

The **Less Than Equals to Operator** in SQL shows those data from the table which are lesser and equal to the value of the right-side operand.

This comparison operator checks that the left side operand is lesser and equal to the right side operand.

**Let's understand the below example which explains how to execute less than equals to the operator (<=) in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, and Emp\_Salary.**

|  |  |  |
| --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** |
| 201 | Abhay | 45000 |
| 202 | Ankit | 45000 |
| 203 | Bheem | 30000 |
| 204 | Ram | 29000 |
| 205 | Sumit | 29000 |

Suppose, we want to access all the records of those employees from the **Employee\_details** table whose employee id is less and equals **203.** For this, we have to write the following query in the SQL database:

SELECT \* FROM Employee\_details WHERE Emp\_Id **<**= 203;

Here, SQL **less than equals to the operator** with WHERE clause displays the rows of those employees from the table whose Employee Id is less than and equals 202.

## **SQL Logical Operators**

The **Logical Operators** in SQL perform the Boolean operations, which give two results **True and False.** These operators provide **True** value if both operands match the logical condition.

**Following are the various logical operators which are performed on the data stored in the SQL database tables:**

1. SQL ALL operator
2. SQL AND operator
3. SQL OR operator
4. SQL BETWEEN operator
5. SQL IN operator
6. SQL NOT operator
7. SQL ANY operator
8. SQL LIKE operator
9. SQL EXISTS operator

### **SQL ALL Operator**

The ALL operator in SQL compares the specified value to all the values of a column from the sub-query in the SQL database.

This operator is always used with the following statement:

1. SELECT,
2. HAVING, and
3. WHERE.

**Syntax of ALL operator:**

SELECT column\_Name1, ...., column\_NameN FROM table\_Name WHERE column Comparison\_operator ALL (SELECT column FROM tablename2)

**Let's understand the below example which explains how to execute ALL logical operators in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Abhay | 25000 | Gurgaon |
| 202 | Ankit | 45000 | Delhi |
| 203 | Bheem | 30000 | Jaipur |
| 204 | Ram | 29000 | Mumbai |
| 205 | Sumit | 40000 | Kolkata |

If we want to access the employee id and employee names of those employees from the table whose salaries are greater than the salary of employees who lives in Jaipur city, then we have to type the following query in SQL.

SELECT Emp\_Id, Emp\_Name FROM Employee\_details WHERE Emp\_Salary **>** ALL (SELECT Emp\_Salary FROM Employee\_details WHERE Emp\_City = Jaipur)

Here, we used the **SQL ALL operator** with greater than the operator.

### **SQL AND Operator**

The **AND operator** in SQL would show the record from the database table if all the conditions separated by the AND operator evaluated to True. It is also known as the conjunctive operator and is used with the WHERE clause.

**Syntax of AND operator:**

1. SELECT column1, ...., columnN FROM table\_Name WHERE condition1 AND condition2 AND condition3 AND ....... AND conditionN;

**Let's understand the below example which explains how to execute AND logical operator in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Abhay | 25000 | Delhi |
| 202 | Ankit | 45000 | Chandigarh |
| 203 | Bheem | 30000 | Delhi |
| 204 | Ram | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

* Suppose, we want to access all the records of those employees from the **Employee\_details** table whose salary is 25000 and the city is Delhi. For this, we have to write the following query in SQL:
  1. SELECT \* FROM Employee\_details WHERE Emp\_Salary = 25000 OR Emp\_City = 'Delhi';

Here,**SQL AND operator** with WHERE clause shows the record of employees whose salary is 25000 and the city is Delhi.

### **SQL OR Operator**

The OR operator in SQL shows the record from the table if any of the conditions separated by the OR operator evaluates to True. It is also known as the conjunctive operator and is used with the WHERE clause.

**Syntax of OR operator:**

SELECT column1, ...., columnN FROM table\_Name WHERE condition1 OR condition2 OR condition3 OR ....... OR conditionN;

**Let's understand the below example which explains how to execute OR logical operator in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Abhay | 25000 | Delhi |
| 202 | Ankit | 45000 | Chandigarh |
| 203 | Bheem | 30000 | Delhi |
| 204 | Ram | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

If we want to access all the records of those employees from the **Employee\_details** table whose salary is 25000 or the city is Delhi. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE Emp\_Salary = 25000 OR Emp\_City = 'Delhi';

Here, **SQL OR operator** with WHERE clause shows the record of employees whose salary is 25000 or the city is Delhi.

### **SQL BETWEEN Operator**

The **BETWEEN operator** in SQL shows the record within the range mentioned in the SQL query. This operator operates on the numbers, characters, and date/time operands.

If there is no value in the given range, then this operator shows NULL value.

**Syntax of BETWEEN operator:**

SELECT column\_Name1, column\_Name2 ...., column\_NameN FROM table\_Name WHERE column\_nameBETWEEN value1 and value2;

**Let's understand the below example which explains how to execute BETWEEN logical operator in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Abhay | 25000 | Delhi |
| 202 | Ankit | 45000 | Chandigarh |
| 203 | Bheem | 30000 | Delhi |
| 204 | Ram | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

Suppose, we want to access all the information of those employees from the **Employee\_details** table who is having salaries between 20000 and 40000. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE Emp\_Salary BETWEEN 30000 AND 45000;

Here, we used the **SQL BETWEEN operator** with the Emp\_Salary field.

### **SQL IN Operator**

The **IN operator** in SQL allows database users to specify two or more values in a WHERE clause. This logical operator minimizes the requirement of multiple OR conditions.

This operator makes the query easier to learn and understand. This operator returns those rows whose values match with any value of the given list.

**Syntax of IN operator:**

SELECT column\_Name1, column\_Name2 ...., column\_NameN FROM table\_Name WHERE column\_name IN (list\_of\_values);

**Let's understand the below example which explains how to execute IN logical operator in SQL query:**

This example consists of an **Employee\_details** table, which has three columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Abhay | 25000 | Delhi |
| 202 | Ankit | 45000 | Chandigarh |
| 203 | Bheem | 30000 | Delhi |
| 204 | Ram | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

Suppose, we want to show all the information of those employees from the **Employee\_details** table whose **Employee Id** is 202, 204, and 205. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE Emp\_Id IN (202, 204, 205);

Here, we used the **SQL IN operator** with the Emp\_Id column.

Suppose, we want to show all the information of those employees from the **Employee\_details** table whose **Employee Id** is not equal to 202 and 205. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE Emp\_Id NOT IN (202,205);

Here, we used the **SQL NOT IN operator** with the Emp\_Id column.

### **SQL NOT Operator**

The **NOT operator** in SQL shows the record from the table if the condition evaluates to false. It is always used with the WHERE clause.

**Syntax of NOT operator:**

SELECT column1, column2 ...., columnN FROM table\_Name WHERE NOT condition;

**Let's understand the below example which explains how to execute NOT logical operator in SQL query:**

This example consists of an **Employee\_details** table, which has four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Abhay | 25000 | Delhi |
| 202 | Ankit | 45000 | Chandigarh |
| 203 | Bheem | 30000 | Delhi |
| 204 | Ram | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

Suppose, we want to show all the information of those employees from the **Employee\_details** table whose Cityis not Delhi. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE NOT Emp\_City = 'Delhi' ;

In this example, we used the **SQL NOT operator** with the Emp\_City column.

Suppose, we want to show all the information of those employees from the **Employee\_details** table whose Cityis not Delhi and Chandigarh. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE NOT Emp\_City = 'Delhi' AND NOT Emp\_City = 'Chandigarh';

In this example, we used the **SQL NOT operator** with the Emp\_City column.

### **SQL ANY Operator**

The **ANY operator** in SQL shows the records when any of the values returned by the sub-query meet the condition.

The ANY logical operator must match at least one record in the inner query and must be preceded by any SQL comparison operator.

**Syntax of ANY operator:**

SELECT column1, column2 ...., columnN FROM table\_Name WHERE column\_name comparison\_operator ANY ( SELECT column\_name FROM table\_name WHERE condition(s)) ;

### **SQL LIKE Operator**

The **LIKE operator** in SQL shows those records from the table which match with the given pattern specified in the sub-query.

The percentage (%) sign is a wildcard which is used in conjunction with this logical operator.

This operator is used in the WHERE clause with the following three statements:

1. SELECT statement
2. UPDATE statement
3. DELETE statement

**Syntax of LIKE operator:**

SELECT column\_Name1, column\_Name2 ...., column\_NameN FROM table\_Name WHERE column\_name LIKE pattern;

**Let's understand the below example which explains how to execute LIKE logical operator in SQL query:**

This example consists of an **Employee\_details** table, which has four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 45000 | Chandigarh |
| 203 | Saket | 30000 | Delhi |
| 204 | Abhay | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

If we want to show all the information of those employees from the **Employee\_details** whose name starts with ''s''. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE Emp\_Name LIKE 's%' ;

In this example, we used the SQL LIKE operator with **Emp\_Name** column because we want to access the record of those employees whose name starts with s.

If we want to show all the information of those employees from the **Employee\_details**whose name ends with ''y''. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE Emp\_Name LIKE '%y' ;

If we want to show all the information of those employees from the **Employee\_details**whose name starts with ''S'' and ends with ''y''. For this, we have to write the following query in SQL:

SELECT \* FROM Employee\_details WHERE Emp\_Name LIKE 'S%y' ;

**SQL Set Operators**

The **Set Operators** in SQL combine a similar type of data from two or more SQL database tables. It mixes the result, which is extracted from two or more SQL queries, into a single result.

Set operators combine more than one select statement in a single query and return a specific result set.

**Following are the various set operators which are performed on the similar data stored in the two SQL database tables:**

1. SQL Union Operator
2. SQL Union ALL Operator
3. SQL Intersect Operator
4. SQL Minus Operator

### **SQL Union Operator**

The SQL Union Operator combines the result of two or more SELECT statements and provides the single output.

The data type and the number of columns must be the same for each SELECT statement used with the UNION operator. This operator does not show the duplicate records in the output table.

**Syntax of UNION Set operator:**

1. SELECT column1, column2 ...., columnN FROM table\_Name1 [WHERE conditions]
2. UNION
3. SELECT column1, column2 ...., columnN FROM table\_Name2 [WHERE conditions];

**Let's understand the below example which explains how to execute Union operator in Structured Query Language:**

In this example, **we used two tables.** Both tables have four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 45000 | Delhi |
| 203 | Saket | 30000 | Aligarh |

**Table: Employee\_details1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 203 | Saket | 30000 | Aligarh |
| 204 | Saurabh | 40000 | Delhi |
| 205 | Ram | 30000 | Kerala |
| 201 | Sanjay | 25000 | Delhi |

**Table: Employee\_details2**

Suppose, we want to see the employee name and employee id of each employee from both tables in a single output. For this, we have to write the following query in SQL:

1. SELECT Emp\_ID, Emp\_Name FROM Employee\_details1
2. UNION
3. SELECT Emp\_ID, Emp\_Name FROM Employee\_details2 ;

### **SQL Union ALL Operator**

The SQL Union Operator is the same as the UNION operator, but the only difference is that it also shows the same record.

**Syntax of UNION ALL Set operator:**

1. SELECT column1, column2 ...., columnN FROM table\_Name1 [WHERE conditions]
2. UNION ALL
3. SELECT column1, column2 ...., columnN FROM table\_Name2 [WHERE conditions];

**Let's understand the below example which explains how to execute Union ALL operator in Structured Query Language:**

In this example, **we used two tables.** Both tables have four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 45000 | Delhi |
| 203 | Saket | 30000 | Aligarh |

**Table: Employee\_details1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 203 | Saket | 30000 | Aligarh |
| 204 | Saurabh | 40000 | Delhi |
| 205 | Ram | 30000 | Kerala |
| 201 | Sanjay | 25000 | Delhi |

**Table: Employee\_details2**

If we want to see the employee name of each employee of both tables in a single output. For this, we have to write the following query in SQL:

1. SELECT Emp\_Name FROM Employee\_details1
2. UNION ALL
3. SELECT Emp\_Name FROM Employee\_details2 ;

### **SQL Intersect Operator**

The SQL Intersect Operator shows the common record from two or more SELECT statements. The data type and the number of columns must be the same for each SELECT statement used with the INTERSECT operator.

**Syntax of INTERSECT Set operator:**

1. SELECT column1, column2 ...., columnN FROM table\_Name1 [WHERE conditions]
2. INTERSECT
3. SELECT column1, column2 ...., columnN FROM table\_Name2 [WHERE conditions];

**Let's understand the below example which explains how to execute INTERSECT operator in Structured Query Language:**

In this example, **we used two tables.** Both tables have four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 45000 | Delhi |
| 203 | Saket | 30000 | Aligarh |

**Table: Employee\_details1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 203 | Saket | 30000 | Aligarh |
| 204 | Saurabh | 40000 | Delhi |
| 205 | Ram | 30000 | Kerala |
| 201 | Sanjay | 25000 | Delhi |

**Table: Employee\_details2**

Suppose, we want to see a common record of the employee from both the tables in a single output. For this, we have to write the following query in SQL:

1. SELECT Emp\_Name FROM Employee\_details1
2. INTERSECT
3. SELECT Emp\_Name FROM Employee\_details2 ;

### **SQL Minus Operator**

The SQL Minus Operator combines the result of two or more SELECT statements and shows only the results from the first data set.

**Syntax of MINUS operator:**

1. SELECT column1, column2 ...., columnN FROM First\_tablename [WHERE conditions]
2. MINUS
3. SELECT column1, column2 ...., columnN FROM Second\_tablename [WHERE conditions];

**Let's understand the below example which explains how to execute INTERSECT operator in Structured Query Language:**

In this example, **we used two tables.** Both tables have four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

**Table: Employee\_details1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 45000 | Delhi |
| 203 | Saket | 30000 | Aligarh |

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 203 | Saket | 30000 | Aligarh |
| 204 | Saurabh | 40000 | Delhi |
| 205 | Ram | 30000 | Kerala |
| 201 | Sanjay | 25000 | Delhi |

**Table: Employee\_details2**

Suppose, we want to see the name of employees from the first result set after the combination of both tables. For this, we have to write the following query in SQL:

1. SELECT Emp\_Name FROM Employee\_details1
2. MINUS
3. SELECT Emp\_Name FROM Employee\_details2 ;

## **SQL Unary Operators**

The **Unary Operators** in SQL perform the unary operations on the single data of the SQL table, i.e., these operators operate only on one operand.

These types of operators can be easily operated on the numeric data value of the SQL table.

**Following are the various unary operators which are performed on the numeric data stored in the SQL table:**

1. SQL Unary Positive Operator
2. SQL Unary Negative Operator
3. SQL Unary Bitwise NOT Operator

### **SQL Unary Positive Operator**

The SQL Positive (+) operator makes the numeric value of the SQL table positive.

**Syntax of Unary Positive Operator**

SELECT +(column1), +(column2) ...., +(columnN) FROM table\_Name [WHERE conditions] ;

**Let's understand the below example which explains how to execute a Positive unary operator on the data of SQL table:**

This example consists of an**Employee\_details** table, which has four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 45000 | Chandigarh |
| 203 | Saket | 30000 | Delhi |
| 204 | Abhay | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

Suppose, we want to see the salary of each employee as positive from the Employee\_details table. For this, we have to write the following query in SQL:

SELECT +Emp\_Salary Employee\_details ;

### **SQL Unary Negative Operator**

The SQL Negative (-) operator makes the numeric value of the SQL table negative.

**Syntax of Unary Negative Operator**

SELECT -(column\_Name1), -(column\_Name2) ...., -(column\_NameN) FROM table\_Name [WHERE conditions];

**Let's understand the below example which explains how to execute Negative unary operator on the data of SQL table:**

This example consists of an **Employee\_details** table, which has four columns **Emp\_Id, Emp\_Name, Emp\_Salary, and Emp\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 45000 | Chandigarh |
| 203 | Saket | 30000 | Delhi |
| 204 | Abhay | 25000 | Delhi |
| 205 | Sumit | 40000 | Kolkata |

Suppose, we want to see the salary of each employee as negative from the Employee\_details table. For this, we have to write the following query in SQL:

SELECT -Emp\_Salary Employee\_details ;

Suppose, we want to see the salary of those employees as negative whose city is Kolkatain the Employee\_details table. For this, we have to write the following query in SQL:

SELECT -Emp\_Salary Employee\_details WHERE Emp\_City = 'Kolkata';

### **SQL Bitwise NOT Operator**

The SQL Bitwise NOT operator provides the one's complement of the single numeric operand. This operator turns each bit of numeric value. If the bit of any numerical value is 001100, then this operator turns these bits into 110011.

**Syntax of Bitwise NOT Operator**

SELECT ~(column1), ~(column2) ...., ~(columnN) FROM table\_Name [WHERE conditions];

**Let's understand the below example which explains how to execute the Bitwise NOT operator on the data of SQL table:**

This example consists of a**Student\_details** table, which has four columns **Roll\_No, Stu\_Name, Stu\_Marks, and Stu\_City.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp Id** | **Stu Name** | **Stu Marks** | **Stu City** |
| 101 | Sanjay | 85 | Delhi |
| 102 | Ajay | 97 | Chandigarh |
| 103 | Saket | 45 | Delhi |
| 104 | Abhay | 68 | Delhi |
| 105 | Sumit | 60 | Kolkata |

If we want to perform the Bitwise Not operator on the marks column of **Student\_details,** we have to write the following query in SQL:

SELECT ~Stu\_Marks Employee\_details ;

## **SQL Bitwise Operators**

The **Bitwise Operators** in SQL perform the bit operations on the Integer values. To understand the performance of Bitwise operators, you just knew the basics of Boolean algebra.

**Following are the two important logical operators which are performed on the data stored in the SQL database tables:**

1. Bitwise AND (&)
2. Bitwise OR(|)

### **Bitwise AND (&)**

The Bitwise AND operator performs the logical AND operation on the given Integer values. This operator checks each bit of a value with the corresponding bit of another value.

**Syntax of Bitwise AND Operator**

SELECT column1 & column2 & .... & columnN FROM table\_Name [WHERE conditions];

**Let's understand the below example which explains how to execute Bitwise AND operator on the data of SQL table:**

This example consists of the following table, which has two columns. Each column holds numerical values.

When we use the Bitwise AND operator in SQL, then SQL converts the values of both columns in binary format, and the AND operation is performed on the converted bits.

After that, SQL converts the resultant bits into user understandable format, i.e., decimal format.

|  |  |
| --- | --- |
| **Column1** | **Column2** |
| 1 | 1 |
| 2 | 5 |
| 3 | 4 |
| 4 | 2 |
| 5 | 3 |

Suppose, we want to perform the Bitwise AND operator between both the columns of the above table. For this, we have to write the following query in SQL:

SELECT Column1 & Column2 From TABLE\_AND ;

### **Bitwise OR (|)**

The Bitwise OR operator performs the logical OR operation on the given Integer values. This operator checks each bit of a value with the corresponding bit of another value.

**Syntax of Bitwise OR Operator**

SELECT column1 | column2 | .... | columnN FROM table\_Name [WHERE conditions] ;

**Let's understand the below example which explains how to execute Bitwise OR operator on the data of SQL table:**

This example consists of a table that has two columns. Each column holds numerical values.

When we used the Bitwise OR operator in SQL, then SQL converts the values of both columns in binary format, and the OR operation is performed on the binary bits. After that, SQL converts the resultant binary bits into user understandable format, i.e., decimal format.

|  |  |
| --- | --- |
| **Column1** | **Column2** |
| 1 | 1 |
| 2 | 5 |
| 3 | 4 |
| 4 | 2 |
| 5 | 3 |

Suppose, we want to perform the Bitwise OR operator between both the columns of the above table. For this, we have to write the following query in SQL:

SELECT Column1 | Column2 From TABLE\_OR;