# What Is SQL?

SQL stands for *Structured Query Language* – a declarative language for storing, manipulating and retrieving data stored in a relational database. It’s the most popular query language over databases.

# Data Types

Each column in a database table is required to have a **name** and a **data type**. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

**Note**: Different database define different names for data types. And even if the name is the same, the size and other details may be different as well. Always check the documentation!

List of data types of MySQL, SQL Server and MS Access SQL:

<https://www.w3schools.com/sql/sql_datatypes.asp>

# NULL vs NOT NULL

# Database Statements

## CREATE DATABASE

The SQL CREATE DATABASE statement is used to create a new SQL database.

**Syntax**

CREATE DATABASE Database-Name;

Notes:

* The database name must be **unique** within the RDBMS.
* Make sure you have the **admin privilege** before creating any database.

**Example**

If you want to create a new database named TestDB, use:

SQL> CREATE DATABASE TestDB;

Once a database is created, you can check it in the list of databases as follows:

SQL> SHOW DATABASES;

+---------------+

| Database      |

+---------------+

| Armood        |

| TutorialPoint |

| Orig          |

| TestDB        |

+---------------+

## DROP DATABASE

The SQL DROP DATABASE statement is used to delete an existing database in SQL schema.

**Syntax**

DROP DATABASE Database-Name;

Note: Make sure you have the admin privilege before creating any database.

**Example**

If you want to delete an existing database named TestDB, use:

DROP DATABASE TestDB;

Once a database is dropped, you can check it in the list of databases as follows:

SQL> SHOW DATABASES;

+---------------+

| Database      |

+---------------+

| Armood        |

| TutorialPoint |

| Orig          |

+---------------+

## USE

When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

The SQL USE statement is used to select any existing database in the SQL schema.

**Syntax**

USE Database-Name;

**Example**

You can check the available databases as shown below:

SQL> SHOW DATABASES;

+---------------+

| Database      |

+---------------+

| Armood        |

| TutorialPoint |

| Orig          |

| TestDB        |

+---------------+

Now, if you want to work with the TestDB database, use:

SQL> USE TestDB;

# Table Statements

## CREATE TABLE

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL CREATE TABLE statement is used to create a new table.

**Syntax**

CREATE TABLE Table-Name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY(one or more columns)

);

**Example**

The following example creates a Customers table with an ID as a primary key. The constraint NOT NULL show that these fields cannot be NULL:

SQL> CREATE TABLE Customers(

ID INT NOT NULL,

Name VARCHAR (20) NOT NULL,

Age INT NOT NULL,

Address CHAR (25),

Salary DECIMAL (18, 2),

PRIMARY KEY (ID)

);

You can verify if your table has been created successfully by looking at the message displayed by the SQL server. Or you can use the DESC command as follows:

SQL> DESC Customers;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int(11) | NO | PRI | | |

| Name | varchar(20) | NO | | | |

| Age | int(11) | NO | | | |

| Address | char(25) | YES | | NULL | |

| Salary | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

## DROP TABLE

The SQL DROP TABLE statement is used to remove a table definition and all of its data (values, indexes, triggers, constraints and permission specifications).

**Syntax**

DROP TABLE table-name;

**Example**

Let us first verify the Customers table and then we will delete it from the database as shown below:

SQL> DESC Customers;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int(11) | NO | PRI | | |

| Name | varchar(20) | NO | | | |

| Age | int(11) | NO | | | |

| Address | char(25) | YES | | NULL | |

| Salary | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

This means that the Customers table is available in the database.

Now drop it:

SQL> DROP TABLE Customers;

Now, if you would try the DESC command, then you will get the following error:

SQL> DESC Customers;

ERROR 1146 (42S02): Table 'TEST.Customers' doesn't exist

Here, TEST is the database name which we are using for our examples.

## SELECT

### Without Condition

The SQL SELECT statement is used to fetch the data from a database table which returns this data in the form of a result table.

**Syntax**

SELECT column1, column2, ... columnN FROM table-name;

Tip: If you want to fetch all fields available in the table, just need to use a \* symbol instead of all column names:

SELECT \* FROM table-name;

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to fetch the ID, Name and Salary fields in Customers table, use:

SQL> SELECT ID, Name, Salary FROM Customers;

This would produce the following result:

+----+----------+----------+

| ID | Name | Salary |

+----+----------+----------+

| 1 | Ramesh | 2000.00 |

| 2 | Khilan | 1500.00 |

| 3 | Kaushik | 2000.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

If you want to fetch all the fields of the Customers table, use:

SQL> SELECT \* FROM Customers;

This would produce the result as shown below:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

### With Condition

To specify a condition while fetching the data from a single table, you can use the SQL WHERE clause. It is very useful to filter the records and fetch only necessary ones.

**Syntax**

SELECT column1, column2, columnN

FROM table-name

WHERE [condition]

You can specify a condition using the comparison or logical operators like >, <, =, LIKE, NOT, etc. The following examples would make this concept clear.

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

To fetch the ID, Name and Salary fields from the Customers table where the salary is greater than 2000, use:

SQL> SELECT ID, Name, Salary

FROM Customers

WHERE Salary > 2000;

This would produce the following result:

+----+----------+----------+

| ID | Name | Salary |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

### SELECT INTO

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

The new table will be created with the column names and types as defined in the old table. But you can create new column names using the AS clause.

**Syntax**

SELECT column1, column2, column3, ...

INTO new-table [IN external-db]

FROM old-table

WHERE condition;

**Examples**

The following statement creates a backup copy of Customers:

SQL> SELECT \* INTO CustomersBk

FROM Customers

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

SQL> SELECT \* INTO CustomersBk IN 'Backup.mdb'

FROM Customers;

The following statement copies only CustomerName and ContactName columns into a new table:

SQL> SELECT CustomerName, ContactName INTO CustomersBk

FROM Customers;

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

SQL> SELECT \* INTO CustomersGermany

FROM Customers

WHERE Country = 'Germany';

### SELECT DISTINCT

The SQL DISTINCT keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

**Syntax**

SELECT DISTINCT column1, column2, ... columnN

FROM table-name

WHERE [condition]

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

First, let us see how the following SELECT query returns the duplicate salary records.

SQL> SELECT Salary FROM Customers;

This would produce the following result, where the salary (2000) is coming twice:

+----------+

| Salary |

+----------+

| 1500.00 |

| 2000.00 |

| 2000.00 |

| 4500.00 |

| 6500.00 |

| 8500.00 |

| 10000.00 |

+----------+

Now, let us use the DISTINCT with SELECT and see the result:

SQL> SELECT DISTINCT Salary FROM Customers;

This would produce the following result where we do not have any duplicate entry:

+----------+

| Salary |

+----------+

| 1500.00 |

| 2000.00 |

| 4500.00 |

| 6500.00 |

| 8500.00 |

| 10000.00 |

+----------+

## INSERT INTO

The SQL INSERT INTO statement is used to add new rows of data to a table in the database.

**Syntax**

INSERT INTO Table-Name (column1, column2, column3, ... columnN)

VALUES (value1, value2, value3, ... valueN);

Note: Make sure column names and values are in the same order as the columns in the table.

**Example**

Following statements create 3 records in the Customers table:

SQL>

INSERT INTO Customers (ID, Name, Age, Address, Salary)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO Customers (ID, Name, Age, Address, Salary)

VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO Customers (ID, Name, Age, Address, Salary)

VALUES (3, 'Kaushik', 23, 'Kota', 2000.00 );

All the above statements would produce the following records in the Customers table as shown below:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

+----+----------+-----+-----------+----------+

### Populate One Table from Another Table

You can populate the data into a table over another table; provided the other table has a set of fields, which are required to populate the first table.

Here is the syntax:

INSERT INTO first-table-name [(column1, column2, ... columnN)]

SELECT column1, column2, ...columnN

FROM second-table-name

[WHERE condition];

### INSERT INTO SELECT

<https://www.w3schools.com/sql/sql_insert_into_select.asp>

## UPDATE

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

**Syntax**

UPDATE table-name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The following query will update the Address for a customer whose ID number is 6 in the table.

SQL> UPDATE Customers

SET Address = 'Pune'

WHERE ID = 6;

Now, the Customers table would have the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | Pune | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to modify all the Address and the Salary column values in the Customers table, you do not need to use the WHERE clause:

SQL> UPDATE Customers

SET Address = 'Pune', Salary = 1000.00;

Now, Customers table would have the following records:

+----+----------+-----+---------+---------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+---------+---------+

| 1 | Ramesh | 32 | Pune | 1000.00 |

| 2 | Khilan | 25 | Pune | 1000.00 |

| 3 | Kaushik | 23 | Pune | 1000.00 |

| 4 | Chaitali | 25 | Pune | 1000.00 |

| 5 | Hardik | 27 | Pune | 1000.00 |

| 6 | Komal | 22 | Pune | 1000.00 |

| 7 | Muffy | 24 | Pune | 1000.00 |

+----+----------+-----+---------+---------+

## DELETE

The SQL DELETE statement is used to delete the existing records from a table.

**Syntax**

DELETE FROM table-name

WHERE [condition];

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The following code has a query, which will DELETE a customer, whose ID is 6.

SQL> DELETE FROM Customers

WHERE ID = 6;

Now, the Customers table would have the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to DELETE all the records from the Customers table, you do not need to use the WHERE clause:

SQL> DELETE FROM Customers;

Now, the Customers table would not have any record.

## JOIN

The SQL JOIN clause is used to combine records from two or more tables in a database by using values common to each.

**Different Types of JOIN:**

* INNER JOIN − returns rows when there is a match in both tables.
* LEFT JOIN − returns all rows from the left table, even if there are no matches in the right table.
* RIGHT JOIN − returns all rows from the right table, even if there are no matches in the left table.
* FULL JOIN − returns rows when there is a match in one of the tables.
* SELF JOIN − join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
* CARTESIAN JOIN − returns the Cartesian product of the sets of records from the two or more joined tables.

### INNER JOIN



**Syntax**

SELECT column1.column1, column2.column2, ...

FROM table1

INNER JOIN table2

ON table1.common\_column = table2.common\_column;

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

And the Orders table having the following records:

+-----+---------------------+-------------+--------+-----------+

| OID | Date | CustomerID | Amount | ShipperID |

+-----+---------------------+-------------+--------+-----------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 | S1 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 | S2 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 | S3 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 | S6 |

+-----+---------------------+-------------+--------+-----------+

And the Shippers table having the following records:

+-----+-------------------+------------------+

| ID | ShipperName | Phone |

+-----+-------------------+------------------+

| S1 | Speedy Express | (503) 555-9831 |

| S2 | United Package | (503) 555-3199 |

| S3 | Federal Shipping | (503) 555-9931 |

| S4 | Plane Service | (503) 555-2060 |

+-----+-------------------+------------------+

The following statement selects all common orders with customer name from 2 tables – Customers and Orders:

SQL> SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

INNER JOIN Orders ON Customers.ID = Orders.CustomerID;

Output:

+----+----------+--------+---------------------+

| ID | Name | Amount | Date |

+----+----------+--------+---------------------+

| 3 | Kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | Kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+----+----------+--------+---------------------+

The following SQL statement selects all orders with customer name and shipper name from 3 tables –Customers, Orders, and Shippers:

SQL> SELECT Customers.ID, Customers.Name, Shippers.Name

FROM ((Customers

INNER JOIN Orders ON Customers.ID = Orders.CustomerID)

INNER JOIN Shippers ON Orders.ShipperID = Shippers.ID);

Output:

+----+----------+--------------------+

| ID | Name | ShipperName |

+----+----------+--------------------+

| 3 | Kaushik | Speedy Express |

| 3 | Kaushik | United Package |

| 2 | Khilan | Federal Shipping |

+----+----------+--------------------+

### LEFT JOIN



**Syntax**

SELECT column1.column1, column2.column2, ...

FROM table1

LEFT JOIN table2

ON table1.common\_column = table2.common\_column;

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

And the Orders table having the following records:

+-----+---------------------+-------------+--------+-----------+

| OID | Date | CustomerID | Amount | ShipperID |

+-----+---------------------+-------------+--------+-----------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 | S1 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 | S2 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 | S3 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 | S6 |

+-----+---------------------+-------------+--------+-----------+

The following statement selects all orders of the Customers table with customer name from 2 tables – Customers and Orders:

SQL> SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

LEFT JOIN Orders ON Customers.ID = Orders.CustomerID;

Output:

+----+----------+--------+---------------------+

| ID | Name | Amount | Date |

+----+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | Kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | Kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

+----+----------+--------+---------------------+

### RIGHT JOIN



**Syntax**

SELECT column1.column1, column2.column2, ...

FROM table1

RIGHT JOIN table2

ON table1.common\_column = table2.common\_column;

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

And the Orders table having the following records:

+-----+---------------------+-------------+--------+-----------+

| OID | Date | CustomerID | Amount | ShipperID |

+-----+---------------------+-------------+--------+-----------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 | S1 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 | S2 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 | S3 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 | S6 |

+-----+---------------------+-------------+--------+-----------+

The following statement selects all orders of the Orders table with customer name from 2 tables – Customers and Orders:

SQL> SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

RIGHT JOIN Orders ON Customers.ID = Orders.CustomerID;

Output:

+------+----------+--------+---------------------+

| ID | Name | Amount | Date |

+------+----------+--------+---------------------+

| 3 | Kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | Kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

### FULL JOIN (OUTER JOIN)



**Syntax**

SELECT column1.column1, column2.column2, ...

FROM table1

FULL JOIN table2

ON table1.common\_column = table2.common\_column;

**Example**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

And the Orders table having the following records:

+-----+---------------------+-------------+--------+-----------+

| OID | Date | CustomerID | Amount | ShipperID |

+-----+---------------------+-------------+--------+-----------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 | S1 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 | S2 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 | S3 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 | S6 |

+-----+---------------------+-------------+--------+-----------+

The following statement selects all orders of both tables with customer name from 2 tables – Customers and Orders:

SQL> SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

FULL JOIN Orders ON Customers.ID = Orders.CustomerID;

Output:

+------+----------+--------+---------------------+

| ID | Name | Amount | Date |

+------+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | Kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | Kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

| 3 | Kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | Kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

**You might not know**:

If your Database does not support FULL JOIN (For example, MySQL), then you can use [UNION ALL](#_UNION) clause to combine LEFT JOIN and RIGHT JOIN to create FULL JOIN.

### SELF JOIN

<https://www.tutorialspoint.com/sql/sql-self-joins.htm>

### CARTESIAN JOIN

<https://www.tutorialspoint.com/sql/sql-cartesian-joins.htm>

# Operators

## Arithmetic Operators

Same as C: +, -, \*, / and %

## Comparison Operators

Same as C: =, !=, >, >=, <, <=

Others:

* <>: Same as !=
* !<: Not less than (equal or greater)
* !>: Not greater than (equal or less)

## Logical Operators

### AND

The AND operator is used to combine multiple conditions to narrow data in an SQL statement.

**Example**:

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

To fetch the ID, Name and Salary fields from the Customers table for a customer with name Hardik and salary greater than 2000, use:

SQL> SELECT ID, Name, Salary

FROM Customers

WHERE Name = 'Hardik' AND Salary > 2000;

This would produce the following result:

+----+----------+----------+

| ID | Name | Salary |

+----+----------+----------+

| 5 | Hardik | 8500.00 |

+----+----------+----------+

### OR

The OR operator is used to combine multiple conditions to expand data in an SQL statement.

**Example**:

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

To fetch the ID, Name and Salary fields from the Customers table where the salary is greater than 2000 or the age is less than 25 years, use:

SQL> SELECT ID, Name, Salary

FROM Customers

WHERE Salary > 2000 OR age < 25;

This would produce the following result:

+----+----------+----------+

| ID | Name | Salary |

+----+----------+----------+

| 3 | Kaushik | 2000.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

### NOT

The NOT operator **reverses the meaning of the logical operator** with which it is used.

E.g.: NOT EXISTS, NOT BETWEEN, NOT IN, etc.

### IN

The IN operator is a **shorthand for multiple OR conditions**.

**Example**:

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

To fetch the all fields which age is 32, 25 or 22 from all columns, use:

SQL> SELECT \*

FROM Customers

WHERE Age IN (32, 25, 22); -- Same as: WHERE Age = 32 OR Age = 25 OR Age = 22

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

### BETWEEN

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

**Example**:

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

To fetch the all fields which age is from 22 to 25 from all columns, use:

SQL> SELECT \*

FROM Customers

WHERE Age BETWEEN 22 AND 25; -- Same as: WHERE Age >= 22 AND Age <= 25

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

### ANY

The ANY operator returns TRUE if **any of the subquery values meet the condition**. It's used with a WHERE or HAVING clause.

**Example:**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

And the Orders table having the following records:

+-----+---------------------+-------------+--------+-----------+

| OID | Date | CustomerID | Amount | ShipperID |

+-----+---------------------+-------------+--------+-----------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 | S1 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 | S2 |

| 101 | 2009-11-20 00:00:00 | 8 | 1560 | S3 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 | S6 |

+-----+---------------------+-------------+--------+-----------+

The following statement returns TRUE and lists ID and Name of customers which meet the condition:

SQL> SELECT ID, Name

FROM Customers

WHERE ID = ANY (SELECT CustomerID FROM Orders WHERE Amount > 1500);

This would produce the following result:

+----+----------+

| ID | Name |

+----+----------+

| 3 | Kaushik |

### ALL

The ALL operator returns TRUE if **all of the subquery values meet the condition**. It's used with a WHERE or HAVING clause.

**Example:**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

And the Orders table having the following records:

+-----+---------------------+-------------+--------+-----------+

| OID | Date | CustomerID | Amount | ShipperID |

+-----+---------------------+-------------+--------+-----------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 | S1 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 | S2 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 | S3 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 | S6 |

+-----+---------------------+-------------+--------+-----------+

The following statement returns TRUE and lists ID and Name of customers which meet the condition:

SQL> SELECT ID, Name

FROM Customers

WHERE ID = ALL (SELECT CustomerID FROM Orders WHERE Amount > 1500);

This would produce the following result:

+----+----------+

| ID | Name |

+----+----------+

| 2 | Khilan |

| 3 | Kaushik |

### LIKE

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

There are some wildcards (used to substitute one or more characters in a string) often used in conjunction with the LIKE operator:

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| \_ | Represents a single character | h\_t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ^ | Represents any character not in the brackets | h[^oa]t finds hit, but not hot and hat |
| - | Represents a range of characters | c[a-b]t finds cat and cbt |

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

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE Name LIKE 'a%' | Finds any values that start with "a" |
| WHERE Name LIKE '%a' | Finds any values that end with "a" |
| WHERE Name LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE Name LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE Name LIKE 'a\_%' | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE Name LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE Name LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

### EXISTS

The EXISTS operator is used to test for the existence of any record in a subquery. It returns true if the subquery returns one or more records.

**Example**:

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

And the Orders table having the following records:

+-----+---------------------+-------------+--------+-----------+

| OID | Date | CustomerID | Amount | ShipperID |

+-----+---------------------+-------------+--------+-----------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 | S1 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 | S2 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 | S3 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 | S6 |

+-----+---------------------+-------------+--------+-----------+

The following statement returns TRUE and lists all ID and Name of customers:

SQL> SELECT ID, Name

FROM Customers

WHERE EXISTS (SELECT OID FROM Orders WHERE ShipperID = 'S1');

This would produce the following result:

+----+----------+

| ID | Name |

+----+----------+

| 1 | Ramesh |

| 2 | Khilan |

| 3 | Kaushik |

| 4 | Chaitali |

| 5 | Hardik |

| 6 | Komal |

| 7 | Muffy |

# UNION

<https://www.w3schools.com/sql/sql_union.asp>

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without **returning any duplicate rows**.

To use this UNION clause, each SELECT statement must have:

* The same number of columns selected
* The same number of column expressions
* The same data type and
* Have them in the same order. But they need not have to be in the same length.

**Syntax:**

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]

UNION [ALL]

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]

**Example:**

Consider the Customers table having the following records:

+----+----------+-----+-----------+----------+

| ID | Name | Age | Address | Salary |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | Kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

And the Orders table having the following records:

+-----+---------------------+-------------+--------+

|OID | Date | CustomerID | Amount |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now join these two tables (excluding duplicate rows) in our SELECT statement as follows:

SQL> SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

LEFT JOIN Orders ON Customers.ID = Orders.CustomerID;

UNION

SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

RIGHT JOIN Orders ON Customers.ID = Orders.CustomerID;

Result:

+------+----------+--------+---------------------+

| ID | Name | Amount | Date |

+------+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | Kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | Kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

+------+----------+--------+---------------------+

Now join these two tables (including duplicate rows) in our SELECT statement as follows:

SQL> SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

LEFT JOIN Orders ON Customers.ID = Orders.CustomerID;

UNION ALL

SELECT Customers.ID, Customers.Name, Orders.Amount, Orders.Date

FROM Customers

RIGHT JOIN Orders ON Customers.ID = Orders.CustomerID;

Result:

+------+----------+--------+---------------------+

| ID | Name | Amount | Date |

+------+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

# Constraints

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL:

* NOT NULL − Ensures that a column cannot have NULL value.
* DEFAULT − Provides a default value for a column when none is specified.
* UNIQUE − Ensures that all values in a column are different.
* PRIMARY − Uniquely identifies each row/record in a database table.
* FOREIGN − Uniquely identifies a row/record in any of the given database table.
* CHECK − Ensures that all the values in a column satisfies certain conditions.
* INDEX − Creates and retrieves data from the database very quickly.

## Create Constraints

Constraints can be specified when a table is created with the CREATE TABLE statement (as shown [here](#_gjdgxs)) or you can use the ALTER TABLE statement to create constraints even after the table is created.

## Dropping Constraints

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the Employees table, use:

ALTER TABLE Employees DROP CONSTRAINT Employees\_PK;

## Integrity Constraints

Integrity constraints are used to ensure accuracy and consistency of the data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in Referential Integrity (RI). These constraints include *Primary Key*, *Foreign Key*, *Unique Constraints* and other constraints which are mentioned above.

# Sorting and Grouping

## ORDER BY

<https://www.tutorialspoint.com/sql/sql-order-by.htm>

<https://www.tutorialspoint.com/sql/sql-sorting-results.htm>

## GROUP BY

<https://www.tutorialspoint.com/sql/sql-group-by.htm>

# Aliases

<https://www.w3schools.com/sql/sql_alias.asp>

# Wildcards

<https://www.w3schools.com/sql/sql_wildcards.asp>

# TOP, LIMIT or ROWNUM

<https://www.tutorialspoint.com/sql/sql-top-clause.htm>

# Functions

## MIN() and MAX()

<https://www.w3schools.com/sql/sql_min_max.asp>

## COUNT(), AVG() and SUM()

<https://www.w3schools.com/sql/sql_count_avg_sum.asp>