

MySQL uses many different data types broken into three categories −

* Numeric
* Date and Time
* String Types.

Let us now discuss them in detail.

Numeric Data Types

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you.

The following list shows the common numeric data types and their descriptions −

* **INT** − A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
* **TINYINT** − A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
* **SMALLINT** − A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
* **MEDIUMINT** − A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
* **BIGINT** − A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
* **FLOAT(M,D)** − A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
* **DOUBLE(M,D)** − A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
* **DECIMAL(M,D)** − An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

Date and Time Types

The MySQL date and time datatypes are as follows −

* **DATE** − A date in YYYY-MM-DD format, between “1000-01-01” and 9999-12-31. For example, December 30th, 1973 would be stored as “1973-12-30”.
* **DATETIME** − A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
* **TIMESTAMP** − A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 ( YYYYMMDDHHMMSS ).
* **TIME** − Stores the time in a HH:MM:SS format.
* **YEAR(M)** − Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

String Types

Although the numeric and date types are fun, most data you'll store will be in a string format. This list describes the common string datatypes in MySQL.

* **CHAR(M)** − A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
* **VARCHAR(M)** − A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.
* **BLOB or TEXT** − A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are **case sensitive** on BLOBs and are **not case sensitive** in TEXT fields. You do not specify a length with BLOB or TEXT.
* **TINYBLOB or TINYTEXT** − A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
* **MEDIUMBLOB or MEDIUMTEXT** − A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
* **LONGBLOB or LONGTEXT** − A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.
* **ENUM** − An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

How to create database

====================

create database javabatch1;

How to connect to database

use javabatch1;

how to create the table

Create table employee(empId int, name varchar(25), gender char, empsalary float, designation varchar(20));

mysql> show tables;

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

| Tables\_in\_javabatch1 |

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

| employee |

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

1 row in set (0.13 sec)

mysql> select \* from employee;

Empty set (0.01 sec)

Insert into employee values (101, “Naresh”, ‘m’, 5000.00, “SoftwareEngineer”);

Insert into employee (empId, gender, salary, designation, empName) values (101, ‘m’,1000.00,”SSE”,”Naresh”);

insert into employee (empId, empName, empSalary,empDesignation,empGender,empAddress,status,empPhoneNumber, empAadhar,empDOB,empJoiningDate,empPan,empLocation) values(107,"Akhil",1000.00,"SE","M","Vijayawada",1,"123456789”,"123456","1992-02-02","2018-05-05","987456","Pune");

mysql> insert into employee values(101, "naresh", "M", 50000.00, "Software Engin

ner");

select name from employee;

CRUD Operations:

C -> create

Example: create table tableName(fieldname fieldType);

create table employee(empId int, name varchar(25), gender char, empsalary float, designation varchar(20));

insert into employee values(101, “naresh”, ‘M’, 10000.00, “Software Engineer”);

R -> retrieve

Example: select \* from employee;

U -> update

Update employee set empSalary=100000.00, designation=”Senior Software Engineer” where empId=101;

D -> delete

delete from employee where empId=101;

Primary Key

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (ID)  
);

create table employee(empId int not null, empName varchar(30) not null, empSalary float(10,2)not null, empDesignation varchar(50) not null, primary key(empId))

Not Null

Alter Table

Distinct

Count

Where clause

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_equal_to) |
| > | Greater than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than) |
| < | Less than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than) |
| >= | Greater than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than2) |
| <= | Less than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than2) |
| <> | Not equal. **Note:** In some versions of SQL this operator may be written as != | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not_equal_to) |
| BETWEEN | Between a certain range (SELECT \* FROM employee  WHERE empId BETWEEN 50 AND 60); | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| LIKE | Search for a pattern (SELECT \* FROM employee  WHERE name LIKE 's%'); | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| IN | To specify multiple possible values for a column (\*SELECT \* FROM Customers  WHERE City IN ('Paris','London')); |  |

## The SQL AND, OR and NOT Operators

### AND Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 AND condition2 AND condition3 ...;

SELECT \* FROM employee  
WHERE Country='Germany' AND City='Berlin';

### OR Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 OR condition2 OR condition3 ...;

SELECT \* FROM employee  
WHERE City='Berlin' OR City='München';

### NOT Syntax

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

SELECT \* FROM employee  
WHERE NOT Country='Germany';

## Combining AND, OR and NOT

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

### Example

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

select \* from employee where empLocation="Hyderabad" and (empAddress="Guntur" or empAdress="Karimnagar");

The following SQL statement selects all fields from "Customers" where country is NOT "Germany" and NOT "USA":

### Example

SELECT \* FROM employee  
WHERE NOT Country='Germany' AND NOT Country='USA';

True and (True or false)

# SQL ORDER BY Keyword

## The SQL ORDER BY Keyword

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

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

### ORDER BY Syntax

SELECT column1, column2, ...  
FROM table\_name  
ORDER BY column1, column2, ... ASC|DESC;

## ORDER BY Example

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

### Example

SELECT \* FROM employee  
ORDER BY empName;

SELECT \* FROM Customers  
ORDER BY Country DESC;

## ORDER BY Several Columns Example

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

### Example

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

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

## The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

### GROUP BY Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)ORDER BY column\_name(s);

SELECT \* FROM employee group by empName;

## The SQL INSERT INTO Statement

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

### INSERT INTO Syntax

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

The first way specifies both the column names and the values to be inserted:

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

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

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

## SQL TOP, LIMIT and ROWNUM Examples

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

### Example

SELECT TOP 3 \* FROM Customers;

The following SQL statement shows the equivalent example using the LIMIT clause (for MySQL):

### Example

SELECT \* FROM Customers  
LIMIT 3;

The following SQL statement shows the equivalent example using ROWNUM (for Oracle):

### Example

SELECT \* FROM Customers  
WHERE ROWNUM <= 3;

## SQL TOP PERCENT Example

The following SQL statement selects the first 50% of the records from the "Customers" table (for SQL Server/MS Access):

### Example

SELECT TOP 50 PERCENT \* FROM Customers;

## ADD a WHERE CLAUSE

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

### Example

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

The following SQL statement shows the equivalent example using the LIMIT clause (for MySQL):

### Example

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

The following SQL statement shows the equivalent example using ROWNUM (for Oracle):

### Example

SELECT \* FROM Customers  
WHERE Country='Germany' AND ROWNUM <= 3;

## The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

### MIN() Syntax

SELECT MIN(column\_name)  
FROM table\_name  
WHERE condition;

### Example

SELECT MIN(empId) AS employeeId  
FROM employee;

### MAX() Syntax

SELECT MAX(column\_name)  
FROM table\_name  
WHERE condition;

### Example

SELECT MAX(empId) AS employeeId  
FROM employee;

## The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

### COUNT() Syntax

SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

### AVG() Syntax

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

### SUM() Syntax

SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;

## COUNT() Example

The following SQL statement finds the number of products:

### Example

SELECT COUNT(empID)  
FROM employee;

## AVG() Example

The following SQL statement finds the average price of all products:

### Example

SELECT AVG(empSalary)  
FROM employee;

## SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

### Example

SELECT SUM(empSalary)  
FROM employee;

## SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of the query.

### Alias Column Syntax

SELECT column\_name AS alias\_name  
FROM table\_name;

## SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

**SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

## AUTO INCREMENT Field

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

## Syntax for MySQL

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE Persons (  
    Personid int NOT NULL AUTO\_INCREMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (Personid)  
);

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

## SQL FOREIGN KEY on CREATE TABLE

## 

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

**MySQL:**

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

## SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

### INNER JOIN Syntax

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;



## SQL LEFT JOIN Keyword

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

### LEFT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2ON table1.column\_name = table2.column\_name;

**Note:** In some databases LEFT JOIN is called LEFT OUTER JOIN.



## SQL RIGHT JOIN Keyword

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

### RIGHT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

**Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



## SQL FULL OUTER JOIN Keyword

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

**Note:** FULL OUTER JOIN can potentially return very large result-sets!

**Tip:** FULL OUTER JOIN and FULL JOIN are the same.

### FULL OUTER JOIN Syntax

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition;



## SQL Self JOIN

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

### Self JOIN Syntax

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

## SQL Self JOIN Example

The following SQL statement matches customers that are from the same city:

### Example

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