

Assignment:-4

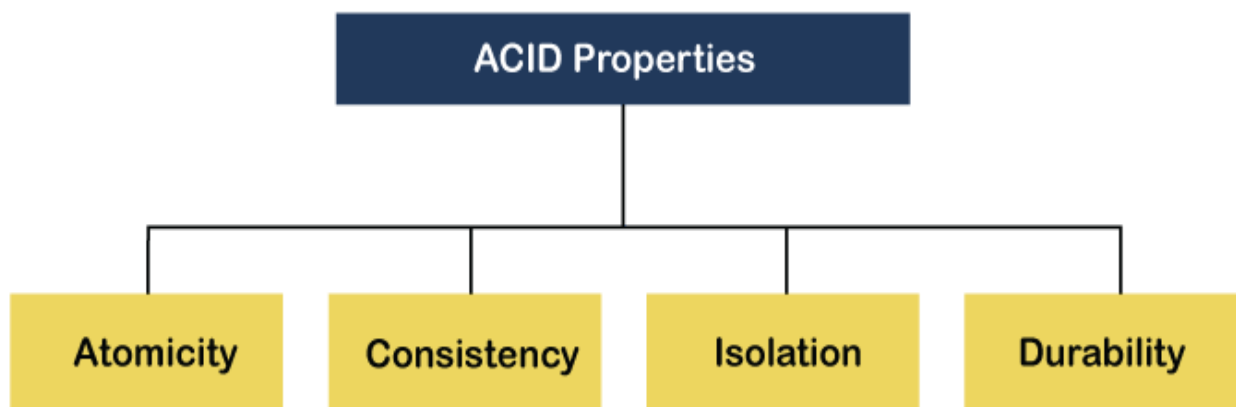
1)ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

ACID Properties

The expansion of the term ACID defines for:



1) Atomicity

The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

2) Consistency

The word **consistency** means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

3) Isolation

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

4) Durability

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

2)List of storage engines

MySQL supported storage engines:

- InnoDB
- MyISAM
- Memory
- CSV
- Merge
- Archive
- Federated
- Blackhole
- Example

InnoDB is the most widely used storage engine with transaction support. It is an ACID compliant storage engine. It supports row-level locking, crash recovery and multi-version concurrency control. It is the only engine which provides foreign key referential integrity constraint. Oracle recommends using InnoDB for tables except for specialized use cases.

Vrushank_PHP

MyISAM is the original storage engine. It is a fast storage engine. It does not support transactions. MyISAM provides table-level locking. It is used mostly in Web and data warehousing.

Memory storage engine creates tables in memory. It is the fastest engine. It provides table-level locking. It does not support transactions. Memory storage engine is ideal for creating temporary tables or quick lookups. The data is lost when the database is restarted.

CSV stores data in CSV files. It provides great flexibility because data in this format is easily integrated into other applications.

Merge operates on underlying MyISAM tables. Merge tables help manage large volumes of data more easily. It logically groups a series of identical MyISAM tables, and references them as one object. Good for data warehousing environments.

Archive storage engine is optimised for high speed inserting. It compresses data as it is inserted. It does not support transactions. It is ideal for storing and retrieving large amounts of seldom referenced historical, archived data.

The **Blackhole** storage engine accepts but does not store data. Retrievals always return an empty set. The functionality can be used in distributed database design where data is automatically replicated, but not stored locally. This storage engine can be used to perform performance tests or other testing.

Federated storage engine offers the ability to separate MySQL servers to create one logical database from many physical servers. Queries on the local server are automatically executed on the remote (federated) tables. No data is stored on the local tables. It is good for distributed environments.

→InnoDB VS MyISAM

InnoDB	MyISAM
Default storage engine as of MySQL 5.5	Default storage engine before MySQL 5.5
ACID* compliant	Not ACID compliant
Transactional (Rollback, Commit)	Non-transactional
Row Level Locking	Table Level Locking
Row data stored in pages as per Primary Key order	No Particular order for data stored
Supports Foreign Keys	Does not support relationship constraint
No Full Text Search	Full Text Search

3) MySQL Stored Procedure

A procedure (often called a stored procedure) is a **collection of pre-compiled SQL statements** stored inside the database. It is a subroutine or a subprogram in the regular computing language. **A procedure always contains a name, parameter lists, and SQL statements.** We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc. It was first introduced in MySQL **version 5**. Presently, it can be supported by almost all relational database systems.

If we consider the enterprise application, we always need to perform specific tasks such as database cleanup, processing payroll, and many more on the database regularly. Such tasks involve multiple SQL statements for executing each task. This process might be easy if we group these tasks into a single task. We can fulfill this requirement in MySQL by creating a stored procedure in our database.

A procedure is called a **recursive stored procedure** when it calls itself. Most database systems support recursive stored procedures. But, it is not supported well in MySQL.

Stored Procedure Features

- Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.
- Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.
- Stored procedures are reusable and transparent to any applications.
- A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

How to create a procedure?

The following syntax is used for creating a stored procedure in MySQL. It can return one or more value through parameters or sometimes may not return at all. By default, a procedure is associated with our current database. But we can also create it into another database from the current database by specifying the name as **database_name.procedure_name**. See the complete syntax:

DELIMITER &&

CREATE PROCEDURE procedure_name [[IN | **OUT** | INOUT] parameter_name datatype [, parameter datatype]]]

BEGIN

Declaration_section

Executable_section

END &&

DELIMITER ;

3)The MySQL LIMIT Clause

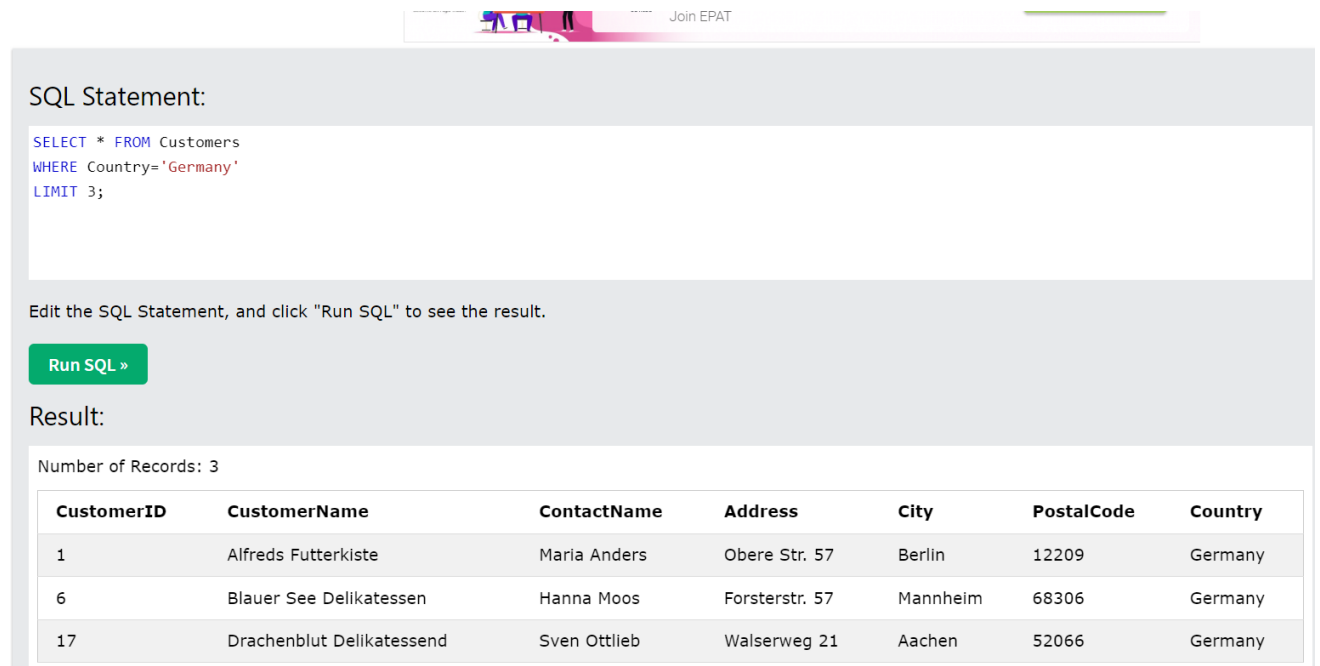
The LIMIT clause is used to specify the number of records to return.

The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

E.g,



The screenshot shows a web interface for running SQL queries. At the top, there is a navigation bar with a logo and a 'Join EPAT' button. Below the navigation bar, the 'SQL Statement:' section contains the following query:

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

 Below the query, there is a green button labeled 'Run SQL »'. The 'Result:' section shows the output of the query, which is a table with 7 columns: CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country. The table contains 3 rows of data.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany

4)The MySQL LIKE Operator

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

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

The percent sign and the underscore can also be used in combinations!

LIKE Syntax

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

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

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length

Vrushank_PHP

WHERE CustomerName LIKE 'a__%'

Finds any values that start with "a" and are at least 3 characters in length

WHERE ContactName LIKE 'a%o'

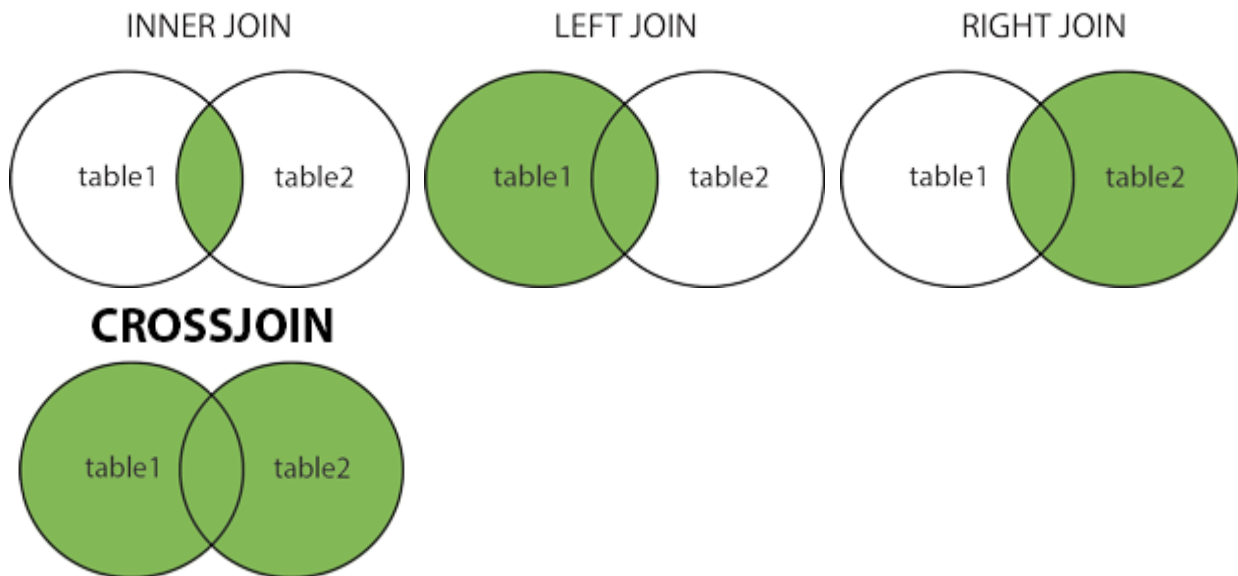
Finds any values that start with "a" and ends with "o"

5)MySQL Joining Tables

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

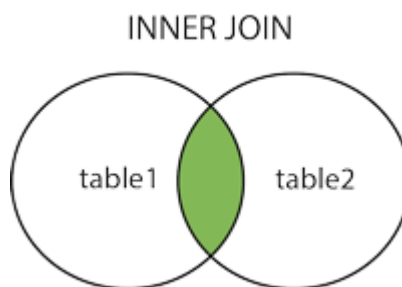
Supported Types of Joins in MySQL

- INNER JOIN: Returns records that have matching values in both tables
- LEFT JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT JOIN: Returns all records from the right table, and the matched records from the left table
- CROSS JOIN: Returns all records from both tables



MySQL INNER JOIN Keyword

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

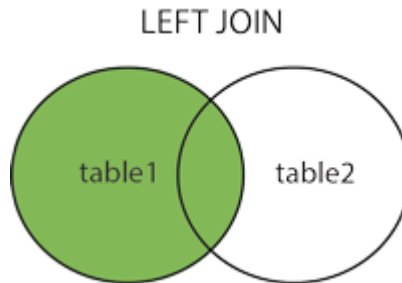


INNER JOIN Syntax

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


MySQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).

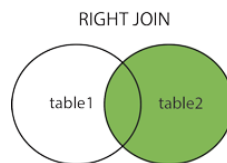


LEFT JOIN Syntax

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

MySQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).

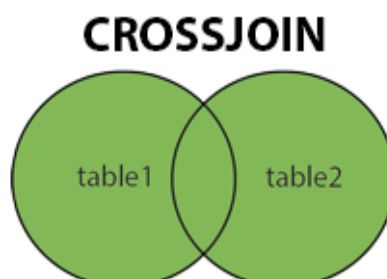


RIGHT JOIN Syntax

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

SQL CROSS JOIN Keyword

The CROSS JOIN keyword returns all records from both tables (table1 and table2).



CROSS JOIN Syntax

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

Note: CROSS JOIN can potentially return very large result-sets!

6)The MySQL 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);
```

The screenshot shows a MySQL database interface. On the left, the 'SQL Statement:' section contains the following code:

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

Below the code is a green 'Run SQL' button. The 'Result:' section shows the output of the query:

Number of Records: 21

COUNT(CustomerID)	Country
13	USA
11	France
11	Germany
9	Brazil
7	UK
5	Spain

On the right side of the interface, the 'MySQL Database:' section lists the tables and their record counts:

Tablenames	Records
Customers	91
Categories	8
Employees	9
OrderDetails	2155
Orders	830
Products	77
Shippers	3
Suppliers	29

7)The MySQL HAVING Clause

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

HAVING Syntax

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

SQL Statement:

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

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

MySQL Database:

Tablenames	Records
Customers	
Categories	
Employees	
OrderDetails	21
Orders	8
Products	
Shippers	
Suppliers	

Result:

Number of Records: 5

COUNT(CustomerID)	Country
13	USA
11	Germany
11	France
9	Brazil
7	UK

Having Without group by

A query with a *having* clause should also have a *group by* clause. If you omit *group by*, all the rows not excluded by the *where* clause return as a single group.

Because no grouping is performed between the *where* and *having* clauses, they cannot act independently of each other. *having* acts like *where* because it affects the rows in a single group rather than groups, except the *having* clause can still use aggregates.

This example uses the *having* clause averages the price, excludes from the results titles with advances greater than \$4,000, and produces results where price is less than the average price:

```
select title_id, advance, price
from titles
where advance < 4000
having price > avg(price)
```

title_id	advance	price
BU1032	5,000.00	19.99
BU7832	5,000.00	19.99
MC2222	0.00	19.99
PC1035	7,000.00	22.95
PC8888	8,000.00	20.00
PS1372	7,000.00	21.59
PS3333	2,000.00	19.99
TC3218	7,000.00	20.95

(8 rows affected)

Vrushank_PHP

You can also use the *having* clause with the Transact-SQL extension that allows you to omit the *group by* clause from a query that includes an aggregate in its select list. These scalar aggregate functions calculate values for the table as a single group, not for groups within the table.

In this example, the *group by* clause is omitted, which makes the aggregate function calculate a value for the entire table. The *having* clause excludes non-matching rows from the result group.

```
select pub_id, count(pub_id)
from publishers
having pub_id < "1000"
```

```
pub_id
-----
0736          3
0877          3

(2 rows affected)
```

8) MySQL Operators

MySQL Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

MySQL Bitwise Operators

Operator	Description
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR

MySQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

MySQL Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
*=	Bitwise OR equals

MySQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

9)MySQL Data Types

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

MySQL Data Types (Version 8.0)

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. 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.

String Data Types

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The <i>size</i> parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The <i>size</i> parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The <i>size</i> parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The <i>size</i> parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data

ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric Data Types

Data type	Description
BIT(<i>size</i>)	A bit-value type. The number of bits per value is specified in <i>size</i> . The <i>size</i> parameter can hold a value from 1 to 64. The default value for <i>size</i> is 1.
TINYINT(<i>size</i>)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The <i>size</i> parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(<i>size</i>)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The <i>size</i> parameter specifies the maximum display width (which is 255)
MEDIUMINT(<i>size</i>)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The <i>size</i> parameter specifies the maximum display width (which is 255)
INT(<i>size</i>)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The <i>size</i> parameter specifies the maximum display width (which is 255)
INTEGER(<i>size</i>)	Equal to INT(<i>size</i>)
BIGINT(<i>size</i>)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The <i>size</i> parameter specifies the maximum display width (which is 255)

FLOAT(<i>size</i> , <i>d</i>)	A floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(<i>p</i>)	A floating point number. MySQL uses the <i>p</i> value to determine whether to use FLOAT or DOUBLE for the resulting data type. If <i>p</i> is from 0 to 24, the data type becomes FLOAT(). If <i>p</i> is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(<i>size</i> , <i>d</i>)	A normal-size floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter
DOUBLE PRECISION(<i>size</i> , <i>d</i>)	
DECIMAL(<i>size</i> , <i>d</i>)	An exact fixed-point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. The maximum number for <i>size</i> is 65. The maximum number for <i>d</i> is 30. The default value for <i>size</i> is 10. The default value for <i>d</i> is 0.
DEC(<i>size</i> , <i>d</i>)	Equal to DECIMAL(<i>size</i> , <i>d</i>)

Note: All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

Date and Time Data Types

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(<i>fsp</i>)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(<i>fsp</i>)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(<i>fsp</i>)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Float	Double
This is generally used for graphic based libraries for making the processing power of your programs faster, as it is simpler to manage by compilers.	This is the most commonly used data type in programming languages for assigning values having a real or decimal based number within, such as 3.14 for pi.
It has single precision.	It has the double precision or you can say two times more precision than float.
According to IEEE, it has a 32-bit floating point precision.	According to IEEE, it has a 64-bit floating point precision.
Float takes 4 bytes for storage.	Double takes 8 bytes for storage.
A value having a range within 1.2E-38 to 3.4E+38 can be assigned to float variables.	A value having range within 2.3E-308 to 1.7E+308 can be assigned to double type variables
Has a precision of 6 decimal places.	Has a precision of 15 decimal places.