Чисельні типи даних

Зміст

[Чисельні типи даних 1](#_Toc57881458)

[Синтаксис чисельних типів даних 1](#_Toc57881459)

[Цілі типи (точні значення) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT 4](#_Toc57881460)

[Точні числа - DECIMAL, NUMERIC 5](#_Toc57881461)

[Апроксимовані типи, або типи з плаваючою комою Floating-Point Types (Approximate Value) - FLOAT, DOUBLE 5](#_Toc57881462)

[Бітові типи Value Type - BIT 6](#_Toc57881463)

[Атрібути чисельних типів Numeric Type Attributes 6](#_Toc57881464)

[Вихід значень за діапазон Out-of-Range and Overflow Handling 7](#_Toc57881465)

[Порівняння даних 9](#_Toc57881466)

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types ([INTEGER](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), [SMALLINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html), and [NUMERIC](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html)), as well as the approximate numeric data types ([FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html), [REAL](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html), and [DOUBLE PRECISION](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html)). The keyword [INT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) is a synonym for [INTEGER](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), and the keywords [DEC](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) and [FIXED](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) are synonyms for [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html). MySQL treats [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) as a synonym for [DOUBLE PRECISION](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) (a nonstandard extension). MySQL also treats [REAL](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) as a synonym for [DOUBLE PRECISION](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) (a nonstandard variation), unless the [REAL\_AS\_FLOAT](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_real_as_float) SQL mode is enabled.

The [BIT](https://dev.mysql.com/doc/refman/8.0/en/bit-type.html) data type stores bit values and is supported for [MyISAM](https://dev.mysql.com/doc/refman/8.0/en/myisam-storage-engine.html%22%20%5Co%20%2216.2%C2%A0The%20MyISAM%20Storage%20Engine), [MEMORY](https://dev.mysql.com/doc/refman/8.0/en/memory-storage-engine.html), [InnoDB](https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html%22%20%5Co%20%22Chapter%C2%A015%C2%A0The%20InnoDB%20Storage%20Engine), and [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) tables.

.

The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them.

###  Синтаксис чисельних типів даних

For integer data types, ***M*** indicates the maximum display width. The maximum display width is 255.

For floating-point and fixed-point data types, ***M*** is the total number of digits that can be stored.

As of MySQL 8.0.17, the display width attribute is deprecated for integer data types; you should expect support for it to be removed in a future version of MySQL.

If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column.

As of MySQL 8.0.17, the ZEROFILL attribute is deprecated for numeric data types; you should expect support for it to be removed in a future version of MySQL. Consider using an alternative means of producing the effect of this attribute. For example, applications could use the [LPAD()](https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_lpad) function to zero-pad numbers up to the desired width, or they could store the formatted numbers in [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) columns.

Numeric data types that permit the UNSIGNED attribute also permit SIGNED. However, these data types are signed by default, so the SIGNED attribute has no effect.

As of MySQL 8.0.17, the UNSIGNED attribute is deprecated for columns of type [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html), [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html), and [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) (and any synonyms); you should expect support for it to be removed in a future version of MySQL. Consider using a simple CHECK constraint instead for such columns.

SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE.

SERIAL DEFAULT VALUE in the definition of an integer column is an alias for NOT NULL AUTO\_INCREMENT UNIQUE.

**Warning**

When you use subtraction between integer values where one is of type UNSIGNED, the result is unsigned unless the [NO\_UNSIGNED\_SUBTRACTION](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_no_unsigned_subtraction) SQL mode is enabled.

* [BIT[(](https://dev.mysql.com/doc/refman/8.0/en/bit-type.html%22%20%5Co%20%2211.1.5%C2%A0Bit-Value%20Type%20-%20BIT)***[M](https://dev.mysql.com/doc/refman/8.0/en/bit-type.html%22%20%5Co%20%2211.1.5%C2%A0Bit-Value%20Type%20-%20BIT)***[)]](https://dev.mysql.com/doc/refman/8.0/en/bit-type.html%22%20%5Co%20%2211.1.5%C2%A0Bit-Value%20Type%20-%20BIT)

A bit-value type. ***M*** indicates the number of bits per value, from 1 to 64. The default is 1 if ***M*** is omitted.

* [TINYINT[(](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[M](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)

A very small integer. The signed range is -128 to 127. The unsigned range is 0 to 255.

* [BOOL](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT), [BOOLEAN](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html)

These types are synonyms for [TINYINT(1)](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html). A value of zero is considered false. Nonzero values are considered true:

mysql> SELECT IF(0, 'true', 'false');

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

| IF(0, 'true', 'false') |

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

| false |

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

mysql> SELECT IF(1, 'true', 'false');

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

| IF(1, 'true', 'false') |

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

| true |

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

mysql> SELECT IF(2, 'true', 'false');

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

| IF(2, 'true', 'false') |

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

| true |

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

However, the values TRUE and FALSE are merely aliases for 1 and 0, respectively, as shown here:

mysql> SELECT IF(0 = FALSE, 'true', 'false');

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

| IF(0 = FALSE, 'true', 'false') |

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

| true |

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

mysql> SELECT IF(1 = TRUE, 'true', 'false');

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

| IF(1 = TRUE, 'true', 'false') |

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

| true |

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

mysql> SELECT IF(2 = TRUE, 'true', 'false');

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

| IF(2 = TRUE, 'true', 'false') |

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

| false |

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

mysql> SELECT IF(2 = FALSE, 'true', 'false');

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

| IF(2 = FALSE, 'true', 'false') |

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

| false |

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

The last two statements display the results shown because 2 is equal to neither 1 nor 0.

* [SMALLINT[(](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[M](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)

A small integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.

* [MEDIUMINT[(](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[M](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)

A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.

* [INT[(](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[M](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)

A normal-size integer. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295.

* [INTEGER[(](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[M](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)

This type is a synonym for [INT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html).

* [BIGINT[(](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[M](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html%22%20%5Co%20%2211.1.2%C2%A0Integer%20Types%20%28Exact%20Value%29%20-%20INTEGER%2C%20INT%2C%20SMALLINT%2C%20TINYINT%2C%20MEDIUMINT%2C%20BIGINT)

A large integer. The signed range is -9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615.

SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE.

Some things you should be aware of with respect to [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) columns:

* + All arithmetic is done using signed [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) or [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) values, so you should not use unsigned big integers larger than 9223372036854775807 (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) value to a [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html).

MySQL can handle [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) in the following cases:

* + - When using integers to store large unsigned values in a [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) column.
		- In [MIN(***col\_name***)](https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_min) or [MAX(***col\_name***)](https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_max), where ***col\_name*** refers to a [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) column.
		- When using operators ([+](https://dev.mysql.com/doc/refman/8.0/en/arithmetic-functions.html#operator_plus), [-](https://dev.mysql.com/doc/refman/8.0/en/arithmetic-functions.html#operator_minus), [\*](https://dev.mysql.com/doc/refman/8.0/en/arithmetic-functions.html#operator_times), and so on) where both operands are integers.
	+ You can always store an exact integer value in a [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
	+ The [-](https://dev.mysql.com/doc/refman/8.0/en/arithmetic-functions.html#operator_minus), [+](https://dev.mysql.com/doc/refman/8.0/en/arithmetic-functions.html#operator_plus), and [\*](https://dev.mysql.com/doc/refman/8.0/en/arithmetic-functions.html#operator_times) operators use [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than 9223372036854775807.

* [DECIMAL[(](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[M](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[[,](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[D](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[])] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)

A packed “exact” fixed-point number. ***M*** is the total number of digits (the precision) and ***D*** is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the - sign are not counted in ***M***. If ***D*** is 0, values have no decimal point or fractional part. The maximum number of digits (***M***) for [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) is 65. The maximum number of supported decimals (***D***) is 30. If ***D*** is omitted, the default is 0. If ***M*** is omitted, the default is 10.

UNSIGNED, if specified, disallows negative values. As of MySQL 8.0.17, the UNSIGNED attribute is deprecated for columns of type [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) (and any synonyms); you should expect support for it to be removed in a future version of MySQL. Consider using a simple CHECK constraint instead for such columns.

All basic calculations (+, -, \*, /) with [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) columns are done with a precision of 65 digits.

* [DEC[(](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[M](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[[,](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[D](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC)***[])] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html%22%20%5Co%20%2211.1.3%C2%A0Fixed-Point%20Types%20%28Exact%20Value%29%20-%20DECIMAL%2C%20NUMERIC), [NUMERIC[(***M***[,***D***])] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html), [FIXED[(***M***[,***D***])] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html)

These types are synonyms for [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html). The [FIXED](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) synonym is available for compatibility with other database systems.

* [FLOAT[(](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[M](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[,](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[D](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)

A small (single-precision) floating-point number. Permissible values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

***M*** is the total number of digits and ***D*** is the number of digits following the decimal point. If ***M*** and ***D*** are omitted, values are stored to the limits permitted by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

FLOAT(***M***,***D***) is a nonstandard MySQL extension. As of MySQL 8.0.17, this syntax is deprecated, and you should expect support for it to be removed in a future version of MySQL.

UNSIGNED, if specified, disallows negative values. As of MySQL 8.0.17, the UNSIGNED attribute is deprecated for columns of type [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple CHECK constraint instead for such columns.

Using [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) might give you some unexpected problems because all calculations in MySQL are done with double precision.

* [FLOAT(](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[p](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[) [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)

A floating-point number. ***p*** represents the precision in bits, but MySQL uses this value only to determine whether to use [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) or [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) for the resulting data type. If ***p*** is from 0 to 24, the data type becomes [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) with no ***M*** or ***D*** values. If ***p*** is from 25 to 53, the data type becomes [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) with no ***M*** or ***D*** values. The range of the resulting column is the same as for the single-precision [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) or double-precision [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) data types described earlier in this section.

UNSIGNED, if specified, disallows negative values. As of MySQL 8.0.17, the UNSIGNED attribute is deprecated for columns of type [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple CHECK constraint instead for such columns.

[FLOAT(](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[p](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[)](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE) syntax is provided for ODBC compatibility.

* [DOUBLE[(](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[M](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[,](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[D](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)

A normal-size (double-precision) floating-point number. Permissible values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

***M*** is the total number of digits and ***D*** is the number of digits following the decimal point. If ***M*** and ***D*** are omitted, values are stored to the limits permitted by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

DOUBLE(***M***,***D***) is a nonstandard MySQL extension. As of MySQL 8.0.17, this syntax is deprecated and you should expect support for it to be removed in a future version of MySQL.

UNSIGNED, if specified, disallows negative values. As of MySQL 8.0.17, the UNSIGNED attribute is deprecated for columns of type [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple CHECK constraint instead for such columns.

* [DOUBLE PRECISION[(](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[M](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[,](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[D](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE)***[)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html%22%20%5Co%20%2211.1.4%C2%A0Floating-Point%20Types%20%28Approximate%20Value%29%20-%20FLOAT%2C%20DOUBLE), [REAL[(***M***,***D***)] [UNSIGNED] [ZEROFILL]](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html)

These types are synonyms for [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html). Exception: If the [REAL\_AS\_FLOAT](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_real_as_float) SQL mode is enabled, [REAL](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) is a synonym for [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) rather than [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html).

### Цілі типи (точні значення) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

MySQL supports the SQL standard integer types INTEGER (or INT) and SMALLINT. As an extension to the standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT. The following table shows the required storage and range for each integer type.

**Table 11.1 Required Storage and Range for Integer Types Supported by MySQL**

| **Type** | **Storage (Bytes)** | **Minimum Value Signed** | **Minimum Value Unsigned** | **Maximum Value Signed** | **Maximum Value Unsigned** |
| --- | --- | --- | --- | --- | --- |
| TINYINT | 1 | -128 | 0 | 127 | 255 |
| SMALLINT | 2 | -32768 | 0 | 32767 | 65535 |
| MEDIUMINT | 3 | -8388608 | 0 | 8388607 | 16777215 |
| INT | 4 | -2147483648 | 0 | 2147483647 | 4294967295 |
| BIGINT | 8 | -263 | 0 | 263-1 | 264-1 |

###  Точні числа - DECIMAL, NUMERIC

The DECIMAL and NUMERIC types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, NUMERIC is implemented as DECIMAL, so the following remarks about DECIMAL apply equally to NUMERIC.

MySQL stores DECIMAL values in binary format.

In a DECIMAL column declaration, the precision and scale can be (and usually is) specified. For example:

salary DECIMAL(5,2)

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

Standard SQL requires that DECIMAL(5,2) be able to store any value with five digits and two decimals, so values that can be stored in the salary column range from -999.99 to 999.99.

In standard SQL, the syntax DECIMAL(***M***) is equivalent to DECIMAL(***M***,0). Similarly, the syntax DECIMAL is equivalent to DECIMAL(***M***,0), where the implementation is permitted to decide the value of ***M***. MySQL supports both of these variant forms of DECIMAL syntax. The default value of ***M*** is 10.

If the scale is 0, DECIMAL values contain no decimal point or fractional part.

The maximum number of digits for DECIMAL is 65, but the actual range for a given DECIMAL column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are permitted by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the permissible number of digits.)

### Апроксимовані типи, або типи з плаваючою комою Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

The FLOAT and DOUBLE types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

For FLOAT, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword FLOAT in parentheses; ; that is, [FLOAT(***p***)](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html). MySQL also supports this optional precision specification, but the precision value in [FLOAT(***p***)](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) is used only to determine storage size. A precision from 0 to 23 results in a 4-byte single-precision FLOAT column. A precision from 24 to 53 results in an 8-byte double-precision DOUBLE column.

MySQL permits a nonstandard syntax: FLOAT(***M***,***D***) or REAL(***M***,***D***) or DOUBLE PRECISION(***M***,***D***). Here, (***M***,***D***) means than values can be stored with up to ***M*** digits in total, of which ***D*** digits may be after the decimal point. For example, a column defined as FLOAT(7,4) is displayed as -999.9999. MySQL performs rounding when storing values, so if you insert 999.00009 into a FLOAT(7,4) column, the approximate result is 999.0001.

As of MySQL 8.0.17, the nonstandard FLOAT(***M***,***D***) and DOUBLE(***M***,***D***) syntax is deprecated and you should expect support for it to be removed in a future version of MySQL.

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies.

For maximum portability, code requiring storage of approximate numeric data values should use FLOAT or DOUBLE PRECISION with no specification of precision or number of digits.

### Бітові типи Value Type - BIT

The BIT data type is used to store bit values. A type of BIT(***M***) enables storage of ***M***-bit values. ***M*** can range from 1 to 64.

To specify bit values, b'***value***' notation can be used. ***value*** is a binary value written using zeros and ones. For example, b'111' and b'10000000' represent 7 and 128, respectively.

If you assign a value to a BIT(***M***) column that is less than ***M*** bits long, the value is padded on the left with zeros. For example, assigning a value of b'101' to a BIT(6) column is, in effect, the same as assigning b'000101'.

**NDB Cluster.** The maximum combined size of all BIT columns used in a given [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) table must not exceed 4096 bits.

### Атрібути чисельних типів Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, [INT(4)](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) specifies an [INT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used is up to the application.)

The display width does not constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as [SMALLINT(3)](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) has the usual [SMALLINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) range of -32768 to 32767, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) ZEROFILL attribute, the default padding of spaces is replaced with zeros. For example, for a column declared as [INT(4) ZEROFILL](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), a value of 5 is retrieved as 0005.

**Note**

The ZEROFILL attribute is ignored for columns involved in expressions or [UNION](https://dev.mysql.com/doc/refman/8.0/en/union.html) queries.

If you store values larger than the display width in an integer column that has the ZEROFILL attribute, you may experience problems when MySQL generates temporary tables for some complicated joins. In these cases, MySQL assumes that the data values fit within the column display width.

As of MySQL 8.0.17, the ZEROFILL attribute is deprecated for numeric data types, as is the display width attribute for integer data types. You should expect support for ZEROFILL and display widths for integer data types to be removed in a future version of MySQL. Consider using an alternative means of producing the effect of these attributes. For example, applications can use the [LPAD()](https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_lpad) function to zero-pad numbers up to the desired width, or they can store the formatted numbers in [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) columns.

All integer types can have an optional (nonstandard) UNSIGNED attribute. An unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the column. For example, if an [INT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) column is UNSIGNED, the size of the column's range is the same but its endpoints shift up, from -2147483648 and 2147483647 to 0 and 4294967295.

Floating-point and fixed-point types also can be UNSIGNED. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same. As of MySQL 8.0.17, the UNSIGNED attribute is deprecated for columns of type [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html), [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html), and [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple CHECK constraint instead for such columns.

If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute.

Integer or floating-point data types can have the AUTO\_INCREMENT attribute. When you insert a value of NULL into an indexed AUTO\_INCREMENT column, the column is set to the next sequence value. Typically this is ***value***+1, where ***value*** is the largest value for the column currently in the table. (AUTO\_INCREMENT sequences begin with 1.)

Storing 0 into an AUTO\_INCREMENT column has the same effect as storing NULL, unless the [NO\_AUTO\_VALUE\_ON\_ZERO](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_no_auto_value_on_zero) SQL mode is enabled.

Inserting NULL to generate AUTO\_INCREMENT values requires that the column be declared NOT NULL. If the column is declared NULL, inserting NULL stores a NULL. When you insert any other value into an AUTO\_INCREMENT column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the inserted value.

Negative values for AUTO\_INCREMENT columns are not supported.

CHECK constraints cannot refer to columns that have the AUTO\_INCREMENT attribute, nor can the AUTO\_INCREMENT attribute be added to existing columns that are used in CHECK constraints.

As of MySQL 8.0.17, AUTO\_INCREMENT support is deprecated for [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) and [DOUBLE](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) columns; you should expect it to be removed in a future version of MySQL. Consider removing the AUTO\_INCREMENT attribute from such columns, or convert them to an integer type.

### Вихід значень за діапазон Out-of-Range and Overflow Handling

When MySQL stores a value in a numeric column that is outside the permissible range of the column data type, the result depends on the SQL mode in effect at the time:

* If strict SQL mode is enabled, MySQL rejects the out-of-range value with an error, and the insert fails, in accordance with the SQL standard.
* If no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the column data type range and stores the resulting value instead.

When an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range.

When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Suppose that a table t1 has this definition:

CREATE TABLE t1 (i1 TINYINT, i2 TINYINT UNSIGNED);

With strict SQL mode enabled, an out of range error occurs:

mysql> SET sql\_mode = 'TRADITIONAL';

mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);

ERROR 1264 (22003): Out of range value for column 'i1' at row 1

mysql> SELECT \* FROM t1;

Empty set (0.00 sec)

With strict SQL mode not enabled, clipping with warnings occurs:

mysql> SET sql\_mode = '';

mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);

mysql> SHOW WARNINGS;

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

| Level | Code | Message |

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

| Warning | 1264 | Out of range value for column 'i1' at row 1 |

| Warning | 1264 | Out of range value for column 'i2' at row 1 |

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

mysql> SELECT \* FROM t1;

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

| i1 | i2 |

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

| 127 | 255 |

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

When strict SQL mode is not enabled, column-assignment conversions that occur due to clipping are reported as warnings for [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html), [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html), [UPDATE](https://dev.mysql.com/doc/refman/8.0/en/update.html), and multiple-row [INSERT](https://dev.mysql.com/doc/refman/8.0/en/insert.html) statements. In strict mode, these statements fail, and some or all the values are not inserted or changed, depending on whether the table is a transactional table and other factors..

Overflow during numeric expression evaluation results in an error. For example, the largest signed [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) value is 9223372036854775807, so the following expression produces an error:

mysql> SELECT 9223372036854775807 + 1;

ERROR 1690 (22003): BIGINT value is out of range in '(9223372036854775807 + 1)'

To enable the operation to succeed in this case, convert the value to unsigned;

mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;

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

| CAST(9223372036854775807 AS UNSIGNED) + 1 |

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

| 9223372036854775808 |

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

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) values have a larger range than integers:

mysql> SELECT 9223372036854775807.0 + 1;

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

| 9223372036854775807.0 + 1 |

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

| 9223372036854775808.0 |

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

Subtraction between integer values, where one is of type UNSIGNED, produces an unsigned result by default. If the result would otherwise have been negative, an error results:

mysql> SET sql\_mode = '';

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;

ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'

If the [NO\_UNSIGNED\_SUBTRACTION](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_no_unsigned_subtraction) SQL mode is enabled, the result is negative:

mysql> SET sql\_mode = 'NO\_UNSIGNED\_SUBTRACTION';

mysql> SELECT CAST(0 AS UNSIGNED) - 1;

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

| CAST(0 AS UNSIGNED) - 1 |

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

| -1 |

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

If the result of such an operation is used to update an UNSIGNED integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if [NO\_UNSIGNED\_SUBTRACTION](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_no_unsigned_subtraction) is enabled. If strict SQL mode is enabled, an error occurs and the column remains unchanged.

### Порівняння даних

**Якщо в програмі використовуються цілі типи даних , то ми можемо легко їх порівняти :**

int a,b,c;

a= b= якась послідовність обчислень, що повертає результат (int short long)

b= якась послідовність обчислень, що повертає результат (int short long)

If(a==b) c=10;

else c=20;

 умовний оператор завжди виконується, коли a буде дорівнювати b

Інші операції порівняння < > <= >= також в такій програмі будуть завжди виконуватися

**Зовсім інакше треба порівнювати числа, що оперують даними з обмеженою точністю**

double a,b,c;

a= b= якась послідовність обчислень, що повертає результат з обмеженою точністю (decimal float double long double)

b= якась послідовність обчислень, що повертає результат (decimal float double long double)

If(a==b) c=10;

else c=20;

**може статися так, що оператор** If(a==b) c=10 ніколи не виконається.

Тому, що , наприклад, a=10 b = 9.999998 float

Або , наприклад, a=10.0000000000002 b = 9.999999999999 float

Але для нас, фактично, ці числа еквівалентні

Тому нам треба зробити так:

Визначити відносну точність для числа. Наприклад, для float Е=10-6 для double Е=10-16

Визначити абсолютну точність результату дельтаA = E \* a дельтаB = E \* b

**Тоді нам треба порівнювати не числа, а інтервали:**

 **[ a-** дельтаA; **a+** дельтаA] **[ b-** дельтаB; **b**+дельтаB]

 **Якщо інтервали перетинаються, це і буде умовою порівняння.**

**Необхідність порівняння інтервалів обумовлена накопиченням помилок.**

 **Інші операції:> < <= >= будуть виконуватися завжди**

**Умовний оператор буде таким**

**If(b-** дельтаB>= **a-** дельтаA && **b-** дельтаB <= **a+** дельтаA &&

 **a+** дельтаA >= **b-** дельтаB && **a+** дельтаA <= **b**+дельтаB ||

**a-** дельтаA>= **b-** дельтаB && **a-** дельтаA <= **b+** дельтаB &&

 **b+** дельтаB >= **a-** дельтаA && **b+** дельтаAB<= **a**+дельтаA) c=10; else c=20;

Цей же оператор можливо записати простіше:

**If( !(b-**+дельтаB <  **a-** дельтаA && **a+** дельтаA < **b-** дельтаB) ) c=10; else c=20;