# MySQL INT (INTEGER) Data Types with Different Examples

 September 6th, 2021

In the article, you will learn about SIGNED and UNSIGNED integer data types in MySQL. The article provides information about integer types with the ZEROFILL, AUTO\_INCREMENT, and display\_width attributes, as well as describes examples of when and how to use integer data types.

### Contents

* [What is MySQL INTEGER](https://blog.devart.com/mysql-int-data-type.html" \l "what_is_mysql_integer)
* [SIGNED vs UNSIGNED](https://blog.devart.com/mysql-int-data-type.html" \l "signed_vs_unsigned)
* [What is UNSIGNED INT Data Type](https://blog.devart.com/mysql-int-data-type.html" \l "what_is_unsigned_int_data_type)
* [INTEGER Display Width: INT()](https://blog.devart.com/mysql-int-data-type.html" \l "integer_display_width)
* [MySQL INT with the ZEROFILL attribute](https://blog.devart.com/mysql-int-data-type.html" \l "mysql_int_with_zerofill_attribute)
* [MySQL INT Types](https://blog.devart.com/mysql-int-data-type.html" \l "mysql_int_types)
* [TINYINT Example](https://blog.devart.com/mysql-int-data-type.html" \l "tinyint_example)
* [SMALLINT Example](https://blog.devart.com/mysql-int-data-type.html" \l "smallint_example)
* [MEDIUMINT Example](https://blog.devart.com/mysql-int-data-type.html" \l "mediumint_example)
* [BIGINT Example](https://blog.devart.com/mysql-int-data-type.html" \l "bigint_example)

## What is MySQL INTEGER

In MySQL, INTEGER (INT) is a numeric value without a decimal. It defines whole numbers that can be stored in a field or column. In addition, MySQL supports the display\_width attribute (for example, INT(1)) and the ZEROFILL attribute, which automatically adds zeros to the value depending on the display width.

Except for standard INT and SMALLINT data types, MySQL supports TINYINT, MEDIUMINT, and BIGINT numeric data types that can be SIGNED and UNSIGNED.

The table displays the allowable storage in bytes, the maximum and minimum values of SIGNED and UNSIGNED integer data types.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

Usually, INTEGER data types are used for representing IDs, date and time (for example, DATETIME, DATE, YEAR, TIMESTAMP), width and height attributes, etc.

## SIGNED vs UNSIGNED

As mentioned before, INTEGER data types can be UNSIGNED and SIGNED. ****UNSIGNED****can store only zero and positive numbers in a column. If the values in the column never use negative numbers or when there is a need in using a larger upper numeric range for the column, you can set UNSIGNED, which in turn may lead to a faster index.

****SIGNED**** can allow zero, positive, and negative numbers. With the MySQL value as SIGNED, one bit is taken for the sign symbol as opposed to the unsigned ones, when they do not have signs preceding the value.

In MySQL, INT is usually selected as a primary key and can have the AUTO\_INCREMENT attribute in the column. For example, when you add NULL or 0 to the INT AUTO\_INCREMENT attribute, the column value is automatically incremented and changed to the next sequential value generated in the ascending order. When you add a non-zero value to the INT AUTO\_INCREMENT attribute, the value is accepted and the sequence is reset meaning that the inserted value precedes the automatically generated sequential values (i.e. value+1). Keep in mind that columns with AUTO\_INCREMENT do not store negative values, which means that columns with AUTO\_INCREMENT are ****unsigned****by default.

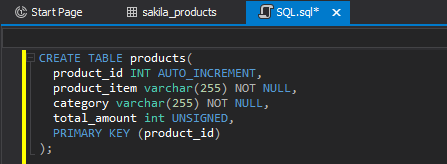
## What is UNSIGNED INT Data Type?

When we set the MySQL INT UNSIGNED column, it means that negative values cannot be inserted in this column. The maximum range value with the UNSIGNED data type is 4294967295, while the minimum one is 0. Thus, if you insert a negative value or one that exceeds the maximum value, you will get an error since it will be an out-of-the-range value. Let’s see the example of how to create a table with the INT UNSIGNED columns. Since ID is usually a numeric data type, we’ll use it as an INT UNSIGNED primary key.

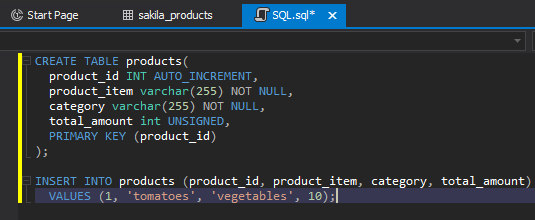
### Create a table with UNSIGNED data example

For a demo example, we will use dbForge Studio for MySQL.

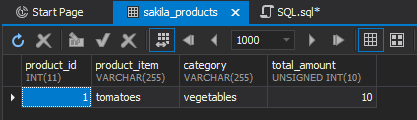
To begin with, create the ****products**** table having total\_amount as the INT UNSIGNED data type. For this, on the standard toolbar, click ****New SQL****, type the query, and click ****Execute****or press ****F5:****



Then, [insert a new row](https://blog.devart.com/mysql-insert-statement-inserting-row-into-a-table.html" \t "https://blog.devart.com/_blank) – VALUES – in the MySQL table and execute the query.



To check the result, retrieve the data. For this, in ****Database Explorer****, right-click the ****products****table and select ****Retrieve Data****.As you can see, the values were added.

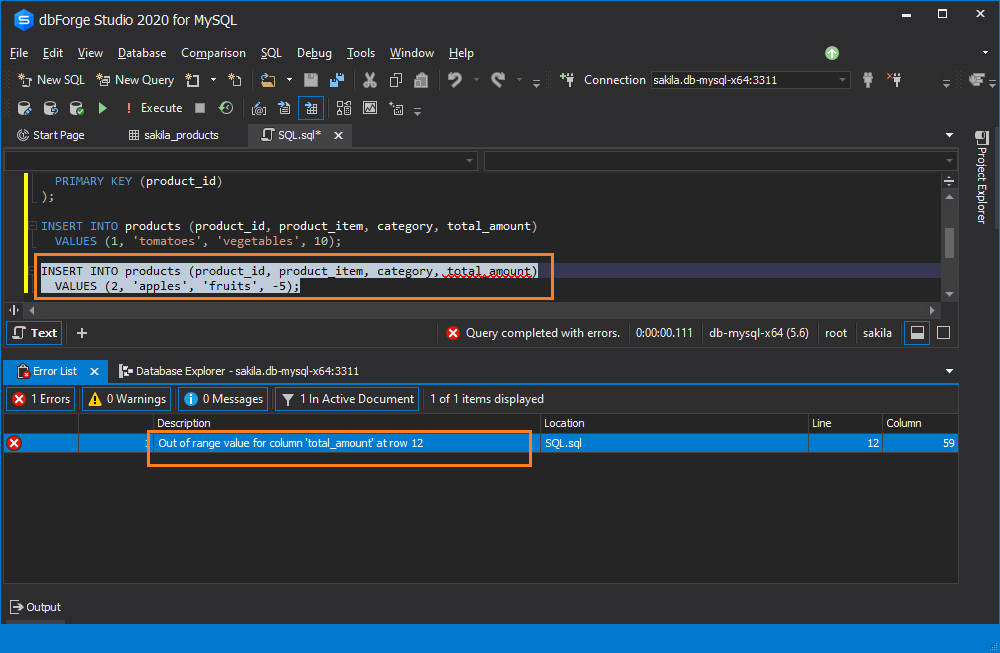


And now, insert a new row but with a negative number for the INT UNSIGNED column and execute the query:

INSERT INTO products (product\_id, product\_item, category, total\_amount)

VALUES (2, 'apples', 'fruits', -5);

As you can see, upon executing the query, we got an error because UNSIGNED does not support negative values:



## INTEGER Display Width: INT()

In MySQL, you can set the display width for the INT data type. The display width is the number in parentheses that represents the ****output length****, i.e. how many characters should be displayed in the column when retrieving data for the column. Thus, it matters only when displaying the output.

Consider the following: If you specify 5 as a value for the ****int(1), int(6), int(10),****and ****int(11****) columns, they all will output the same value – 5. Actually, display width makes difference when you set the ZEROFILL attribute. In this case, the result will display 5, 000005, 0000000005, and 00000000005 respectively.

Note that the display width does not influence the maximum and minimum value ranges. The minimum and maximum values of the column and the storage size (in bytes) depend only on the integer type you use but not on the length. The values ****int(1), int(6), int(10),****and ****int(11****) will have the maximum value in a range that equals 2147483647 (for signed INT) and 4294967295 (for unsigned INT).

## MySQL INT with the ZEROFILL attribute

If you need to replace spaces with zeros, you can use the ZEROFILL attribute for the MySQL INT UNSIGNED column. For example, for the INT(3) ZEROFILL column with the value 1, the result you retrieve will be 001.

**Note:** Keep in mind that UNSIGNED is automatically added to the numeric column.

Have a look at the example. We create a MySQL ****ProductsNew****table having columns with the ZEROFILL attribute.

In a new SQL document, execute the following CREATE statement:

CREATE TABLE ProductsNew(

product\_id INT AUTO\_INCREMENT,

product\_item varchar(255) NOT NULL,

category varchar(255) NOT NULL,

weight int(2) ZEROFILL,

review int(3) ZEROFILL,

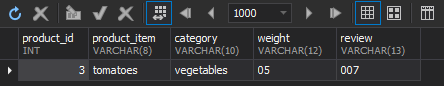
PRIMARY KEY (product\_id));

Then, insert a new row in the ****ProductsNew**** table using the INSERT INTO statement and execute it:

INSERT INTO ProductsNew (product\_id, product\_item, category, weight, review)

VALUES (3, 'tomatoes', 'vegetables', 5, 7);

Retrieve data and check the result.



Keep in mind that as for MySQL 8.0.17, the ZEROFILL attribute has been deprecated for numeric data types.

## MySQL INT Types

As it was mentioned earlier, MySQL supports SQL standard data types INT and SMALLINT. Besides, MySQL supports TINYINT, MEDIUMINT, and BIGINT. What integer data type to choose depends on the data you are going to store in the column. For example, if you are sure that the value in the column will not exceed the maximum value in the INT range – 255, then you can use the TINYINT data type, which, in turn, will improve database performance. However, if you need to store huge data such as client base that will constantly increase, it would be better to use INT or BIGINT. Also, if data stored in the column will never use negative values, UNSIGNED should be set. The INT range will be the same, for example, for INT(5), INT(1), INT(100).

Let’s have a look at each integer data type and examples of their usage.

## TINYINT Example

TINYINT is a very small integer. The minimum and maximum SIGNED values are -128 and 127 respectively, while for UNSIGNED values TINYINT range is from 0 to 255. TINYINT uses 1 byte per row. It is the best option when you want to save space on your disk and enhance performance.

Consider the example of how to use the TINYINT integer. Create a ****StudentGroup****table with the CREATE statement:

CREATE TABLE StudentGroup(

id tinyint AUTO\_INCREMENT,

Name varchar(255) NOT NULL,

Surname varchar(255) NOT NULL,

Grade tinyint(4) UNSIGNED,

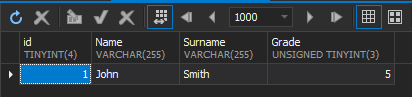
PRIMARY KEY (id));

Then, insert data in the column using the INSERT statement:

INSERT INTO StudentGroup (id, Name, Surname, Grade)

VALUES (1, 'John', 'Smith', 5);

After retrieving data, we see that the query was completed successfully.



But if we insert the value not included in the range and try to retrieve data,

INSERT INTO StudentGroup (id, Name, Surname, Grade)

VALUES (1, 'John', 'Smith', -5);

we’ll get the error ‘Out of range value for column ‘Grade’ at row 12′.

****Note**** that starting with MySQL 8.0, a display width was deprecated for TINYINT and you should take it into account when migrating to a new version. However, this deprecation did not affect TINYINT(1), but matters for TINYINT(4).

## SMALLINT Example

SMALLINT is a small integer. The SMALLINT range for SIGNED values is from -32768 to 32767. The minimum and maximum values for SMALLINT UNSIGNED are 0 and 65535 respectively. The size is 2 bytes per row.

An example of how to use SMALLINT is as follows. Create a ****SmallintTable****table using the CREATE statement:

CREATE TABLE SmallintTable(

ID smallint AUTO\_INCREMENT,

value\_1 smallint SIGNED,

value\_2 smallint(6) UNSIGNED,

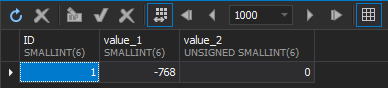
PRIMARY KEY (id));

Insert column values that correspond to the SMALLINT range and execute the query:

INSERT INTO SmallintTable (ID, value\_1, value\_2)

VALUES (1, -768, 0);

Upon retrieving data, the query was completed without errors.



Now, change the values by inserting data that is not included in the SMALLINT range.

INSERT INTO SmallintTable (ID, value\_1, value\_2)

VALUES (1, 0, 65536);

As a result, you will see an error message that the values are out of the range.

****Note**** that starting with MySQL 8.0, a display width was deprecated for SMALLINT and you should take it into account when migrating to a new version. However, this deprecation did not affect SMALLINT(1).

## MEDIUMINT Example

MEDIUMINT is a medium-sized integer. The minimum and maximum values for MEDIUMINT SIGNED are -8388608 and 8388607 respectively. The range for minimum and maximum UNSIGNED values equals 0 and 16777215 respectively. The size is 3 bytes per row.

An example of how to use MEDIUMINT is as follows. First, create a ****MediumIntTable****table with the CREATE statement.

CREATE TABLE MediumIntTable(

ID mediumint AUTO\_INCREMENT,

value\_1 mediumint,

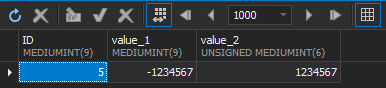
value\_2 mediumint(6) UNSIGNED,

PRIMARY KEY (ID));

Then, add correct values to the query, retrieve data to see that everything works correctly.

INSERT INTO MediumIntTable (ID, value\_1, value\_2)

VALUES (5, -1234567, 1234567);



After that, modify the values to the ones that exceed the allowable range.

-- insert a number in the value\_2 column that is not included in the MEDIUMINT range

INSERT INTO MediumIntTable (ID, value\_1, value\_2)

VALUES (5, 8388607, -16777215);

As a result, the error message informing that the value\_2 is out of the range appears.

****Note**** that starting with MySQL 8.0, a display width was deprecated for MEDIUMINT and you should take it into account when migrating to a new version. However, this deprecation did not affect MEDIUMINT(1).

## BIGINT Example

BIGINT is a large integer. The minimum and maximum values for SIGNED are – -263 and 263-1 respectively. The range for minimum and maximum values for UNSIGNED equals 0 and 264-1 respectively. The size is 8 bytes per row.

An example of how to use BIGINT is as follows. Let’s create a ****BigIntTable****table with BIGINT data type and the AUTO\_INCREMENT attribute for the ID column. Then, add to the query a negative number as a value\_1 and a positive number as a value\_2 that are included in the range.

--create a table

CREATE TABLE BigIntTable(

ID bigint AUTO\_INCREMENT,

value\_1 bigint,

value\_2 bigint(20) UNSIGNED,

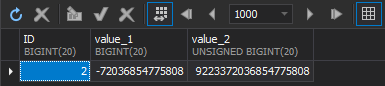
PRIMARY KEY (ID));

-- insert data

INSERT INTO BigIntTable (ID, value\_1, value\_2)

VALUES (2, -72036854775808, 9223372036854775808);

After that, retrieve data to make sure that it works correctly.



Once done, modify the values to the ones that exceed the allowable range.

-- modify the value\_2

INSERT INTO BigIntTable (ID, value\_1, value\_2)

VALUES (2, -72036854775808, 99223372036854775808);

As a result, the error message informing that the value\_2 is out of the range appears.

We would like to mention that using INT as a primary key, foreign key, or index instead of BIGINT can be a significant point in improving performance, saving disk space, and decreasing index size.

****Note**** that starting with MySQL 8.0, a display width was deprecated for BIGINT and you should take it into account when migrating to a new version. However, this deprecation did not affect BIGINT(1).

## Conclusion

In the article, we have reviewed MySQL INTEGER data types, SIGNED and UNSIGNED values, AUTO\_INCREMENT, ZEROFILL, and display\_width attributes. Depending on your projects, you should choose which INTEGER data type suits you most. Also, the article provides examples of how and when to INTEGER data types in MySQL.