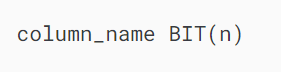
# **MySQL BIT Data Type**

**Summary**: in this tutorial, you will learn about MySQL BIT data type and how to store BIT data in a column of a table.

## **Introduction to MySQL BIT data type**

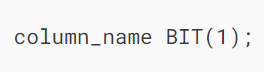
The BIT data type that allows you to store bit values, which are 0 and 1.

Here’s the syntax for defining BIT type of a column:



The BIT(n) can store up to n-bit values. The n can range from 1 to 64. The default value of n is 1 if you skip it.

So the following syntaxes are equivalent:



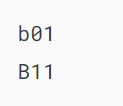
and

### 

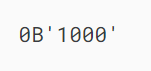
### **BIT literals**

To specify a bit value literal, you use b'val' or 0bval notation, which val is a binary value that contains only 0 and 1.

The leading b can be written as B, for example, the following are valid bit literals:



However, the leading 0b is case-sensitive. Therefore, you cannot use 0B. For example, the following is an invalid bit literal value:

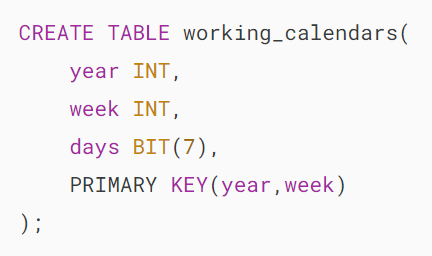


By default the [character set](https://www.mysqltutorial.org/mysql-basics/mysql-character-set/) of a bit-value literal is the binary string as follows:

## 

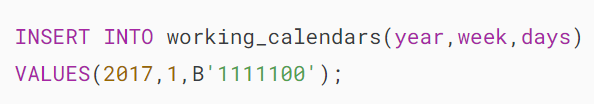
## **MySQL BIT data type examples**

The following statement [creates a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named working\_calendars that has the days column is BIT(7):

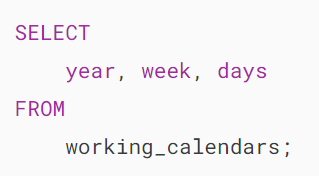


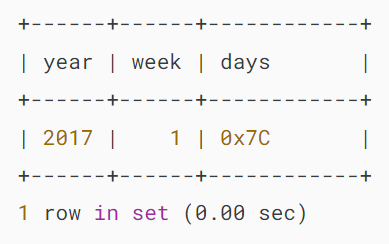
The values in the column days indicate whether the day is a working day or day off i.e., 1: working day and 0: day off.

Suppose that Saturday and Friday of the first week of 2017 are not working days, you can [insert a row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the working\_calendars table:

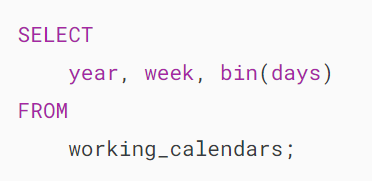


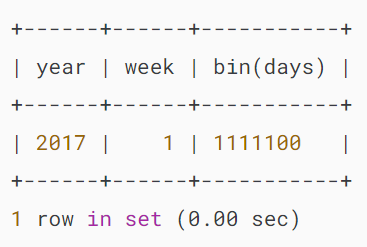
The following query retrieves data from the working\_calendar table:





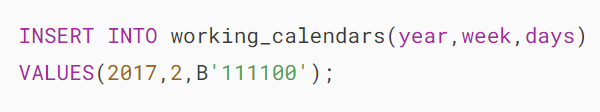
The output indicates that the bit value in the  days column is converted into an integer. To represent it as bit values, you use the [BIN](https://www.mysqltutorial.org/mysql-string-functions/mysql-bin-function/) function:

’

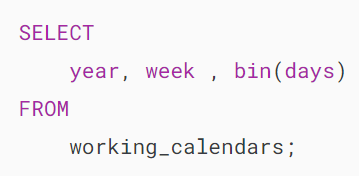


If you insert a value into a BIT(n) column that is less than n bits long, MySQL will pad zeros on the left of the bit value.

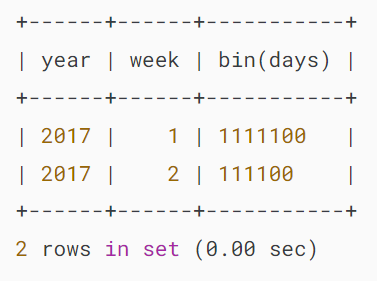
Suppose the first day of the second week is off, you can insert 01111100 into the  days column. However, the 111100 value will also work because MySQL will pad one zero to the left.



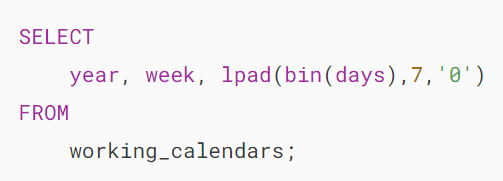
To view the data you use the same query as above:



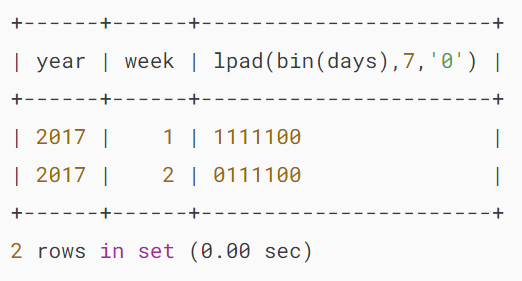
Output:



The output shows that MySQL removed the leading zeros before returning the result. To display it correctly, you can use the [LPAD()](https://www.mysqltutorial.org/mysql-string-functions/mysql-lpad/) function:



Output:



The output shows the expected format.

## **Summary**

* Use MySQL BIT data type to store BIT data in a table.

# **MySQL INT Data Type**

**Summary**: in this tutorial, you will learn about MySQL INT data type, and how to use it to store whole numbers in the databases.

## **Introduction to MySQL INT data type**

In MySQL, INT stands for the integer that represents the whole numbers. An integer can be written without a fractional component such as 1, 100, 4, -10, and it cannot be 1.2, 5/3, etc. An integer can be zero, positive, and negative.

MySQL supports all standard SQL integer types INTEGER or INT and SMALLINT. Additionally, MySQL provides TINYINT MEDIUMINT, and BIGINT as extensions to the SQL standard.

MySQL INT [data type](https://www.mysqltutorial.org/mysql-basics/mysql-data-types/) can be signed and unsigned. The following table illustrates the characteristics of each integer type including storage in bytes, minimum value, and maximum value.

| **Type** | **Storage** | **Minimum Value** | **Maximum Value** |
| --- | --- | --- | --- |
|  | **(Bytes)** | **(Signed/Unsigned)** | **(Signed/Unsigned)** |
| TINYINT | 1 | -128 | 127 |
|  |  | 0 | 255 |
| SMALLINT | 2 | -32768 | 32767 |
|  |  | 0 | 65535 |
| MEDIUMINT | 3 | -8388608 | 8388607 |
|  |  | 0 | 16777215 |
| INT | 4 | -2147483648 | 2147483647 |
|  |  | 0 | 4294967295 |
| BIGINT | 8 | -9223372036854775808 | 9223372036854775807 |
|  |  | 0 | 18446744073709551615 |

## **MySQL INT data type examples**

Let’s look at some examples of using the integer data type.

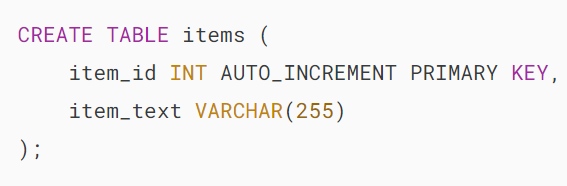
### **1) Using INT for a column example**

Because integer type represents exact numbers, you usually use it as the [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/)of a table. In addition, the INT column can have an [AUTO\_INCREMENT](https://www.mysqltutorial.org/mysql-basics/mysql-auto_increment/) attribute.

When you [insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) a NULL value or 0 into the INT AUTO\_INCREMENT column, the value of the column is set to the next [sequence](https://www.mysqltutorial.org/mysql-basics/mysql-auto_increment/) value. Notice that the sequence value starts with 1.

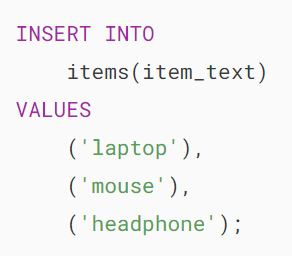
When you insert a value, which is not NULL or zero, into the AUTO\_INCREMENT column, the column accepts the value. In addition, the sequence is reset to the next value of the inserted value.

First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named items with an integer column as the [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/):

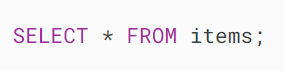


You can use either INT or INTEGER in the [CREATE TABLE](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) statement above because they are interchangeable. Whenever you insert a new row into the items table, the value of the item\_id column is increased by 1.

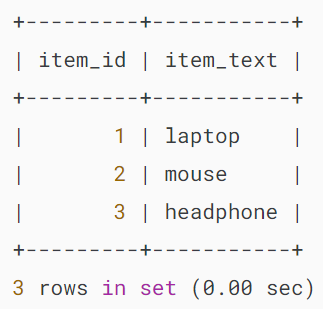
Next, the following [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) statement [inserts three rows](https://www.mysqltutorial.org/mysql-basics/mysql-insert-multiple-rows/) into the items table.



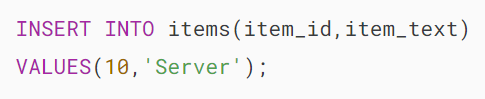
Then, query data from the items table using the following [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) statement:



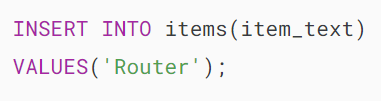
Output:



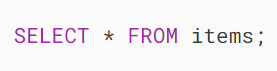
After that, insert a new row whose value of the item\_id column is specified explicitly.



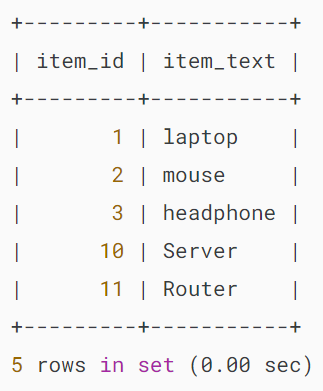
Since the current value of the item\_id column is 10, the sequence is reset to 11. If you insert a new row, the AUTO\_INCREMENT column will use 11 as the next value.



Finally, query the data of the items table again to see the result.

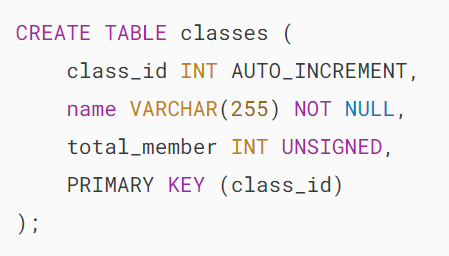


Output:

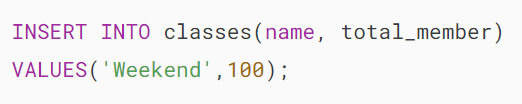


### **2) Using INT UNSIGNED example**

First, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) called classes that has the column total\_member with the unsigned integer data type:

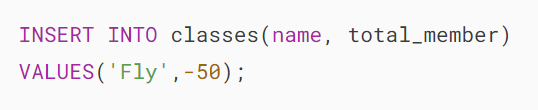


Second, [insert a new row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the classes table:

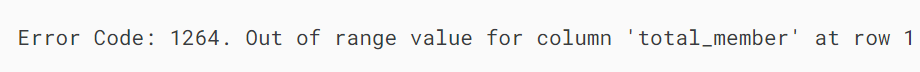


It worked as expected.

Third, attempt to insert a negative value into the total\_member column:



MySQL issued the following error:



Note that the display width has been deprecated. Additionally, the ZEROFILL attribute has also been deprecated, and the suggested alternatives are to use [LPAD](https://www.mysqltutorial.org/mysql-string-functions/mysql-lpad/) for zero-padding numbers or to store the formatted numbers in a [CHAR](https://www.mysqltutorial.org/mysql-basics/mysql-char-data-type/) column.

## **Summary**

* INT represents the integer type.
* MySQL offers various variants of the INT type including TINYINT, SMALLINT, MEDIUMINT, and BIGINT.

# **MySQL BOOLEAN Data Type**

**Summary**: in this tutorial, you will learn about MySQL BOOLEAN data type and how to use it to store Boolean values in the databases.

## **Introduction to MySQL BOOLEAN data type**

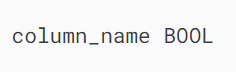
MySQL does not have a dedicated Boolean data type. Instead, MySQL uses TINYINT(1) to represent the BOOLEAN data type.

To make it more convenient when defining BOOLEAN column, MySQL offers BOOLEAN or BOOL as the synonym for TINYINT(1).

So instead of defining a BOOLEAN column like this:



You can use the BOOL or BOOLEAN keyword as follows:



In MySQL, the convention is that zero is considered false, while a non-zero value is considered true.

When working with Boolean literals, you can use the constants true and false case-insensitively, which is equivalent to 1 and 0 respectively. For example:



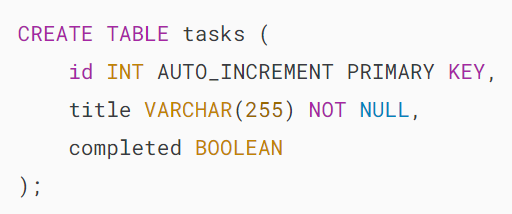
Output:

## 

## **MySQL BOOLEAN example**

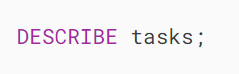
We’ll take an example of using the MySQL BOOLEAN data type.

First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) called tasks:

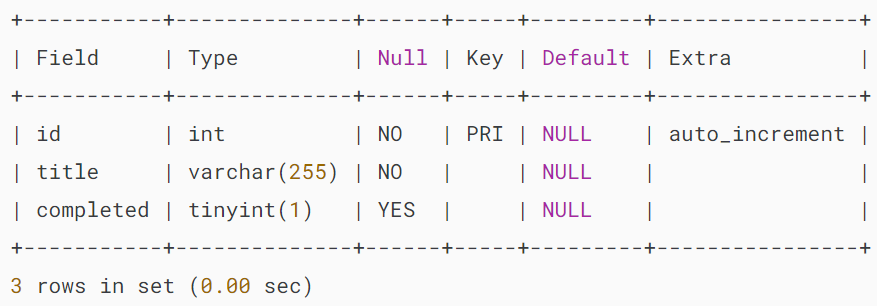


The tasks table has three columns id, title, and completed.

The completed is a BOOLEAN column. Since the BOOLEAN is a synonym for TINYINT(1), when you [describe the table structure](https://www.mysqltutorial.org/mysql-administration/mysql-show-columns/), MySQL shows the TINYINT(1) instead:



Output:

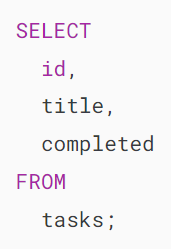


Second, insert two rows into the tasks table:

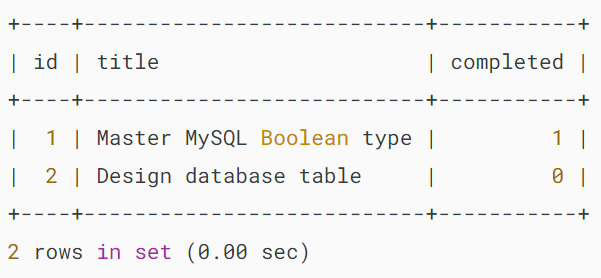


Before saving data into the Boolean column, MySQL converts it into 1 or 0.

Third, retrieve data from tasks table:

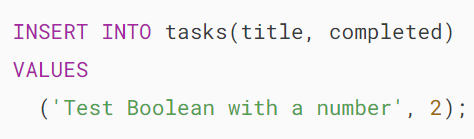


Output:



The output indicates that MySQL converted the true and false to 1 and 0 respectively.

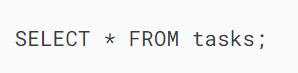
Fourth, because BOOLEAN is TINYINT(1), you can insert values other than 1 and 0 into the BOOLEAN column. For example:

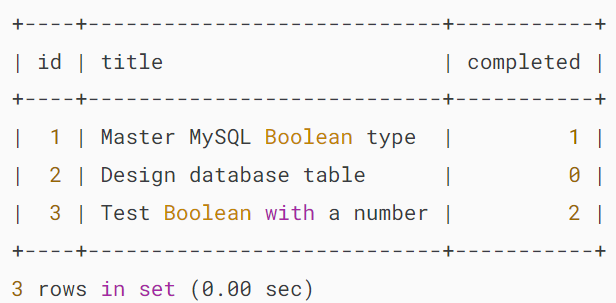


Output:

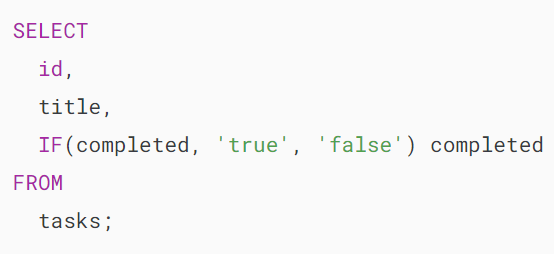


Fifth, query data from the tasks table:

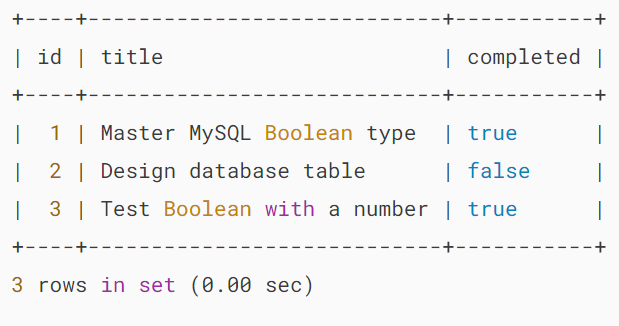
Output:



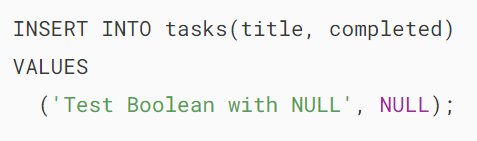
If you want to output the result as true and false, you can use the [IF](https://www.mysqltutorial.org/mysql-control-flow-functions/mysql-if-function/) function as follows:



Output:



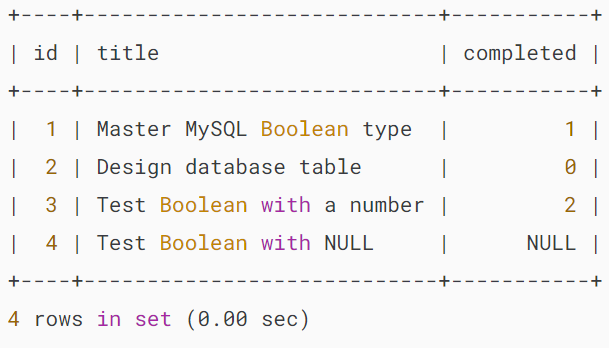
Sixth, insert NULL into the completed column:



Finally, retrieve data from the tasks table:

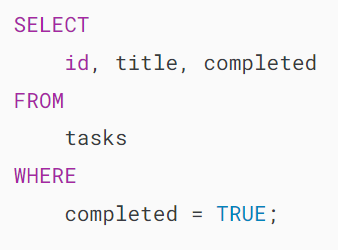


Output:

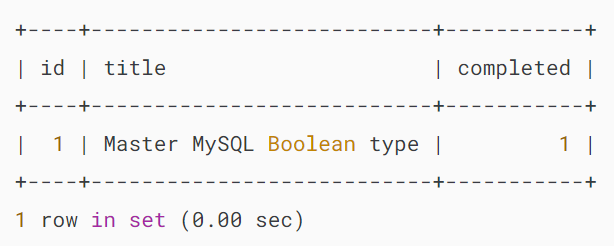


## **MySQL BOOLEAN operators**

To retrieve all completed tasks from the tasks table, you might come up with the following query:

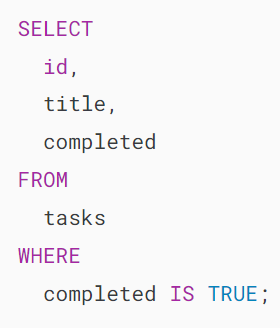


Output:

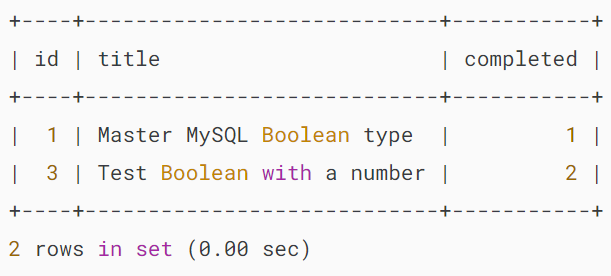


The query returned the task with completed value 1. It does not show the task with the completed value 2 because TRUE is 1, not 2.

To fix it, you can use the IS operator:

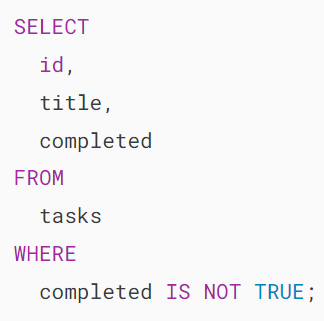


Output:

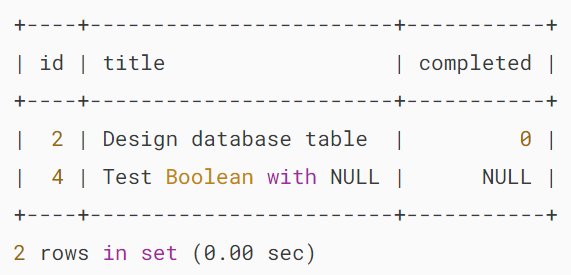


In this example, we used the IS operator to test a value against the TRUE value.

To get all the pending tasks, you can use IS FALSE or IS NOT TRUE as follows:



Output:



## **Summary**

* MySQL has no dedicated BOOLEAN data type. Instead, it uses TINYINT(1) to represent the BOOLEAN type.
* Use the BOOLEAN keyword to declare a column with the BOOLEAN type. The BOOLEAN and TINYINT(1) are synonyms.
* By convention, zero is false while non-zero is true.

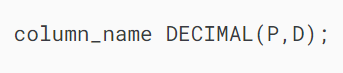
# **MySQL DECIMAL Data Type**

**Summary**: in this tutorial, you will learn about MySQL DECIMAL data type and how to use it to store exact numeric values in the databases.

## **Introduction to MySQL DECIMAL data type**

The MySQL DECIMAL data type allows you to store exact numeric values in the database. In practice, you often use the DECIMAL data type for columns that preserve exact precision e.g., monetary data in financial systems.

To define a column whose data type is DECIMAL, you use the following syntax:



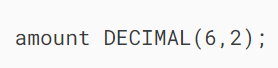
In this syntax:

* P is the precision that represents the number of significant digits. The range of P is 1 to 65.
* D is the scale that represents the number of digits after the decimal point. The range of D is 0 and 30. MySQL requires that D is less than or equal to (<=) P.

The DECIMAL(P,D) means that the column can store up to P digits with D decimals. The actual range of the decimal column depends on the precision and scale.

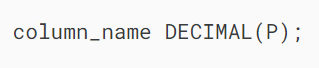
Besides the DECIMAL keyword, you can also use DEC, FIXED, or NUMERIC because they are synonyms for DECIMAL.

The following example defines the amount column with DECIMAL data type.

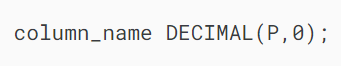


In this example, the amount column can store 6 digits with 2 decimal places; therefore, the range of the amount column is from 9999.99 to -9999.99.

MySQL allows you to use the following syntax:



This is equivalent to:



In this case, the column contains no fractional part or decimal point.

In addition, you can even use the following syntax:



The default value of P is 10 and D is 0, which is equivalent to the following:

## 

## **MySQL DECIMAL storage**

MySQL assigns the storage for integer and fractional parts separately. MySQL uses a binary format to store the DECIMAL values. It packs 9 digits into 4 bytes.

For each part, it takes 4 bytes to store each multiple of 9 digits. The storage required for leftover digits is illustrated in the following table:

| **Leftover Digits** | **Bytes** |
| --- | --- |
| 0 | 0 |
| 1–2 | 1 |
| 3–4 | 2 |
| 5–6 | 3 |
| 7–9 | 4 |

For example, DECIMAL(19,9) has 9 digits for the fractional part and 19-9 = 10 digits for the integer part. The fractional part requires 4 bytes. The integer part requires 4 bytes for the first 9 digits, for 1 leftover digit, it requires 1 more byte. In total, the DECIMAL(19,9) column requires 9 bytes.

## **MySQL DECIMAL data type and monetary data**

We often use the DECIMAL data type for monetary data such as prices, salary, account balances, and so on. If you design a database that handles the monetary data, the following syntax should be fine.

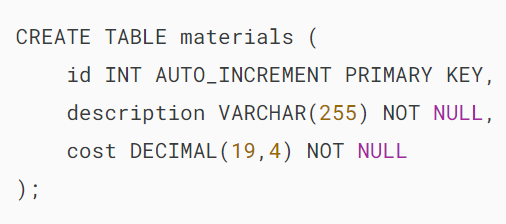


However, if you want to comply with [Generally Accepted Accounting Principles (GAAP)](https://en.wikipedia.org/wiki/Generally_Accepted_Accounting_Principles_(United_States)) rules, the monetary column must have at least 4 decimal places to make sure that the rounding value does not exceed $0.01. In this case, you should define the column with 4 decimal places as follows:

## 

## **MySQL DECIMAL data type example**

First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named materials:

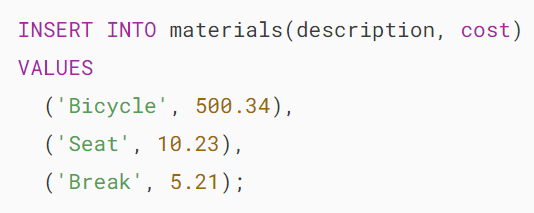


The materials table has three columns:

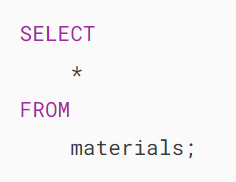
* id is the [auto-increment](https://www.mysqltutorial.org/mysql-basics/mysql-auto_increment/) [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) column with the INT data type.
* description represents the material’s description with the VARCHAR data type.
* cost represents the cost of the material, which has the DECIMAL(19,4).

The cost column can store up to 19 digits with 4 decimal places.

Second, [insert data](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the materials table.



Third, [query data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from the materials table.



Output:

# 

# **Summary**

* Use MySQL DECIMAL data type to store exact numeric values such as financial data in the database.
* Use column\_name DECIMAL (P, D) to define a column with the DECIMAL data type that has up to P digits and D decimal places.
* The DECIMAL(P) is equivalent to DECIMAL(P,0) and DECIMAL is equivalent to DECIMAL(P, 0).

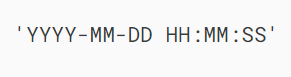
# **MySQL DATETIME Data Type**

**Summary**: in this tutorial, you will learn about MySQL DATETIME data type and how to use some handy functions for manipulating DATETIME effectively.

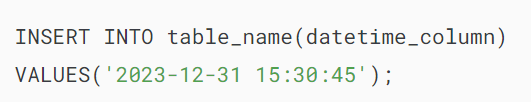
## **Introduction to MySQL DATETIME data type**

MySQL DATETIME data type allows you to store a value that contains both [date](https://www.mysqltutorial.org/mysql-basics/mysql-date/) and [time](https://www.mysqltutorial.org/mysql-basics/mysql-time/).

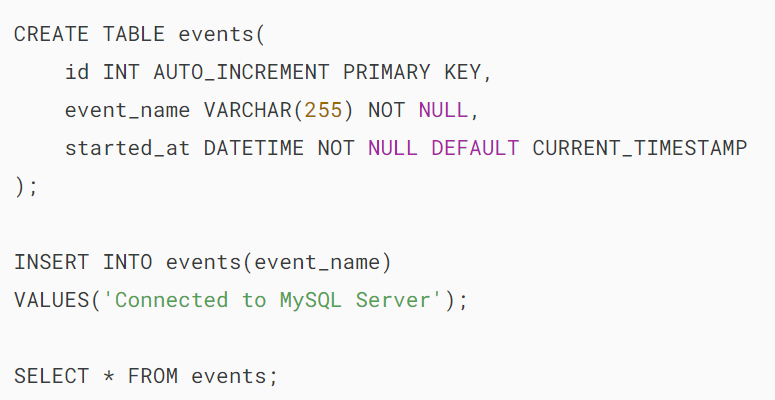
When you [query data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from a DATETIME column, MySQL displays the DATETIME value in the following format:



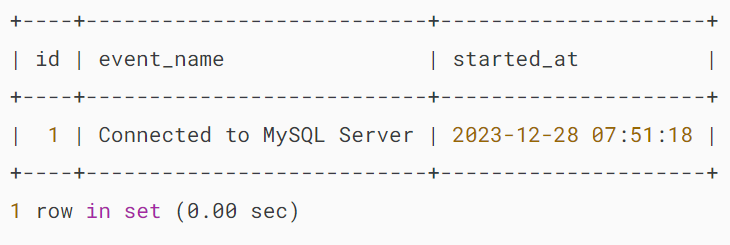
When you insert a value into a DATETIME column, you use the same format. For example:



To populate a column with the current date and time, you use the result of the CURRENT\_TIMESTAMP or NOW() function as the default value. For example:



Output:



By default, DATETIME values range from 1000-01-01 00:00:00 to 9999-12-31 23:59:59. MySQL uses 5 bytes to store a DATETIME value.

In addition, a DATETIME value can include a trailing fractional second up to microseconds with the format YYYY-MM-DD HH:MM:SS[.fraction] e.g., 2015-12-20 10:01:00.999999.

When including the fractional second precision, DATETIME values require more storage as illustrated in the following table:

| **Fractional Seconds Precision** | **Storage (Bytes)** |
| --- | --- |
| 0 | 0 |
| 1, 2 | 1 |
| 3, 4 | 2 |
| 5, 6 | 3 |

For example, 2015-12-20 10:01:00.999999 requires 8 bytes, 5  bytes for 2015-12-20 10:01:00 and 3 bytes for .999999 while 2015-12-20 10:01:00.9  requires only 6 bytes, 1 byte for the fractional second precision.

## **MySQL DATETIME vs.TIMESTAMP**

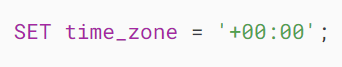
MySQL provides another temporal data type that is similar to the DATETIME called  [TIMESTAMP](https://www.mysqltutorial.org/mysql-basics/understanding-mysql-timestamp/).

The TIMESTAMP requires 4 bytes while DATETIME requires 5 bytes. Both TIMESTAMP and DATETIME require additional bytes for fractional seconds precision.

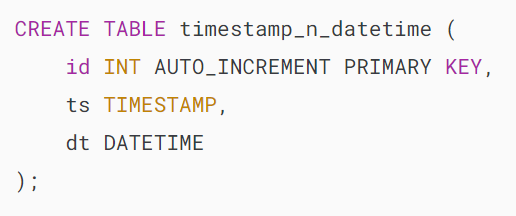
TIMESTAMP values range from 1970-01-01 00:00:01 UTC to 2038-01-19 03:14:07 UTC. If you want to store temporal values that are beyond 2038, you should use DATETIME instead of TIMESTAMP.

MySQL stores TIMESTAMP in UTC value. However, MySQL stores the DATETIME value as is without timezone. Let’s see the following example.

First, set the timezone of the current connection to +00:00.



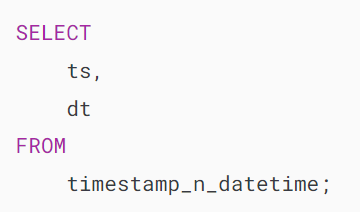
Next, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named timestamp\_n\_datetime that consists of two columns: ts and dt with TIMESTAMP and DATETIME types using the following statement.



Then, [insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) the current date and time into both ts and dt columns of the timestamp\_n\_datetime table,



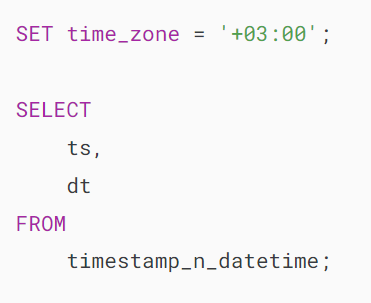
After that, [query data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from the timestamp\_n\_datetime table.

MySQL DATETIME vs TIMESTAMP

Both values in DATETIME and TIMESTAMP columns are the same.

Finally, set the connection’s time zone to +03:00 and query data from the timestamp\_n\_datetime table again.

SET time\_zone = '+03:00';



MySQL DATETIME vs TIMESTAMP timezone changes

The output indicates that the value in the TIMESTAMP column is different. This is because the TIMESTAMP column stores the date and time value in UTC when we change the time zone, the value of the TIMESTAMP column is adjusted according to the new time zone.

It means that if you use the TIMESTAMP data to store date and time values, you should take serious consideration when you move your database to a server located in a different time zone.

## **MySQL DATETIME functions**

The following statement sets the variable @dt to the current date and time using the [NOW()](https://www.mysqltutorial.org/mysql-date-functions/mysql-now-function/) function.



To query the value of the @dt variable, you use the following SELECT statement:



MySQL DATETIME - NOW function

### **MySQL DATE() function**

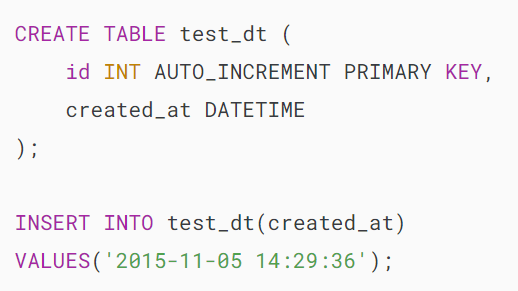
To extract the date portion from a DATETIME value, you use the DATE function as follows:



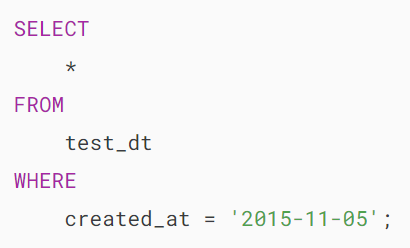
MySQL DATETIME - DATE function example

This function is very useful in case you want to query data based on a date but the data stored in the column is based on both date and time.

Let’s see the following example.

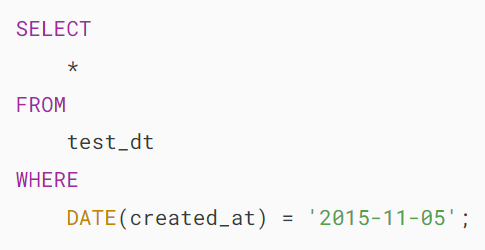


Suppose you want to know which row created on 2015-11-05, you use the following query:



It returns no rows.

This is because the created\_at column contains not only the date but also the time. To correct it, you use the DATE function as follows:



MySQL DATETIME - DATE function

It returns one row as expected. In case the table has many rows, MySQL has to perform a full table scan to locate the rows that match the condition.

### **MySQL TIME function**

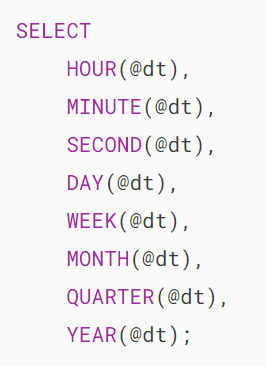
To extract the time portion from a DATETIME value, you use the TIME function as the following statement:



MySQL DATETIME - TIME function

### **MySQL YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE and SECOND functions**

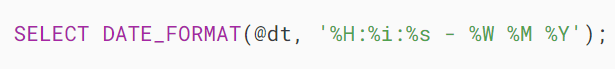
To get the year, quarter, month, week, day, hour, minute, and second from a DATETIME value, you use the functions as shown in the following statement:



MySQL DATETIME - datetime functions

### **MySQL DATE\_FORMAT function**

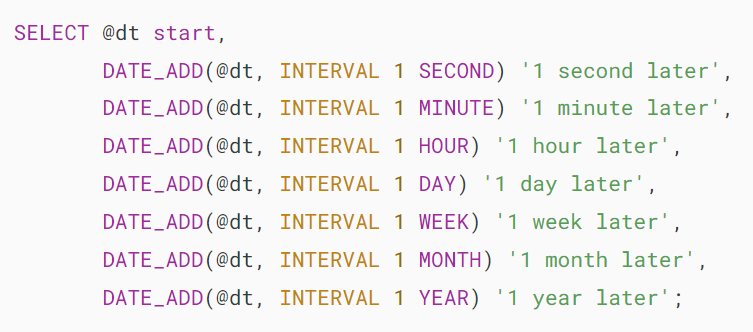
To format a DATETIME value, you use the [DATE\_FORMAT](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_format/) function. For example, the following statement formats a DATETIME value based on the %H:%i:%s - %W %M %Y format:



MySQL DATETIME - DATE_FORMAT function

### **MySQL DATE\_ADD function**

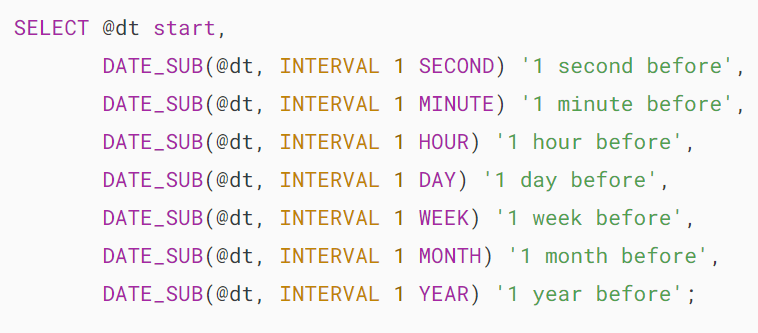
To add an [interval](https://www.mysqltutorial.org/mysql-basics/mysql-interval/)to a DATETIME value, you use [DATE\_ADD](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_add/) function as follows:



MySQL DATETIME - DATE_ADD function

### **MySQL DATE\_SUB function**

To subtract an interval from a DATETIME value, you use [DATE\_SUB](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_sub/) function as follows:



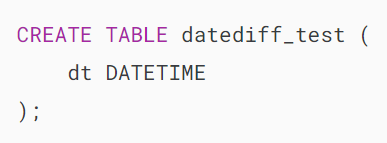
MySQL DATETIME - DATE_SUB function

### **MySQL DATE\_DIFF function**

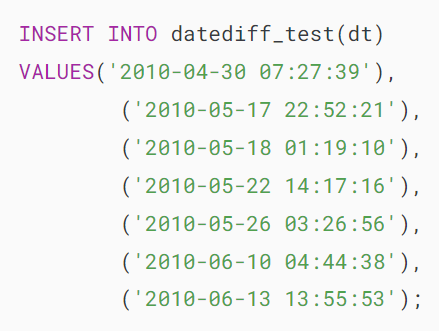
To calculate a difference in days between two DATETIME values, you use the [DATEDIFF](https://www.mysqltutorial.org/mysql-date-functions/mysql-datediff-function/) function. Notice that the DATEDIFF function only considers the date part of a DATETIME value in the calculation.

See the following example.

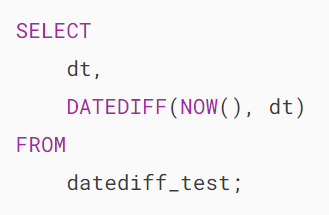
First, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named datediff\_test that has one column whose data type is DATETIME.

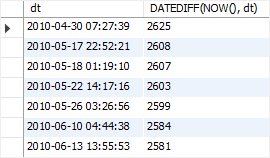


Second, insert some rows into the datediff\_test table.



Third, use the DATEDIFF function to compare the current date and time with the value in each row of the datediff\_test table.





# **MySQL TIMESTAMP Data Type**

**Summary**: in this tutorial, you will learn about MySQL TIMESTAMP data type and how to use it to define columns that store timestamp data.

## **Introduction to MySQL TIMESTAMP data type**

The MySQL TIMESTAMP is a temporal [data type](https://www.mysqltutorial.org/mysql-basics/mysql-data-types/) that holds the combination of [date](https://www.mysqltutorial.org/mysql-basics/mysql-date/) and [time](https://www.mysqltutorial.org/mysql-basics/mysql-time/). The [format](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_format/)of a TIMESTAMP is YYYY-MM-DD HH:MM:SS which is fixed at 19 characters.

The TIMESTAMP value has a range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

When you [insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/)a TIMESTAMP value into a table, MySQL converts it from your connection’s time zone to UTC for storing.

When you [query](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) a TIMESTAMP value, MySQL converts the UTC value back to your connection’s time zone. This conversion does not occur for other temporal data types, such as [DATETIME](https://www.mysqltutorial.org/mysql-basics/mysql-datetime/).

By default, the connection time zone is the MySQL Server’s time zone. You also have the option to use a different time zone when connecting to the MySQL Server.

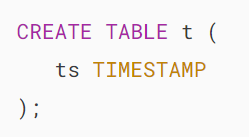
When you retrieve a TIMESTAMP value that was inserted by a client in a different time zone, you will receive a value different from the one stored in the database.

However, as long as you don’t change the time zone, you can retrieve the originally stored TIMESTAMP value.

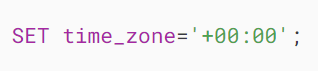
## **MySQL TIMESTAMP time zone example**

Let’s take an example to see how MySQL handles TIMESTAMP values.

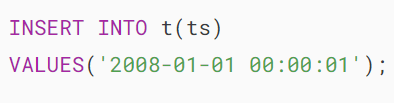
First, [created a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) called t that has a TIMESTAMP column: t1;



Second, set the session’s time zone to '+00:00' UTC by using the SET time\_zone statement.



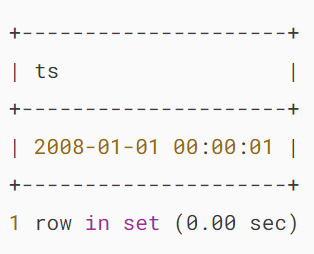
Third, insert a TIMESTAMP value into the t table.



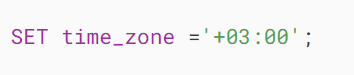
Fourth, select the TIMESTAMP value from the t table.



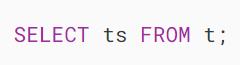
Output:



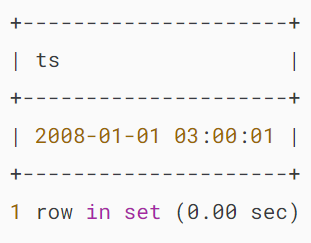
Fifth, set the session’s time zone to a different time zone to observe what value we receive from the database server:



Finally, query data from the table:



Output:

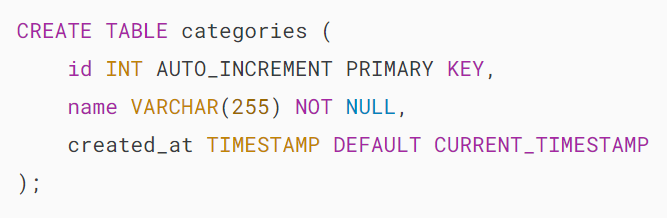


As you can see, we received a time value adjusted to the new time zone, which is different from the original.

## **Automatic initialization and updating for TIMESTAMP columns**

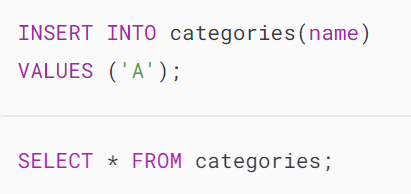
Consider the following example.

First, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named  categories:



In the categories table, the created\_at column is a TIMESTAMP column whose default value is set to CURRENT\_TIMESTAMP.

Second, [insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) a new row into the categories table without specifying the value for the created\_at column:

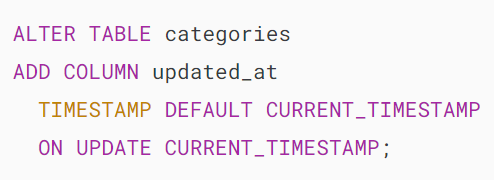


MySQL TIMESTAMP - Automatic Initialization

The output indicates that MySQL used the timestamp at the time of insertion as a default value for the created\_at column.

So a TIMESTAMP column can be automatically initialized to the current timestamp for inserted rows that specify no value for the column. This feature is called **automatic initialization**.

Third, [add a new column](https://www.mysqltutorial.org/mysql-basics/mysql-add-column/) named updated\_at to the categories table:



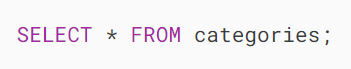
The default value of the updated\_at column is CURRENT\_TIMESTAMP.

And, we have a new clause ON UPDATE CURRENT\_TIMESTAMP that follows the DEFAULT CURRENT\_TIMESTAMP clause. Let’s see its effect.

Fourth, [insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) a new row into the categories table.



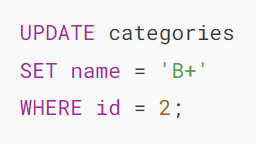
Fifth, query data from the categories table:



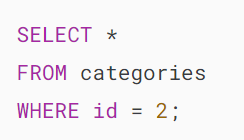
MySQL TIMESTAMP - Automatic Updating

The default value of the column created\_at is the timestamp when the row was inserted.

Sixth, update the value in the column name of the row id 2:



Seventh, query data from the categories table to check the update:



MySQL TIMESTAMP - Automatic Updating feature

Notice that the value in the updated\_at column changed to the timestamp at the time the row was updated.

The ability of a TIMESTAMP column to be automatically updated to the current timestamp when the value in any other column in the row changes from its current value is called **automatic updating**.

The column updated\_at is referred to as an auto-updated column.

Note that if you execute the UPDATE statement to update the same value for the name column, the updated\_at column will not be updated.

The value in the updated\_at remains unchanged.

MySQL TIMESTAMP - Automatic Updating feature

For more information on automatic initialization and updating, please check out the [time initialization](https://dev.mysql.com/doc/refman/5.7/en/timestamp-initialization.html).

As of MySQL 5.6.5, the [DATETIME](https://www.mysqltutorial.org/mysql-basics/mysql-datetime/) columns also have automatic initialization and updating features. In addition, the DEFAULT\_CURRENT\_TIMESTAMP and ON UPDATE CURRENT TIMESTAMP can be applied to multiple columns.

## **Summary**

* Use the MySQL TIMESTAMP data type to represent date and time values.
* Set the DEFAULT CURRENT\_TIMESTAMP attribute for a TIMESTAMP column to automatically initialize the column with the current timestamp when a new row is inserted.
* Set the ON UPDATE CURRENT\_TIMESTAMP attribute to update the timestamp whenever the row is modified.
* MySQL stores the TIMESTAMP values in UTC format but converts them to the current session timezone when displayed.

# **MySQL DATE Data Type**

**Summary**: in this tutorial, we will introduce you to the MySQL DATE data type and show you some useful date functions to handle the date data effectively.

## **Introduction to MySQL DATE data type**

MySQL DATE is one of the five temporal [data types](https://www.mysqltutorial.org/mysql-basics/mysql-data-types/) used for managing date values. MySQL uses yyyy-mm-dd format for storing a date value. This format is fixed and it is not possible to change it.

For example, you may prefer to use mm-dd-yyyy format but you can’t. Instead, you follow the standard date format and use the [DATE\_FORMAT](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_format/) function to format the date the way you want.

MySQL uses three bytes to store a DATE value. The DATE values range from 1000-01-01 to 9999-12-31.

If you want to store a date value that is out of this range, you need to use a non-temporal data type like an [integer](https://www.mysqltutorial.org/mysql-basics/mysql-int/) e.g., three columns, and each column for the year, month, and day.

Also, you need to create [stored functions](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-stored-function/) to simulate the built-in [date functions](https://www.mysqltutorial.org/mysql-date-functions/) provided by MySQL, which is not recommended.

## **MySQL Date values with two-digit years**

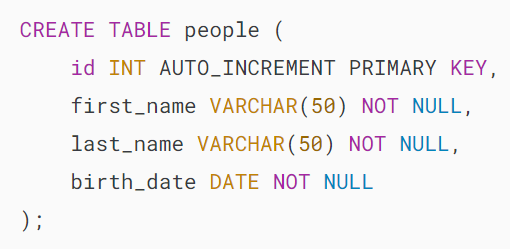
MySQL stores the year of the date value using four digits. In case you use two-digit year values, MySQL still accepts them with the following rules:

* Year values in the range 00-69 are converted to 2000-2069.
* Year values in the range 70-99 are converted to 1970 – 1999.

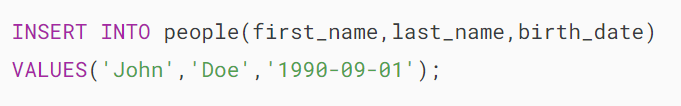
However, a date value with two digits is ambiguous therefore you should avoid using it.

Let’s take a look at the following example.

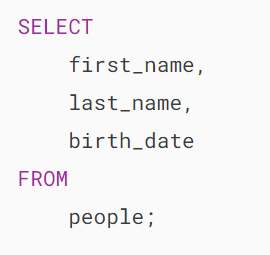
First, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named people with birth date column with DATE data type.



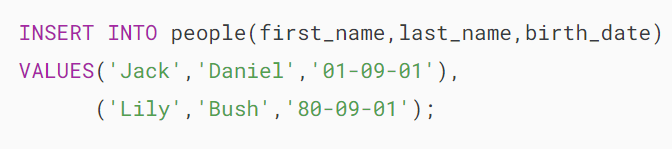
Next, [insert a row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the people table.



Then, [query the data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from the people table.

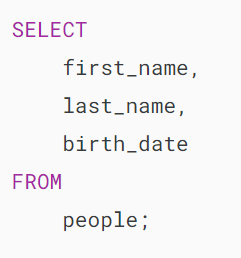
MySQL DATE Data Type Example

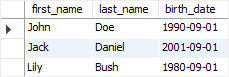
After that, use the two-digit year format to insert data into the people table.



In the first row, we used 01 (range 00-69) as the year, so MySQL converted it to 2001. In the second row, we used 80 (range 70-99) as the year, MySQL converted it to 1980.

Finally, retrieve data from the people table to check whether data was converted based on the conversion rules.

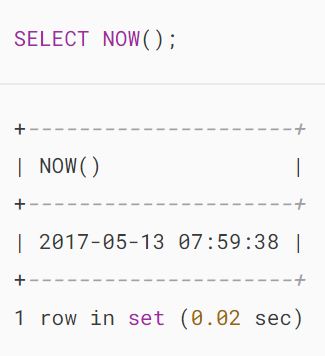




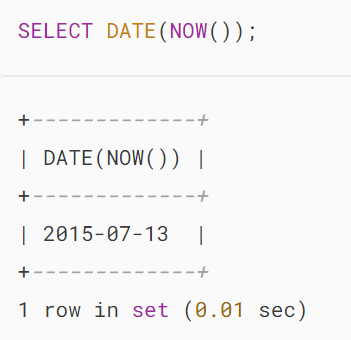
## **MySQL Date Functions**

MySQL provides many useful [date functions](https://www.mysqltutorial.org/mysql-date-functions/) that allow you to manipulate dates effectively.

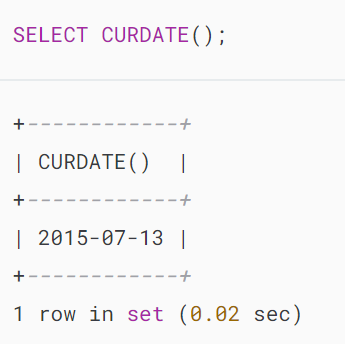
To get the current date and time, you use [NOW()](https://www.mysqltutorial.org/mysql-date-functions/mysql-now-function/) function.



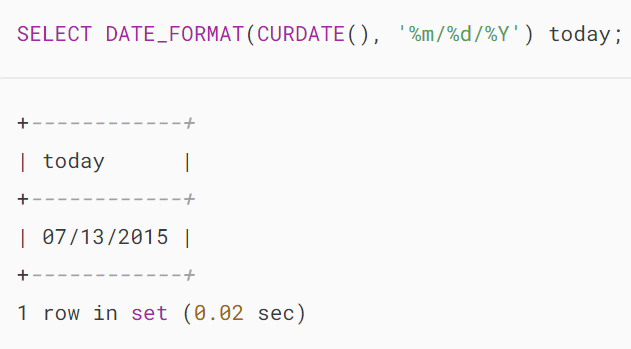
To get only the date part of a [DATETIME](https://www.mysqltutorial.org/mysql-basics/mysql-datetime/) value, you use the DATE() function.



To get the current system date, you use  [CURDATE()](https://www.mysqltutorial.org/mysql-date-functions/mysql-curdate/) function as follows:



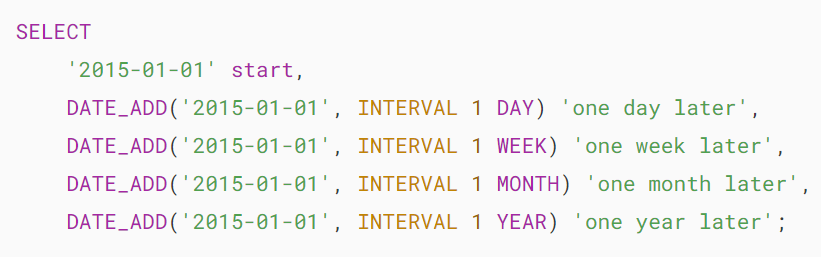
To format a date value, you use  [DATE\_FORMAT](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_format/) function. The following statement formats the date as mm/dd/yyyy using the date format pattern %m/%d/%Y :



To calculate the number of days between two date values, you use the [DATEDIFF](https://www.mysqltutorial.org/mysql-date-functions/mysql-datediff-function/) function as follows:

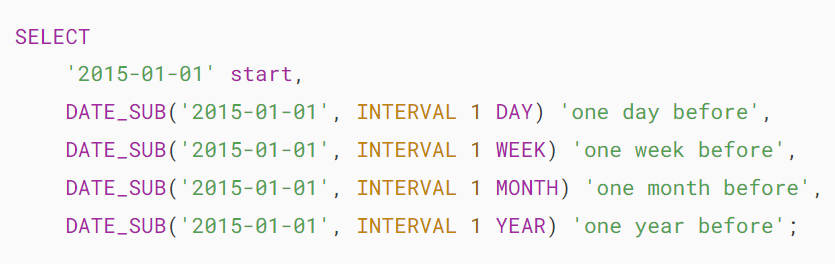


To add a number of days, weeks, months, years, etc., to a date value, you use the [DATE\_ADD](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_add/) function:

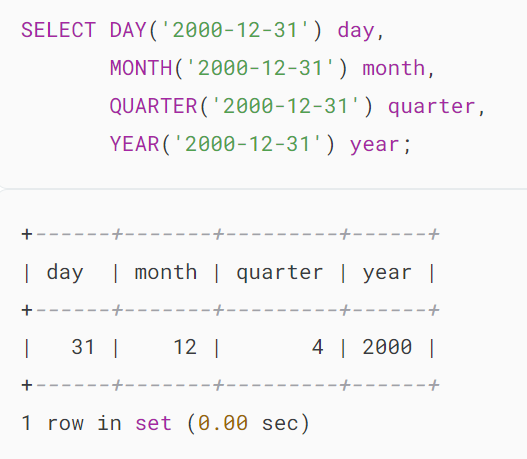


MySQL DATE - DATE_ADD example

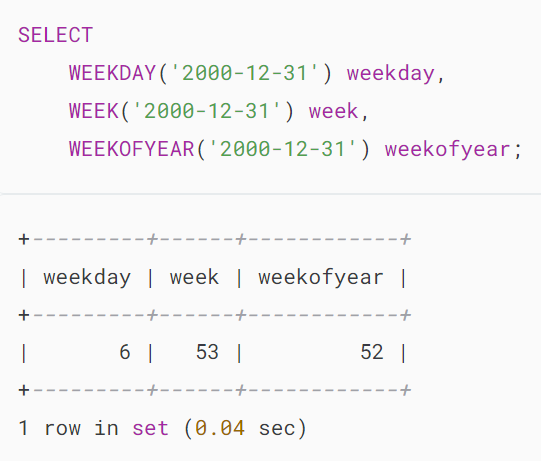
Similarly, you can subtract an [interval](https://www.mysqltutorial.org/mysql-basics/mysql-interval/) from a date using the [DATE\_SUB](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_sub/) function:

MySQL DATE - DATE_SUB example

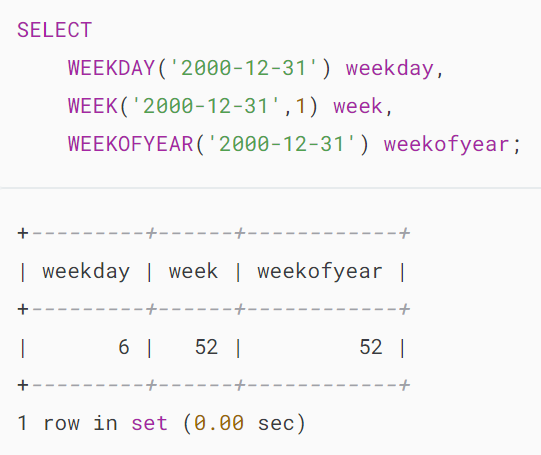
If you want to get the day, month, quarter, and year of a date value, you can use the corresponding function [DAY](https://www.mysqltutorial.org/mysql-date-functions/mysql-day/), [MONTH](https://www.mysqltutorial.org/mysql-month/), QUARTER, and [YEAR](https://www.mysqltutorial.org/mysql-year/) as follows:



To get the week’s information, you use the week-related functions. For example, [WEEK](https://www.mysqltutorial.org/mysql-date-functions/mysql-week/) function returns the week number, WEEKDAY function returns the weekday index, and WEEKOFYEAR function returns the calendar week.



The week function returns the week number with the zero-based index if you don’t pass the second argument or if you pass 0. If you pass 1, it will return the week number with 1-indexed.



# **MySQL TIME Data Type**

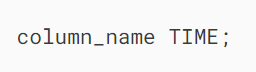
**Summary**: in this tutorial, you will learn about the MySQL TIME data type and how to use temporal functions to manipulate time data effectively.

## **Introduction to MySQL TIME data type**

MySQL uses the 'HH:MM:SS' format for querying and displaying a time value that represents a time of day, which is within 24 hours.

To represent a time [interval](https://www.mysqltutorial.org/mysql-basics/mysql-interval/) between two events, MySQL uses the 'HHH:MM:SS' format, which is larger than 24 hours.

To define a TIMEcolumn, you use the following syntax:

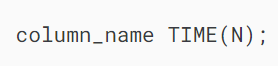


For example, the following snippet defines a column named start\_at with TIME data type.



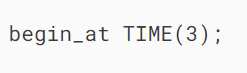
A TIME value ranges from -838:59:59 to 838:59:59. In addition, a TIME value can have a fractional seconds part that is up to microseconds precision (6 digits).

To define a column whose data type is TIME with a fractional-second precision part, you use the following syntax:



In this syntax, N is an [integer](https://www.mysqltutorial.org/mysql-basics/mysql-int/) that represents the fractional part, which is up to 6 digits.

The following defines a column with TIME data type including 3 digits of fractional seconds.



A TIME value takes 3 bytes for storage. If a TIME value includes fractional second precision, it will take additional bytes based on the number of digits of the fractional second precision.

The following table illustrates the storage required for fractional second precision.

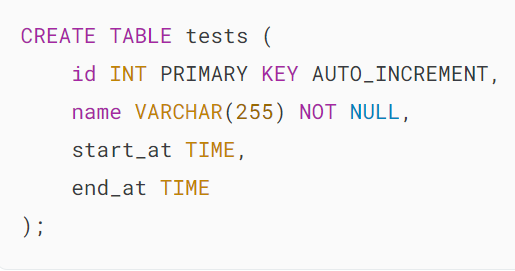
|  |  |
| --- | --- |
| **Fractional Second Precision** | **Storage (BYTES)** |
| 0 | 0 |
| 1, 2 | 1 |
| 3, 4 | 2 |
| 5, 6 | 3 |

For example, TIME and TIME(0) takes 3 bytes. TIME(1) and TIME(2) takes 4 bytes (3  + 1); TIME(3) and TIME(6) take 5 and 6 bytes.

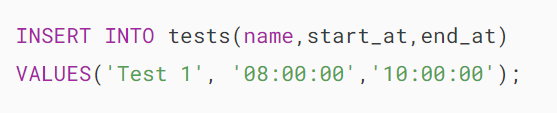
## **MySQL TIME data type example**

Let’s take a look at an example of using the TIME data type for columns in a table.

First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named tests that consists of four columns: id, name, start\_at, and end\_at. The data types of the start\_at and end\_at columns are TIME:



Second, [insert a row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the tests table.



Third, [query data](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) from the tests table.



Notice that we use 'HH:MM:SS' as the literal time value in the [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) statement. Let’s examine all the valid time literals that MySQL can recognize.

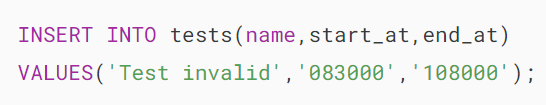
## **MySQL TIME literals**

MySQL recognizes various time formats besides the 'HH:MM:SS' format that we mentioned earlier.

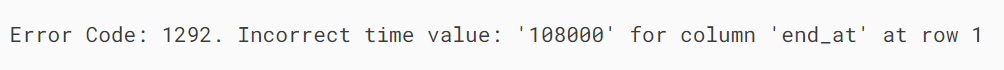
MySQL allows you to use the 'HHMMSS' format without delimiter ( : ) to represent time value. For example, '08:30:00' and '10:15:00' can be rewritten as '083000' and '101500'.



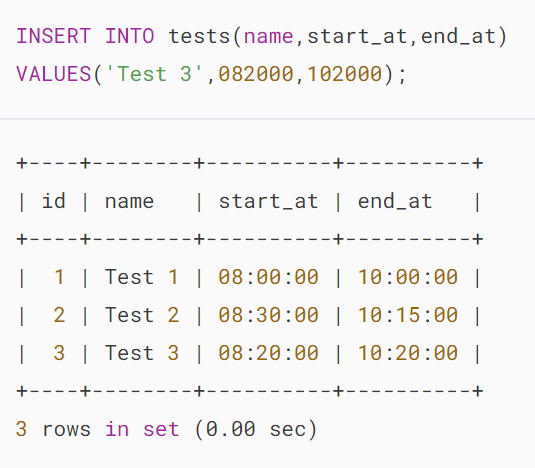
However, 108000 is not a valid time value because 80 does not represent the correct minute. In this case, MySQL will raise an error if you try to insert an invalid time value into a table.



MySQL issued the following error message after executing the above statement.



In addition to the string format, MySQL accepts the HHMMSS as a number that represents a time value. You can also use SS, MMSS. For example, instead of using '082000', you can use 082000 as follows:



For the time interval, you can use the 'D HH:MM:SS' format where D represents days with a range from 0 to 34. A more flexible syntax is 'HH:MM', 'D HH:MM', 'D HH', or 'SS'.

If you use the delimiter :, you can use one digit to represent hours, minutes, or seconds. For example, 9:5:0 can be used instead of '09:05:00'.

## 

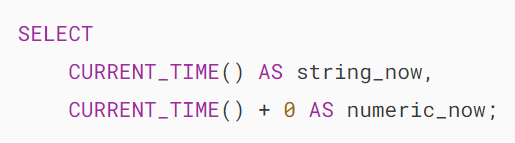
## **Useful MySQL TIME functions**

MySQL provides several useful temporal functions for manipulating TIME data.

### **1) Getting the current time**

To get the current time of the database server, you use the CURRENT\_TIME function. The CURRENT\_TIME function returns the current time value as a string ( 'HH:MM:SS') or a numeric value ( HHMMSS) depending on the context where the function is used.

The following statements illustrate the CURRENT\_TIME function in both string and numeric contexts:

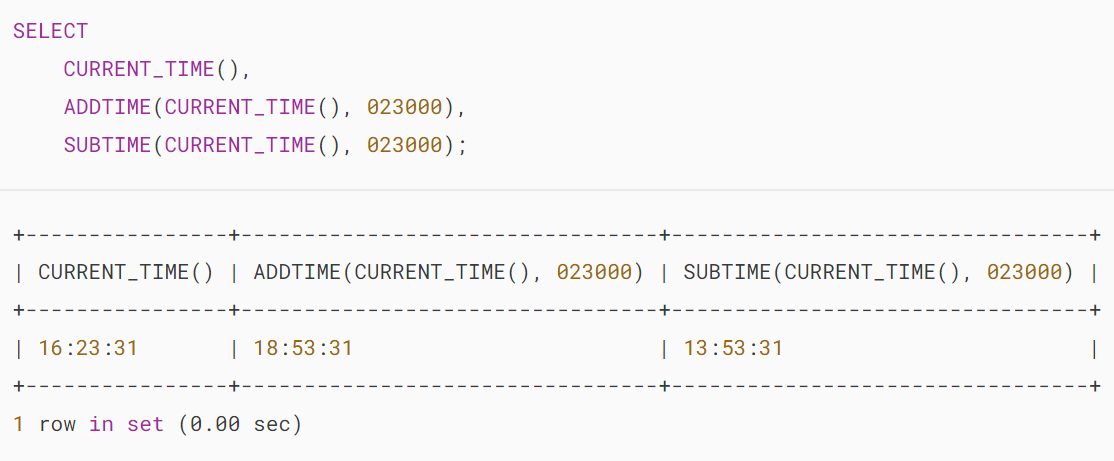


MySQL CURRENT_TIME function

### **2) Adding and Subtracting time from a TIME value**

To add a TIME value to another TIME value, you use the ADDTIME function. To subtract a TIME value from another TIME value, you use  the SUBTIME function.

The following statement adds and subtracts 2 hours 30 minutes to and from the current time.



In addition, you can use the TIMEDIFF() function to get a difference between two TIME values.

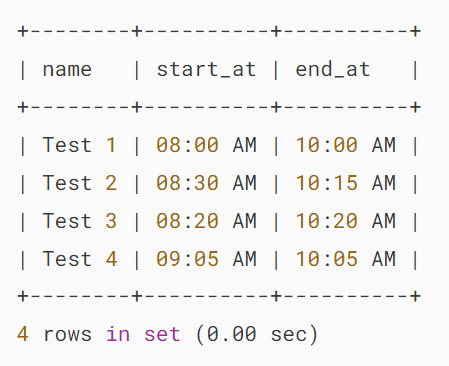
### **.**

### **3) Formatting MySQL TIME values**

Although MySQL uses 'HH:MM:SS' when retrieving and displaying a TIME value, you can display the TIME value in your preferred way using the TIME\_FORMAT function.

The TIME\_FORMAT function is like the [DATE\_FORMAT](https://www.mysqltutorial.org/mysql-date-functions/mysql-date_format/) function except that the TIME\_FORMAT function is used to format a TIME value only.

See the following example.

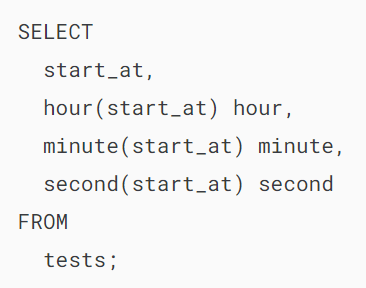


In the time format string above:

* %h means two-digit hours from 0 to 12.
* %i means two-digit minutes from 0 to 60.
* %p means AM or PM.

### **4) Extracting hour, minute, and second from a TIME value**

To extract the hour, minute, and second from a TIME value, you use HOUR, MINUTE, and SECOND functions as follows:

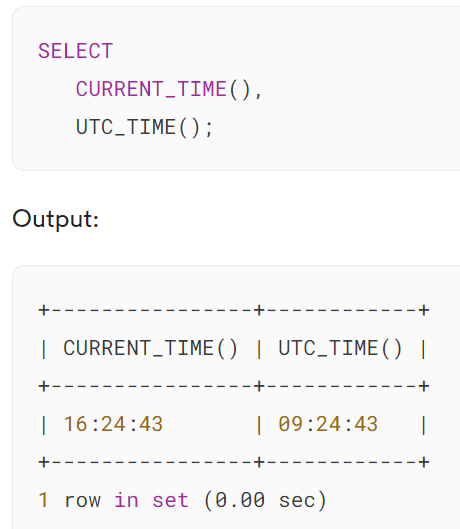


Output:

### 

### **5) Getting UTC time value**

To get the UTC time, you use UTC\_TIME function as follows:



# **MySQL CHAR Data Type**

**Summary**: in this tutorial, you will learn about MySQL CHAR data type and how to apply it in your database table design.

## **Introduction to MySQL CHAR data type**

The CHAR data type is a fixed-length character type in MySQL. You often declare the CHAR type with a length that specifies the maximum number of characters that you want to store. For example, CHAR(20) can hold up to 20 characters.

If the data that you want to store is a fixed size, then you should use the CHAR data type. You’ll get a better performance in comparison with VARCHAR this case.

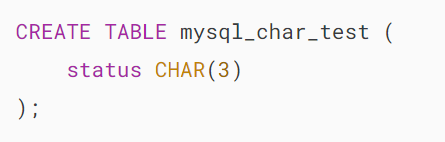
The length of the CHAR data type can be any value from 0 to 255. When you store a CHAR value, MySQL pads its value with spaces to the length that you declared.

When you query the CHAR value, MySQL removes the trailing spaces.

Note that MySQL will not remove the trailing spaces if you enable the [PAD\_CHAR\_TO\_FULL\_LENGTH](http://dev.mysql.com/doc/refman/5.7/en/sql-mode.html#sqlmode_pad_char_to_full_length) SQL mode.

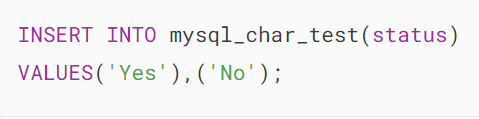
Consider the following example.

First, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) with a CHAR column.

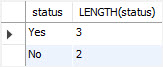
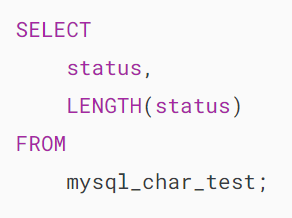


The data type of the  status column is CHAR . It can hold up to 3 characters.

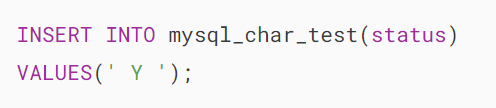
Second, [insert two rows](https://www.mysqltutorial.org/mysql-basics/mysql-insert-multiple-rows/) into the mysql\_char\_test table.



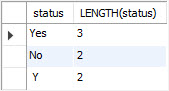
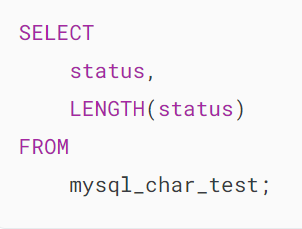
Third, use the [length](https://www.mysqltutorial.org/mysql-string-functions/mysql-string-length/) function to get the length of each CHAR value.



Fourth, insert a CHAR value with the leading and trailing spaces.



Finally, query the inserted values, and you will see that MySQL removes the trailing spaces.



## **Comparing MySQL CHAR values**

When storing or comparing the CHAR values, MySQL uses the [character set](https://www.mysqltutorial.org/mysql-basics/mysql-character-set/) [collation](https://www.mysqltutorial.org/mysql-basics/mysql-collation/) assigned to the column.

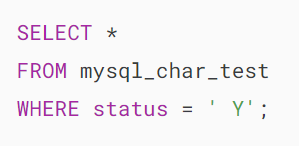
MySQL does not consider trailing spaces when comparing CHAR values using the comparison operator such as =, <>, >, <, etc.

Notice that the [LIKE](https://www.mysqltutorial.org/mysql-basics/mysql-like/) operator does consider the trailing spaces when you do pattern matching with CHAR values.

In the previous example, we stored the value Y with both leading and trailing spaces. However, when we execute the following query:

’

MySQL returns no row because it does not consider the trailing space. To match with the ‘ Y ‘, we need to remove the trailing space as follows:



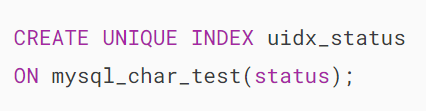
MySQL CHAR query

## **MySQL CHAR and UNIQUE index**

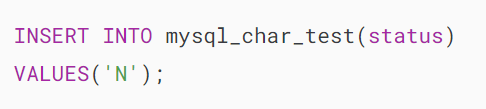
If the CHAR column has a [UNIQUE](https://www.mysqltutorial.org/mysql-unique/) index and you insert a value that is different from an existing value in a number of trailing spaces, MySQL will reject the changes because of duplicate-key error.

See the following example.

First, [create a unique index](https://www.mysqltutorial.org/mysql-unique/) for the status column of the mysql\_char\_test table.



Second, [insert a new row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the mysql\_char\_test table.



Third, insert the following value will cause a duplicate key error.



## **Summary**

* MySQL CHAR data type is a fixed-length character type.
* Use the CHAR data type to store fixed-length character data.

# **MySQL VARCHAR Data Type**

**Summary**: this tutorial introduces you to the MySQL VARCHAR data type and discusses some important features of VARCHAR.

## **Introduction to MySQL VARCHAR data type**

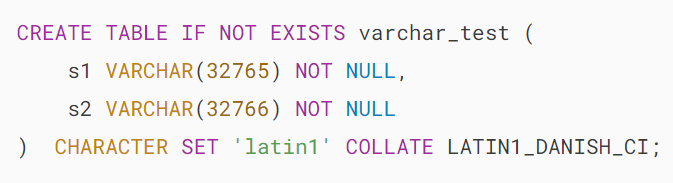
MySQL VARCHAR is the variable-length string whose length can be up to 65,535. MySQL stores a VARCHAR value as a 1-byte or 2-byte length prefix plus actual data.

The length prefix specifies the number of bytes in the value. If a column requires less than 255 bytes, the length prefix is 1 byte. In case the column requires more than 255 bytes, the length prefix is two length bytes.

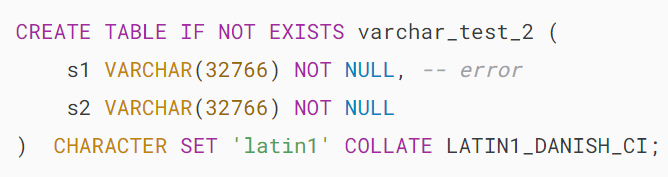
The maximum length, however, is subject to the maximum row size (65,535 bytes) and the [character set](https://www.mysqltutorial.org/mysql-basics/mysql-character-set/) used. It means that the total length of all columns should be less than 65,535 bytes.

Let’s take a look at an example.

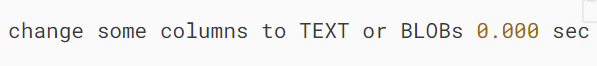
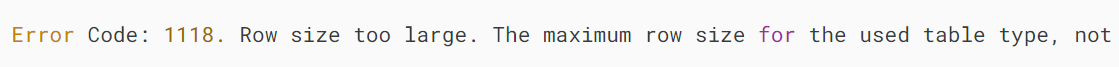
We will [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) that has two columns s1 and s2 with the length of 32765(+2 for length prefix) and 32766 (+2).Note that 32765+2+32766+2=65535, which is the maximum row size.



The statement created the table successfully. However, if we increase the length of the s1 column by 1.

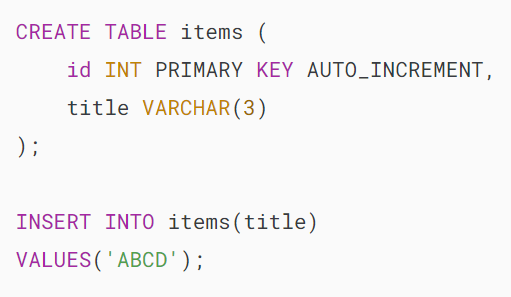


MySQL will issue the error message:



The output indicates that the row size is too large and the statement fails.

If you [insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) a string whose length is greater than the length of a VARCHAR column, MySQL will issue an error and skip inserting data. For example:

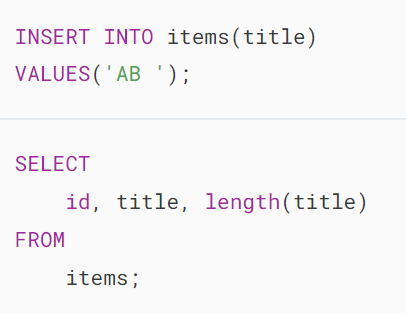


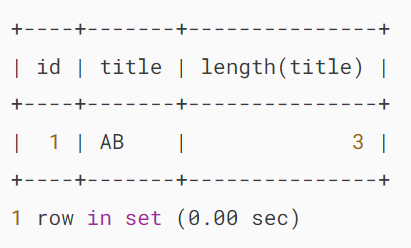
In this example, MySQL will issue the following error message:

## 

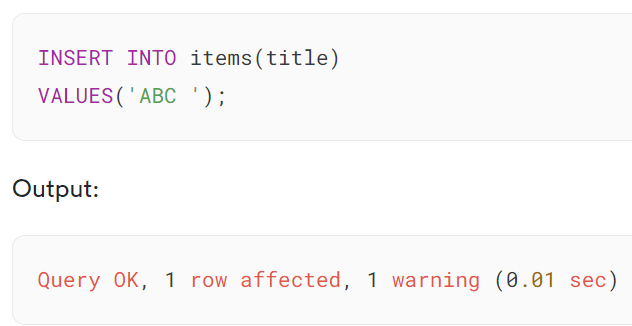
## **MySQL VARCHAR and spaces**

MySQL does not implicitly pad space when storing the VARCHAR values. Additionally, MySQL retains the trailing spaces when inserting or retrieving VARCHAR values. For example:



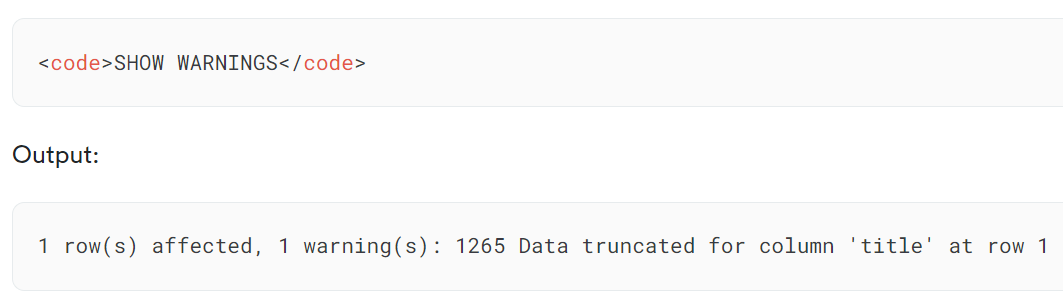


However, MySQL will truncate the trailing spaces when inserting a VARCHAR value that contains trailing spaces which cause the column length exceeded. In addition, MySQL issues a warning. Let’s see the following example:

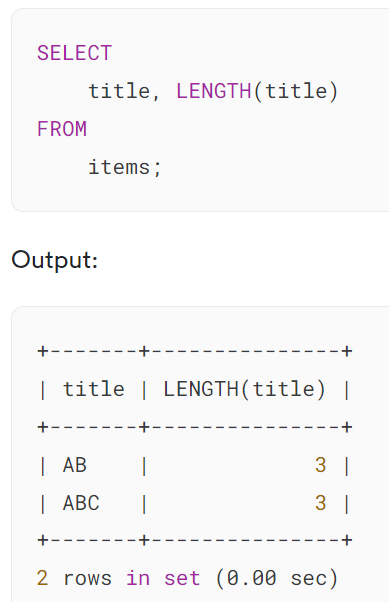


This statement inserts a string whose length is 4 into the title column. MySQL inserts the string but truncates the trailing space before inserting the value.

To show the warning message, you can use the SHOW WARNINGS statement:



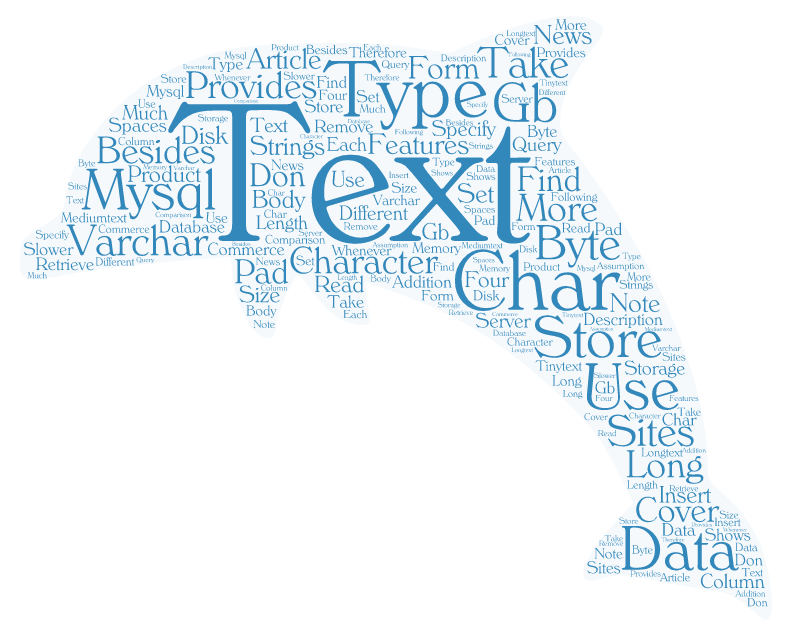
The following query verifies the action:



# **MySQL TEXT Data Type**

**Summary**: in this tutorial, you will learn how to use MySQL TEXT for storing text data in the database table.

## **Introduction to MySQL TEXT data type**

[](https://www.mysqltutorial.org/wp-content/uploads/2017/02/MySQL-TEXT-data-type.png)

Besides [CHAR](https://www.mysqltutorial.org/mysql-basics/mysql-char-data-type/) and [VARCHAR](https://www.mysqltutorial.org/mysql-basics/mysql-varchar/) character types, MySQL supports the TEXT type that provides more features.

The TEXT is useful for storing long-form text strings that can take from 1 byte to 4GB. In practice, you often use the TEXT data type for storing articles in news sites, and product descriptions in e-commerce sites.

Unlike CHAR and VARCHAR type, you don’t have to specify a storage length when you use a TEXT type for a column.

Also, MySQL does not remove or pad spaces when retrieving or inserting text data like CHAR and VARCHAR.

Note that the TEXT data is not stored in the database server’s memory. Therefore, when you query TEXT data, MySQL has to read from it from the disk, which is much slower in comparison with CHAR and VARCHAR.

MySQL provides four TEXT types:

* TINYTEXT
* TEXT
* MEDIUMTEXT
* LONGTEXT

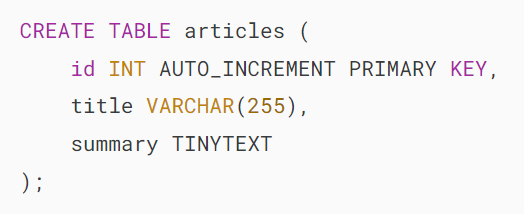
The following shows the size of each TEXT type with the assumption that you are using a character set that takes one byte to store a character

## **TINYTEXT – 255 Bytes (255 characters)**

The maximum number of characters that TINYTEXT can store is 255 ( 2^8 = 256, 1 byte overhead).

In practice, you use TINYTEXT for the column that requires less than 255 characters, has inconsistent length, and does not require sorting. For example, you can use the TINYTEXT to store the excerpt of a blog post.

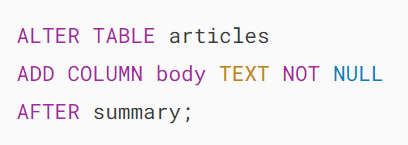
The following example [creates a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) called articles that has a summary column with the data type is TINYTEXT:



## **TEXT – 64KB (65,535 characters)**

The TEXT data type can hold up to 64 KB which is equivalent to 65535 (2^16 – 1) characters. The TEXT datatype also requires 2 bytes overhead.

The following example adds the column body with the TEXT type to the articles table using the [ALTER TABLE](https://www.mysqltutorial.org/mysql-basics/mysql-alter-table/) statement:



## **MEDIUMTEXT – 16MB (16,777,215 characters)**

The MEDIUMTEXT can hold up to 16MB text data which is equivalent to 16,777,215 characters. It requires 3 bytes overhead.

The MEDIUMTEXT is useful for storing quite large text data like the text of a book, white papers, etc. For example:

## 

## **LONGTEXT – 4GB (4,294,967,295 characters)**

The LONGTEXT can store text data up to 4GB, which is quite big in common scenarios. It has 4 bytes overhead.

## **Summary**

* Use the TEXT data type to store long texts in the database.

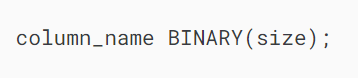
# **MySQL BINARY Data Type**

**Summary**: in this tutorial, you will learn about MySQL BINARY data type and how to use it to store fixed-length binary data.

## **Introduction to MySQL BINARY data type**

The BINARY data type is used to store fixed-length binary data. For example, you can use BINARY data type for columns that store hashes and checksums such as SHA-256 because these values have a fixed length.

To declare a column that uses the BINARY data type, you specify the maximum length of binary data it can hold:



In this syntax, the size specifies the number of bytes that the column name can store.

### **Right-padding with 0x00 (zero bytes)**

[When you insert a binary value](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) whose length is less than the specified length for the BINARY column, MySQL will automatically pad the value with zero bytes (0x00) on the right side to reach the defined size length.

### **No trailing byte removal for retrievals**

When you retrieve a value from the BINARY column, MySQL does not remove any trailing zero bytes that were padded during insertion.

In other words, if you inserted a binary value and it was right-padded with zero bytes, those zero bytes will be present when you retrieve the data.

### **All bytes are significant in comparisons**

When comparing BINARY value in the [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-binary/) clause, [ORDER](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) clause, or [DISTINCT](https://www.mysqltutorial.org/mysql-basics/mysql-distinct/), MySQL considers all bytes.

This means that even trailing zero bytes can affect the comparison results, and MySQL will not consider two BINARY values are equal unless all of their bytes match.

### **0x00 and space differ in comparisons**

MySQL treats the zero bytes (0x00) and the space character (0x20) differently in comparisons.

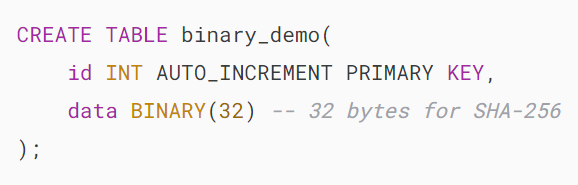
If you have a BINARY column with values that contain zero bytes and space characters, MySQL will not consider these values to be equal.

Additionally, MySQL places null bytes before space characters in sorting operations (e.g., ORDER BY).

## **MySQL BINARY data type example**

We’ll take an example of using the BINARY data type to store SHA-256 hashes.

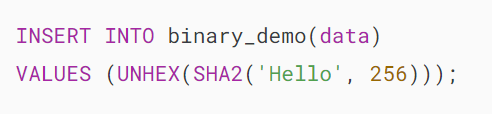
First, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) to store the SHA-256 hashes:



The binary\_demo has two columns:

* id: An [auto-incremented](https://www.mysqltutorial.org/mysql-basics/mysql-auto_increment/) [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) column.
* data: A BINARY column with a fixed size of 32 bytes to store SHA-256 hashes.

Second, insert a SHA-256 hash into the table:



The SHA2('Hello', 256) computes the SHA-256 hash of the string ‘Hello’.

The UNHEX() function converts the hexadecimal representation of the SHA-256 hash into binary data before inserting it into the BINARY column.

Third, retrieve data from the BINARY column and convert the data back to its hexadecimal using the HEX() function:

## 

## 

## **Summary**

* Use BINARY data type to store fixed-length binary data such as hashes or UUID.

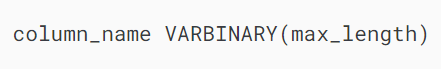
# **MySQL VARBINARY Data Type**

**Summary**: in this tutorial, you will learn how to use the MySQL VARBINARY data type to store variable-length binary data.

## **Introduction to MySQL VARBINARY data type**

The VARBINARY data type is used to store variable-length binary data. It is similar to the [BINARY](https://www.mysqltutorial.org/mysql-basics/mysql-binary/) data type but allows you to store **binary data of variable length**.

The following shows how to define a VARBINARY column in a table:



In this syntax, we define the column\_name with the VARBINARY data type that can store up to max\_length bytes. The maximum value for the max\_length is 65,535 bytes, which is equivalent to 64KB.

In practice, you often use the VARBINARY data type for storing variable binary data such as small images, audio files, and other non-textual data.

Unlike the [BINARY](https://www.mysqltutorial.org/mysql-basics/mysql-binary/) data type, [When you insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) data into a VARBINARY column, MySQL does not pad zero bytes (0x00) if the length of the data is not equal to the max\_length of the column. Additionally, MySQL will not strip any bytes when you retrieve data from a VARBINARY column.

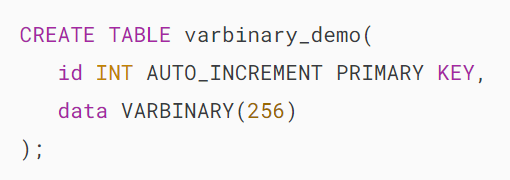
If you sort VARBINARY data, MySQL treats zero bytes (0x00) and space differently in sorting operations such as ORDER BY and DISTINCT. It places the zero bytes (0x00) before the space.

When you insert data whose length exceeds the max\_length, MySQL drops extra bytes and issues a warning if SQL strict mode is not enabled and an error if the SQL strict mode is enabled.

## **MySQL VARBINARY data type example**

We’ll take an example of using the VARBINARY data type to define a column that stores data.

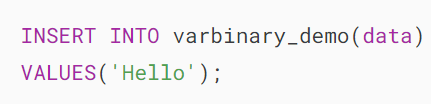
First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) called varbinary\_demo:



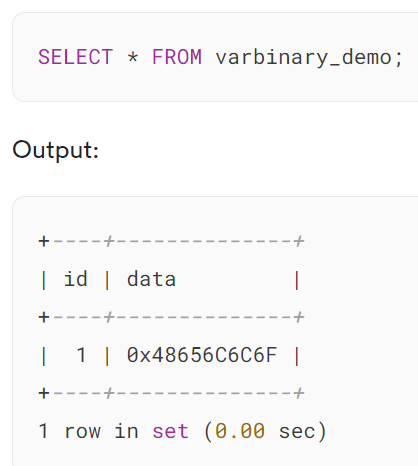
The table varbinary\_demo has two columns:

* id: [Auto-incremented](https://www.mysqltutorial.org/mysql-basics/mysql-auto_increment/) [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) column.
* data: VARBINARY data column.

Second, [insert a new row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the varbinary\_demo table:



Third, select data from the varbinary\_demo table:



## **Summary**

* Use MySQL VARBINARY data type to define a column that can store variable binary data.

# **MySQL BLOB**

**Summary**: in this tutorial, you will learn about MySQL BLOB data type and its applications.

## **Introduction to MySQL BLOB data type**

In MySQL, a BLOB (Binary Large Object) is a data type that allows you to store large binary data, such as images, audio, video, and so on. BLOBs are useful when you want to store and retrieve data in your database.

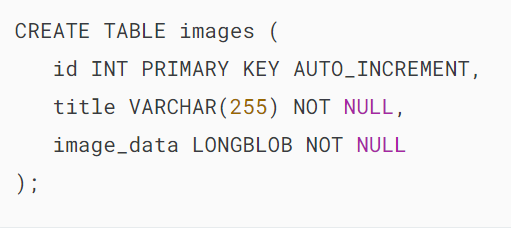
MySQL supports the following types of BLOBs:

* **TINYBLOB**: Maximum length of 255 bytes.
* **BLOB**: Maximum length of 65,535 bytes.
* **MEDIUMBLOB**: Maximum length of 16,777,215 bytes.
* **LONGBLOB**: Maximum length of 4,294,967,295 bytes.

When you create tables that store BLOBs, you select the appropriate BLOB type based on the size of binary data you plan to store.

## **MySQL BLOB data type example**

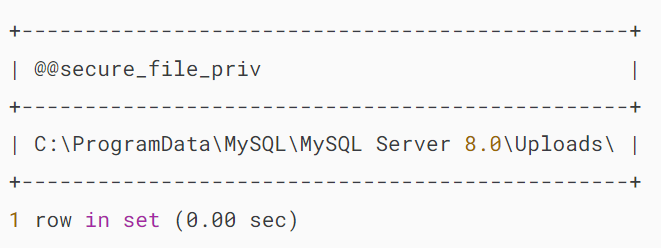
First, [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) that includes a BLOB column:



Second, show the value of the secure\_file\_priv variable:



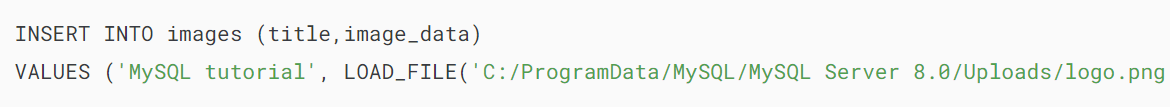
Output:



The secure\_file\_priv system variable restricts the locations on the MySQL server from which the LOAD\_FILE() function can read files.

If you attempt to load the file from other locations, the LOAD\_FILE() function returns NULL.

Third, insert a binary image into the image\_data of the images table using the LOAD\_FILE() function:



Make sure to replace C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/logo.png with the actual absolute path to your image file.

Notice that you need to replace the backslash (\) with the forward-slash (/) in the path to the file on Windows to make it work properly.

Finally, retrieve the binary data from the BLOB column:



In practice, you often use applications written in Python, PHP, Java, and so on to read files in the client and store them in the BLOB column:

* [PHP BLOB](https://www.mysqltutorial.org/php-mysql/php-mysql-blob/)
* [Python BLOB](https://www.mysqltutorial.org/python-mysql/python-mysql-blob/)
* [Java BLOB](https://www.mysqltutorial.org/mysql-jdbc-tutorial/writing-and-reading-mysql-blob-using-jdbc/)

## **Summary**

* Use MySQL BLOB to store large binary data in the database.

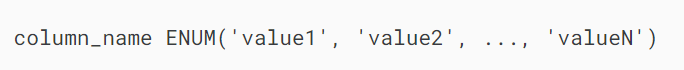
# **MySQL ENUM**

**Summary**: in this tutorial, you will learn how to use MySQL ENUM data type for defining columns that store enumeration values.

## **Introduction to MySQL ENUM data type**

In MySQL, an ENUM is a string object whose value is chosen from a list of permitted values defined at the time of column creation.

To define an ENUM column, you use the following syntax:



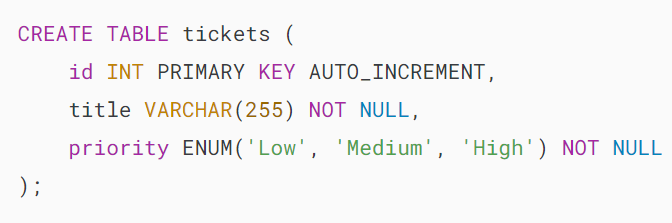
In this syntax:

* column\_name: This is the name of the column that uses the ENUM data type
* 'value1', 'value2', … 'valueN': These are the list of values that the column can hold. The values are separated by commas.

## **MySQL ENUM data type example**

Suppose you have to store ticket information with the priority: low, medium, and high. To assign these string values to the priority column, you can use the ENUM data type.

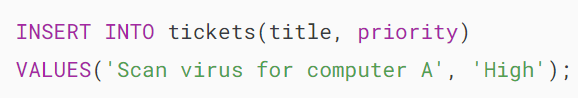
First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) that includes a priority column with the ENUM type:



The priority column accepts only three values Low, Medium and High.

Behind the scenes, MySQL maps each enumeration member to a numeric index. In this case, it maps the Low, Medium, and High values to 1, 2, and 3 respectively.

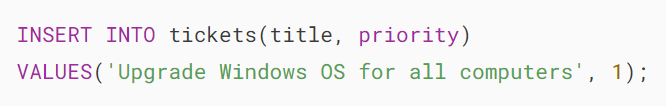
Second, [insert a new row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the tickets table:



In this example, we use the predefined value 'High' to insert into the priority column.

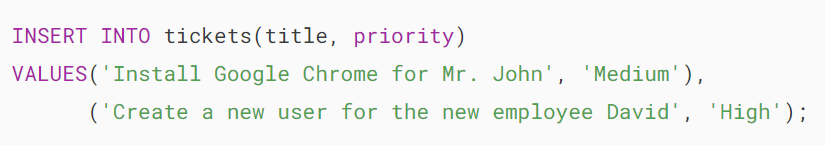
Besides the enumeration values, you can use the numeric index of the enumeration member to insert data into an ENUM column.

Third, insert a new row into the tickets table using a numeric index value instead of the predefined values:

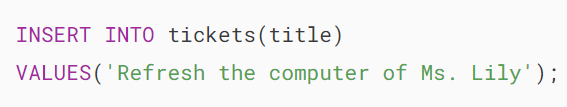


In this example, instead of using the Low enumeration value, we used value 1. Since Low is mapped to 1, it is acceptable.

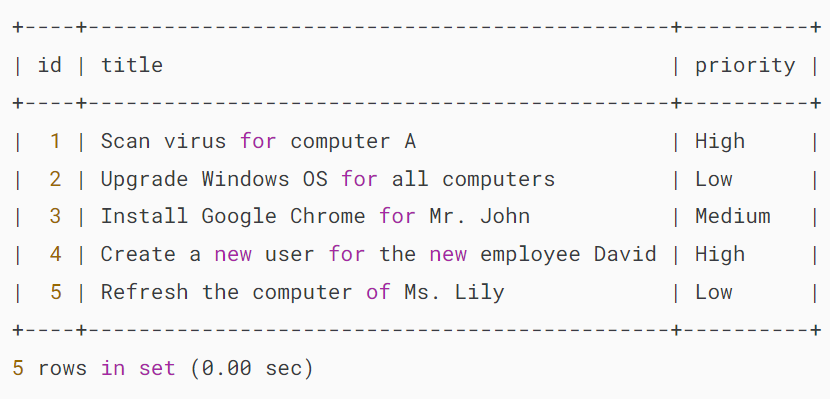
Fourth, [insert multiple rows](https://www.mysqltutorial.org/mysql-basics/mysql-insert-multiple-rows/) into the tickets table:



Because we define the priority as a NOT NULL column, when you insert a new row without specifying the value for the priority column, MySQL will use the first enumeration member as the default value. For example:

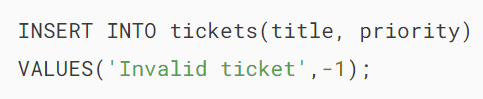


The contents of the tickets table are as follows:

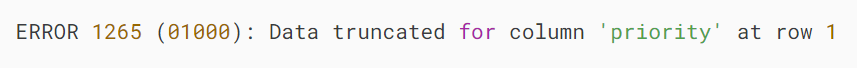


In the non-strict SQL mode, if you insert an invalid value into an ENUM column, MySQL will use an empty string '' with the numeric index 0 for inserting.

If you enable the SQL strict mode and you attempt to insert an invalid ENUM value, you will get an error. For example:



Error:



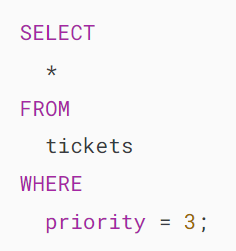
Note that an ENUM column can accept NULL values if you define it as a nullable column.

## **Filtering MySQL ENUM values**

The following statement retrieves all the tickets with high priority:



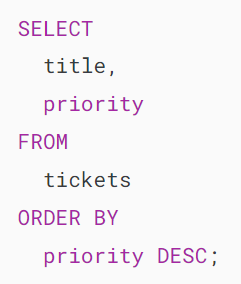
Because the enumeration member 'High' is mapped to 3, the following query returns the same result set:



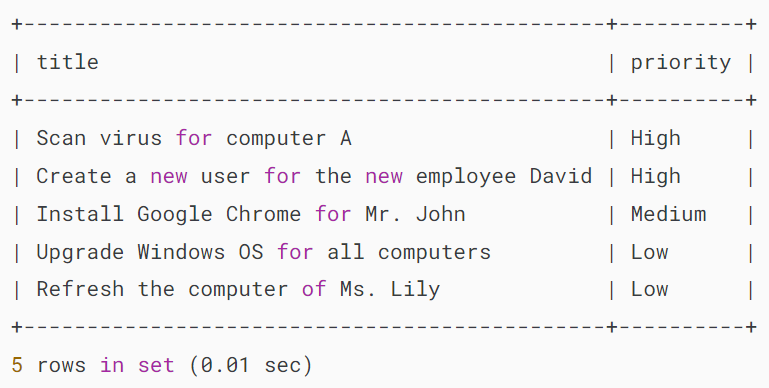
## **Sorting MySQL ENUM values**

MySQL [sorts](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/) ENUM values based on their index numbers. Therefore, the order of members depends on how they were defined in the enumeration list.

The following query selects the tickets and sorts them by priority from High to Low:



Output:



It’s a good practice to define the enumeration values in the order that you want to sort when you create the ENUM column.

## **Advantages of ENUM data type**

* **Data Validation:** ENUM data types provide strong data validation because they restrict column values to a predefined set of options. This helps maintain data integrity.
* **Readability:** ENUM values are human-readable and self-explanatory, making it easy to understand the data in the column.
* **Space Efficiency:** ENUM values are stored as integers, which are more space-efficient than storing strings.

## **Limitations of ENUM data type**

* **Limited Flexibility:** Once ENUM values are defined, they cannot be easily changed or extended. If you need to add or remove values, you may need to alter the table structure, which can be a complex operation.
* **Portability:** The ENUM data type is specific to MySQL and may not be supported in other database systems.
* **Maintenance:** ENUM values can make the schema harder to maintain as the application evolves, as adding or removing values can be complex.

## **Summary**

* Use MySQL ENUM for defining columns with a limited set of allowed values.

# **MySQL UUID Smackdown: UUID vs. INT for Primary Key**

**Summary**: This tutorial introduces you to MySQL UUID, shows you how to use it as the primary key for a table, and discusses the pros and cons of using it as the primary key.

## **Introduction to MySQL UUID**

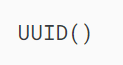
UUID stands for Universally Unique IDentifier. UUID is defined based on [RFC 4122](https://tools.ietf.org/html/rfc4122), “a Universally Unique Identifier (UUID) URN Namespace”.

UUID is designed as a number that is unique globally in space and time. Two UUID values are expected to be distinct, even if they are generated on two independent servers.

In MySQL, a UUID value is a 128-bit number represented as a utf8 string of five hexadecimal numbers in the following format:



To generate a UUID value, you use the UUID() function as follows:



The UUID() function returns a UUID value in compliance with UUID version 1 described in the RFC 4122.

For example, the following statement uses the UUID() function to generate a UUID value:

## 

## **MySQL UUID vs. Auto-Increment INT as the primary key**

### **Pros**

Using UUID for a [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) has the following advantages:

* UUID values are unique across tables, databases, and even servers that allow you to merge rows from different databases or distribute databases across servers.
* UUID values do not expose the information about your data so they are safer to use in a URL. For example, if a customer with ID 10 accesses his account via http://www.example.com/customers/10/ URL, it is easy to guess that there is a customer 11, 12, etc., and this could be a target for an attack.
* UUID values can be generated anywhere which can help avoid a round trip to the database server. It also simplifies logic in the application. For example, to insert data into a parent table and child tables, you have to insert into the parent table first, get the generated ID, and then insert data into the child tables. By using UUID, you can generate the primary key value of the parent table up front and insert rows into both parent and child tables at the same time within a [transaction](https://www.mysqltutorial.org/mysql-stored-procedure/mysql-transactions/).

### **Cons**

Besides the advantages, UUID values also come with some disadvantages:

* Storing UUID values (16 bytes) takes more storage than integers (4 bytes) or even big integers(8 bytes).
* Debugging seems to be  more difficult, imagine the expression WHERE id = 'df3b7cb7-6a95-11e7-8846-b05adad3f0ae' instead of WHERE id = 10
* Using UUID values may cause performance issues due to their size and not being ordered.

## **MySQL UUID solution**

In MySQL, you can store UUID values in a compact format (BINARY) and display them in human-readable format ([VARCHAR](https://www.mysqltutorial.org/mysql-basics/mysql-varchar/)) with the help of the following functions:

* UUID\_TO\_BIN
* BIN\_TO\_UUID
* IS\_UUID

Notice that UUID\_TO\_BIN(), BIN\_TO\_UUID(), and IS\_UUID() functions are only available in MySQL 8.0 or later.

The UUID\_TO\_BIN() function converts a UUID from a human-readable format (VARCHAR) into a compact format (BINARY) format for storing and the BIN\_TO\_UUID() function converts UUID from the compact format (BINARY) to human-readable format (VARCHAR) for displaying.

The IS\_UUID() function returns 1 if the argument is a valid string-format UUID. If the argument is not valid string format UUID, the IS\_UUID function returns 0. In case the argument is NULL, the IS\_UUID() function returns NULL.

The following are the valid string-format UUIDs in MySQL:

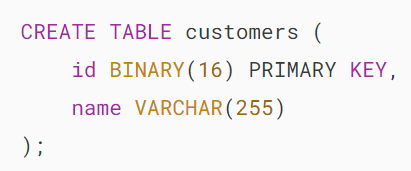
## 

## **MySQL UUID examples**

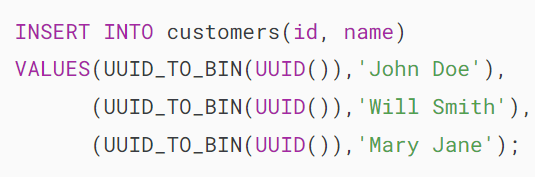
Let’s take a look at an example of using UUID as the primary key.

### **1) Basic MySQL UUID example**

First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) called customers:



Second, [insert](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) UUID values into the id column using the UUID() and UUID\_TO\_BIN() functions as follows:

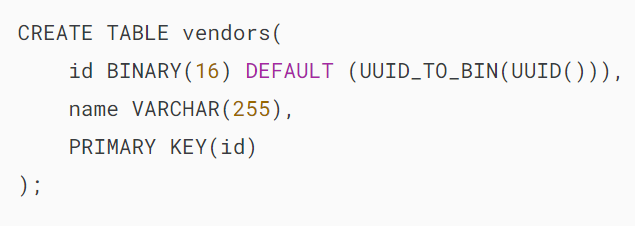


Third, query data from a UUID column and use BIN\_TO\_UUID() function to convert binary format to human-readable format:

### 

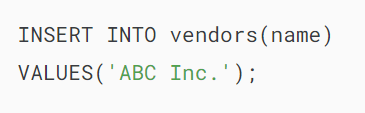
### **2) Using UUID as the default value for the primary key**

First, create a new table called vendors:



In the vendors table, we use the result of the expression UUID\_TO\_BIN(UUID()) as the default value for the primary key column. Therefore, we don’t need to specify the value for the id column whenever we insert a new row into the table.

Second, insert a new row into the vendors table:



Third, retrieve data from the vendors:

