**String Data Type Syntax**

[**String Data Type Syntax** 1](#_Toc58439808)

[The CHAR and VARCHAR Types 4](#_Toc58439809)

[The BINARY and VARBINARY Types 5](#_Toc58439810)

[The BLOB and TEXT Types 6](#_Toc58439811)

[The ENUM Type 8](#_Toc58439812)

[Creating and Using ENUM Columns 8](#_Toc58439813)

[Index Values for Enumeration Literals 8](#_Toc58439814)

[Handling of Enumeration Literals 9](#_Toc58439815)

[Empty or NULL Enumeration Values 9](#_Toc58439816)

[Enumeration Sorting 10](#_Toc58439817)

[Enumeration Limitations 10](#_Toc58439818)

[The SET Type 10](#_Toc58439819)

The string data types are [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [BINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html), [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html), [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html), [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html), [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html), and [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html).

In some cases, MySQL may change a string column to a type different from that given in a [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) or [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) statement. See [Section 13.1.20.7, “Silent Column Specification Changes”](https://dev.mysql.com/doc/refman/8.0/en/silent-column-changes.html).

For definitions of character string columns ([CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), and the [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) types), MySQL interprets length specifications in character units. For definitions of binary string columns ([BINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html), [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html), and the [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) types), MySQL interprets length specifications in byte units.

Column definitions for character string data types [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), the [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) types, [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html), [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html), and any synonyms) can specify the column character set and collation:

* CHARACTER SET specifies the character set. If desired, a collation for the character set can be specified with the COLLATE attribute, along with any other attributes. For example:
* CREATE TABLE t
* (
* c1 VARCHAR(20) CHARACTER SET utf8,
* c2 TEXT CHARACTER SET latin1 COLLATE latin1\_general\_cs

);

This table definition creates a column named c1 that has a character set of utf8 with the default collation for that character set, and a column named c2 that has a character set of latin1 and a case-sensitive (\_cs) collation.

The rules for assigning the character set and collation when either or both of CHARACTER SET and the COLLATE attribute are missing are described in [Section 10.3.5, “Column Character Set and Collation”](https://dev.mysql.com/doc/refman/8.0/en/charset-column.html).

CHARSET is a synonym for CHARACTER SET.

* Specifying the CHARACTER SET binary attribute for a character string data type causes the column to be created as the corresponding binary string data type: [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) becomes [BINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html), [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) becomes [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html), and [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) becomes [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html). For the [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html) and [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:
* CREATE TABLE t
* (
* c1 VARCHAR(10) CHARACTER SET binary,
* c2 TEXT CHARACTER SET binary,
* c3 ENUM('a','b','c') CHARACTER SET binary

);

The resulting table has this definition:

CREATE TABLE t

(

c1 VARBINARY(10),

c2 BLOB,

c3 ENUM('a','b','c') CHARACTER SET binary

);

* The BINARY attribute is a nonstandard MySQL extension that is shorthand for specifying the binary (\_bin) collation of the column character set (or of the table default character set if no column character set is specified). In this case, comparison and sorting are based on numeric character code values. Suppose that you specify a table using this definition:
* CREATE TABLE t
* (
* c1 VARCHAR(10) CHARACTER SET latin1 BINARY,
* c2 TEXT BINARY

) CHARACTER SET utf8mb4;

The resulting table has this definition:

CREATE TABLE t (

c1 VARCHAR(10) CHARACTER SET latin1 COLLATE latin1\_bin,

c2 TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin

) CHARACTER SET utf8mb4;

In MySQL 8.0, this nonstandard use of the BINARY attribute is ambiguous because the utf8mb4 character set has multiple \_bin collations. As of MySQL 8.0.17, the BINARY attribute is deprecated and you should expect support for it to be removed in a future version of MySQL. Applications should be adjusted to use an explicit \_bin collation instead.

The use of BINARY to specify a data type or character set remains unchanged.

* The ASCII attribute is shorthand for CHARACTER SET latin1.
* The UNICODE attribute is shorthand for CHARACTER SET ucs2.

Character column comparison and sorting are based on the collation assigned to the column. For the [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html), [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html), and [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) data types, you can declare a column with a binary (\_bin) collation or the BINARY attribute to cause comparison and sorting to use the underlying character code values rather than a lexical ordering.

For additional information about use of character sets in MySQL, see [Chapter 10, *Character Sets, Collations, Unicode*](https://dev.mysql.com/doc/refman/8.0/en/charset.html).

* [NATIONAL] CHAR[(***M***)] [CHARACTER SET ***charset\_name***] [COLLATE ***collation\_name***]

A fixed-length string that is always right-padded with spaces to the specified length when stored. ***M*** represents the column length in characters. The range of ***M*** is 0 to 255. If ***M*** is omitted, the length is 1.

**Note**

Trailing spaces are removed when [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) values are retrieved unless the [PAD\_CHAR\_TO\_FULL\_LENGTH](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_pad_char_to_full_length) SQL mode is enabled.

[CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) is shorthand for [CHARACTER](https://dev.mysql.com/doc/refman/8.0/en/char.html). [NATIONAL CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) (or its equivalent short form, [NCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html)) is the standard SQL way to define that a [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column should use some predefined character set. MySQL uses utf8 as this predefined character set. [Section 10.3.7, “The National Character Set”](https://dev.mysql.com/doc/refman/8.0/en/charset-national.html).

The [CHAR BYTE](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html) data type is an alias for the [BINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html) data type. This is a compatibility feature.

MySQL permits you to create a column of type CHAR(0). This is useful primarily when you must be compliant with old applications that depend on the existence of a column but that do not actually use its value. CHAR(0) is also quite nice when you need a column that can take only two values: A column that is defined as CHAR(0) NULL occupies only one bit and can take only the values NULL and '' (the empty string).

* [NATIONAL] VARCHAR(***M***) [CHARACTER SET ***charset\_name***] [COLLATE ***collation\_name***]

A variable-length string. ***M*** represents the maximum column length in characters. The range of ***M*** is 0 to 65,535. The effective maximum length of a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, utf8 characters can require up to three bytes per character, so a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column that uses the utf8 character set can be declared to be a maximum of 21,844 characters. See [Section  “Limits on Table Column Count and Row Size”](https://dev.mysql.com/doc/refman/8.0/en/column-count-limit.html).

MySQL stores [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) values as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

**Note**

MySQL follows the standard SQL specification, and does *not* remove trailing spaces from [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) values.

[VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) is shorthand for [CHARACTER VARYING](https://dev.mysql.com/doc/refman/8.0/en/char.html). [NATIONAL VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) is the standard SQL way to define that a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column should use some predefined character set. MySQL uses utf8 as this predefined character set. [Section 10.3.7, “The National Character Set”](https://dev.mysql.com/doc/refman/8.0/en/charset-national.html). [NVARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) is shorthand for [NATIONAL VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html).

* [BINARY[(](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html" \o "11.3.3 The BINARY and VARBINARY Types)***[M](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html" \o "11.3.3 The BINARY and VARBINARY Types)***[)]](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html" \o "11.3.3 The BINARY and VARBINARY Types)

The [BINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html) type is similar to the [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) type, but stores binary byte strings rather than nonbinary character strings. An optional length ***M*** represents the column length in bytes. If omitted, ***M*** defaults to 1.

* [VARBINARY(](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html" \o "11.3.3 The BINARY and VARBINARY Types)***[M](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html" \o "11.3.3 The BINARY and VARBINARY Types)***[)](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html" \o "11.3.3 The BINARY and VARBINARY Types)

The [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html) type is similar to the [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) type, but stores binary byte strings rather than nonbinary character strings. ***M*** represents the maximum column length in bytes.

* [TINYBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 255 (2**8** − 1) bytes. Each [TINYBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

* [TINYTEXT [CHARACTER SET](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[charset\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[] [COLLATE](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[collation\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[]](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 255 (2**8** − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each [TINYTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

* [BLOB[(](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[M](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[)]](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 65,535 (2**16** − 1) bytes. Each [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

An optional length ***M*** can be given for this type. If this is done, MySQL creates the column as the smallest [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) type large enough to hold values ***M*** bytes long.

* [TEXT[(](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[M](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[)] [CHARACTER SET](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[charset\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[] [COLLATE](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[collation\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[]](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 65,535 (2**16** − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

An optional length ***M*** can be given for this type. If this is done, MySQL creates the column as the smallest [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) type large enough to hold values ***M*** characters long.

* [MEDIUMBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 16,777,215 (2**24** − 1) bytes. Each [MEDIUMBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

* [MEDIUMTEXT [CHARACTER SET](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[charset\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[] [COLLATE](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[collation\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[]](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 16,777,215 (2**24** − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each [MEDIUMTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

* [LONGBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 4,294,967,295 or 4GB (2**32** − 1) bytes. The effective maximum length of [LONGBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) columns depends on the configured maximum packet size in the client/server protocol and available memory. Each [LONGBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

* [LONGTEXT [CHARACTER SET](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[charset\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[] [COLLATE](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[collation\_name](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)***[]](https://dev.mysql.com/doc/refman/8.0/en/blob.html" \o "11.3.4 The BLOB and TEXT Types)

A [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column with a maximum length of 4,294,967,295 or 4GB (2**32** − 1) characters. The effective maximum length is less if the value contains multibyte characters. The effective maximum length of [LONGTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each [LONGTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

* [ENUM('](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[value1](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[','](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[value2](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[',...) [CHARACTER SET](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[charset\_name](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[] [COLLATE](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[collation\_name](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)***[]](https://dev.mysql.com/doc/refman/8.0/en/enum.html" \o "11.3.5 The ENUM Type)

An enumeration. A string object that can have only one value, chosen from the list of values '***value1***', '***value2***', ..., NULL or the special '' error value. [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html) values are represented internally as integers.

An [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html) column can have a maximum of 65,535 distinct elements.

The maximum supported length of an individual ENUM element is ***M*** <= 255 and (***M*** x ***w***) <= 1020, where M is the element literal length and ***w*** is the number of bytes required for the maximum-length character in the character set.

* [SET('](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[value1](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[','](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[value2](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[',...) [CHARACTER SET](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[charset\_name](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[] [COLLATE](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[collation\_name](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)***[]](https://dev.mysql.com/doc/refman/8.0/en/set.html" \o "11.3.6 The SET Type)

A set. A string object that can have zero or more values, each of which must be chosen from the list of values '***value1***', '***value2***', ... [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) values are represented internally as integers.

A [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) column can have a maximum of 64 distinct members.

The maximum supported length of an individual SET element is ***M*** <= 255 and (***M*** x ***w***) <= 1020, where M is the element literal length and ***w*** is the number of bytes required for the maximum-length character in the character set.

### The CHAR and VARCHAR Types

The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained.

The CHAR and VARCHAR types are declared with a length that indicates the maximum number of characters you want to store. For example, CHAR(30) can hold up to 30 characters.

The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed unless the [PAD\_CHAR\_TO\_FULL\_LENGTH](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sqlmode_pad_char_to_full_length) SQL mode is enabled.

Values in VARCHAR columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a VARCHAR is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. See [Section 8.4.7, “Limits on Table Column Count and Row Size”](https://dev.mysql.com/doc/refman/8.0/en/column-count-limit.html).

In contrast to CHAR, VARCHAR values are stored as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

If strict SQL mode is not enabled and you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.1.11, “Server SQL Modes”](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html).

For VARCHAR columns, trailing spaces in excess of the column length are truncated prior to insertion and a warning is generated, regardless of the SQL mode in use. For CHAR columns, truncation of excess trailing spaces from inserted values is performed silently regardless of the SQL mode.

VARCHAR values are not padded when they are stored. Trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL.

The following table illustrates the differences between CHAR and VARCHAR by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns (assuming that the column uses a single-byte character set such as latin1).

| **Value** | CHAR(4) | **Storage Required** | VARCHAR(4) | **Storage Required** |
| --- | --- | --- | --- | --- |
| '' | '    ' | 4 bytes | '' | 1 byte |
| 'ab' | 'ab  ' | 4 bytes | 'ab' | 3 bytes |
| 'abcd' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |
| 'abcdefgh' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |

The values shown as stored in the last row of the table apply only when not using strict SQL mode; if strict mode is enabled, values that exceed the column length are not stored, and an error results.

InnoDB encodes fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

If a given value is stored into the CHAR(4) and VARCHAR(4) columns, the values retrieved from the columns are not always the same because trailing spaces are removed from CHAR columns upon retrieval. The following example illustrates this difference:

mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));

Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab ', 'ab ');

Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('(', v, ')'), CONCAT('(', c, ')') FROM vc;

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

| CONCAT('(', v, ')') | CONCAT('(', c, ')') |

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

| (ab ) | (ab) |

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

1 row in set (0.06 sec)

Values in CHAR, VARCHAR, and TEXT columns are sorted and compared according to the character set collation assigned to the column.

MySQL collations have a pad attribute of PAD SPACE, other than Unicode collations based on UCA 9.0.0 and higher, which have a pad attribute of NO PAD. (see [Section 10.10.1, “Unicode Character Sets”](https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-sets.html)).

To determine the pad attribute for a collation, use the INFORMATION\_SCHEMA [COLLATIONS](https://dev.mysql.com/doc/refman/8.0/en/information-schema-collations-table.html) table, which has a PAD\_ATTRIBUTE column.

For nonbinary strings (CHAR, VARCHAR, and TEXT values), the string collation pad attribute determines treatment in comparisons of trailing spaces at the end of strings. NO PAD collations treat trailing spaces as significant in comparisons, like any other character. PAD SPACE collations treat trailing spaces as insignificant in comparisons; strings are compared without regard to trailing spaces. See [Trailing Space Handling in Comparisons](https://dev.mysql.com/doc/refman/8.0/en/charset-binary-collations.html#charset-binary-collations-trailing-space-comparisons). The server SQL mode has no effect on comparison behavior with respect to trailing spaces.

**Note**

For more information about MySQL character sets and collations, see [Chapter 10, *Character Sets, Collations, Unicode*](https://dev.mysql.com/doc/refman/8.0/en/charset.html). For additional information about storage requirements, see [Section 11.7, “Data Type Storage Requirements”](https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html).

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters results in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error.

### The BINARY and VARBINARY Types

The BINARY and VARBINARY types are similar to [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) and [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), except that they store binary strings rather than nonbinary strings. That is, they store byte strings rather than character strings. This means they have the binary character set and collation, and comparison and sorting are based on the numeric values of the bytes in the values.

The permissible maximum length is the same for BINARY and VARBINARY as it is for [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) and [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), except that the length for BINARY and VARBINARY is measured in bytes rather than characters.

The BINARY and VARBINARY data types are distinct from the CHAR BINARY and VARCHAR BINARY data types. For the latter types, the BINARY attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary (\_bin) collation for the column character set (or the table default character set if no column character set is specified) to be used, and the column itself stores nonbinary character strings rather than binary byte strings. For example, if the default character set is utf8mb4, CHAR(5) BINARY is treated as CHAR(5) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin. This differs from BINARY(5), which stores 5-byte binary strings that have the binary character set and collation. For information about the differences between the binary collation of the binary character set and the \_bin collations of nonbinary character sets, see [Section 10.8.5, “The binary Collation Compared to \_bin Collations”](https://dev.mysql.com/doc/refman/8.0/en/charset-binary-collations.html).

If strict SQL mode is not enabled and you assign a value to a BINARY or VARBINARY column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For cases of truncation, to cause an error to occur (rather than a warning) and suppress insertion of the value, use strict SQL mode. See [Section 5.1.11, “Server SQL Modes”](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html).

When BINARY values are stored, they are right-padded with the pad value to the specified length. The pad value is 0x00 (the zero byte). Values are right-padded with 0x00 for inserts, and no trailing bytes are removed for retrievals. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 and space differ in comparisons, with 0x00 sorting before space.

Example: For a BINARY(3) column, 'a ' becomes 'a \0' when inserted. 'a\0' becomes 'a\0\0' when inserted. Both inserted values remain unchanged for retrievals.

For VARBINARY, there is no padding for inserts and no bytes are stripped for retrievals. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 and space differ in comparisons, with 0x00 sorting before space.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting values into the column that differ only in number of trailing pad bytes results in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a\0' causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the BINARY data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how 0x00-padding of BINARY values affects column value comparisons:

mysql> CREATE TABLE t (c BINARY(3));

Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';

Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;

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

| HEX(c) | c = 'a' | c = 'a\0\0' |

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

| 610000 | 0 | 1 |

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

1 row in set (0.09 sec)

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use VARBINARY or one of the [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) data types instead.

The BLOB and TEXT Types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the values they can hold. The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. These correspond to the four BLOB types and have the same maximum lengths and storage requirements. See [Section 11.7, “Data Type Storage Requirements”](https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html).

BLOB values are treated as binary strings (byte strings). They have the binary character set and collation, and comparison and sorting are based on the numeric values of the bytes in column values. TEXT values are treated as nonbinary strings (character strings). They have a character set other than binary, and values are sorted and compared based on the collation of the character set.

If strict SQL mode is not enabled and you assign a value to a BLOB or TEXT column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.1.11, “Server SQL Modes”](https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html).

Truncation of excess trailing spaces from values to be inserted into [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) columns always generates a warning, regardless of the SQL mode.

For TEXT and BLOB columns, there is no padding on insert and no bytes are stripped on select.

If a TEXT column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors occur for values that differ only in the number of trailing spaces. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error. This is not true for BLOB columns.

In most respects, you can regard a BLOB column as a [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html) column that can be as large as you like. Similarly, you can regard a TEXT column as a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column. BLOB and TEXT differ from [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html) and [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) in the following ways:

* For indexes on BLOB and TEXT columns, you must specify an index prefix length. For [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) and [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), a prefix length is optional. See [Section 8.3.5, “Column Indexes”](https://dev.mysql.com/doc/refman/8.0/en/column-indexes.html).
* BLOB and TEXT columns cannot have DEFAULT values.

If you use the BINARY attribute with a TEXT data type, the column is assigned the binary (\_bin) collation of the column character set.

LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature.

MySQL Connector/ODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values can be extremely long, you might encounter some constraints in using them:

* Only the first [max\_sort\_length](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_max_sort_length) bytes of the column are used when sorting. The default value of [max\_sort\_length](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_max_sort_length) is 1024. You can make more bytes significant in sorting or grouping by increasing the value of [max\_sort\_length](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_max_sort_length) at server startup or runtime. Any client can change the value of its session [max\_sort\_length](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_max_sort_length) variable:
* mysql> SET max\_sort\_length = 2000;
* mysql> SELECT id, comment FROM t

-> ORDER BY comment;

* Instances of BLOB or TEXT columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the MEMORY storage engine does not support those data types (see [Section 8.4.4, “Internal Temporary Table Use in MySQL”](https://dev.mysql.com/doc/refman/8.0/en/internal-temporary-tables.html)). Use of disk incurs a performance penalty, so include BLOB or TEXT columns in the query result only if they are really needed. For example, avoid using [SELECT \*](https://dev.mysql.com/doc/refman/8.0/en/select.html), which selects all columns.
* The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing the value of the [max\_allowed\_packet](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_max_allowed_packet) variable, but you must do so for both the server and your client program. For example, both **[mysql](https://dev.mysql.com/doc/refman/8.0/en/mysql.html" \o "4.5.1 mysql — The MySQL Command-Line Client)** and **[mysqldump](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html" \o "4.5.4 mysqldump — A Database Backup Program)** enable you to change the client-side [max\_allowed\_packet](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_max_allowed_packet) value. See [Section 5.1.1, “Configuring the Server”](https://dev.mysql.com/doc/refman/8.0/en/server-configuration.html), [Section 4.5.1, “**mysql** — The MySQL Command-Line Client”](https://dev.mysql.com/doc/refman/8.0/en/mysql.html), and [Section 4.5.4, “**mysqldump** — A Database Backup Program”](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html). You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see [Section 11.7, “Data Type Storage Requirements”](https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html)

Each BLOB or TEXT value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in BLOB or TEXT columns. You may find MySQL's string handling functions useful for working with such data. See [Section 12.8, “String Functions and Operators”](https://dev.mysql.com/doc/refman/8.0/en/string-functions.html). For security and other reasons, it is usually preferable to do so using application code rather than giving application users the [FILE](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_file) privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (<http://forums.mysql.com/>).

### The ENUM Type

An ENUM is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time.

See [Section 11.3.1, “String Data Type Syntax”](https://dev.mysql.com/doc/refman/8.0/en/string-type-syntax.html) for [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html) type syntax and length limits.

The [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html) type has these advantages:

* Compact data storage in situations where a column has a limited set of possible values. The strings you specify as input values are automatically encoded as numbers. See [Section 11.7, “Data Type Storage Requirements”](https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html) for storage requirements for the ENUM type.
* Readable queries and output. The numbers are translated back to the corresponding strings in query results.

and these potential issues to consider:

* If you make enumeration values that look like numbers, it is easy to mix up the literal values with their internal index numbers, as explained in [Enumeration Limitations](https://dev.mysql.com/doc/refman/8.0/en/enum.html#enum-limits).
* Using ENUM columns in ORDER BY clauses requires extra care, as explained in [Enumeration Sorting](https://dev.mysql.com/doc/refman/8.0/en/enum.html#enum-sorting).

#### Creating and Using ENUM Columns

An enumeration value must be a quoted string literal. For example, you can create a table with an ENUM column like this:

CREATE TABLE shirts (

name VARCHAR(40),

size ENUM('x-small', 'small', 'medium', 'large', 'x-large')

);

INSERT INTO shirts (name, size) VALUES ('dress shirt','large'), ('t-shirt','medium'),

('polo shirt','small');

SELECT name, size FROM shirts WHERE size = 'medium';

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

| name | size |

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

| t-shirt | medium |

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

UPDATE shirts SET size = 'small' WHERE size = 'large';

COMMIT;

Inserting 1 million rows into this table with a value of 'medium' would require 1 million bytes of storage, as opposed to 6 million bytes if you stored the actual string 'medium' in a VARCHAR column.

#### Index Values for Enumeration Literals

Each enumeration value has an index:

* The elements listed in the column specification are assigned index numbers, beginning with 1.
* The index value of the empty string error value is 0. This means that you can use the following [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) statement to find rows into which invalid ENUM values were assigned:

mysql> SELECT \* FROM tbl\_name WHERE enum\_col=0;

* The index of the NULL value is NULL.
* The term “index” here refers to a position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as ENUM('Mercury', 'Venus', 'Earth') can have any of the values shown here. The index of each value is also shown.

| **Value** | **Index** |
| --- | --- |
| NULL | NULL |
| '' | 0 |
| 'Mercury' | 1 |
| 'Venus' | 2 |
| 'Earth' | 3 |

An [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html) column can have a maximum of 65,535 distinct elements.

If you retrieve an ENUM value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an ENUM column like this:

mysql> SELECT enum\_col+0 FROM tbl\_name;

Functions such as [SUM()](https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_sum) or [AVG()](https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_avg) that expect a numeric argument cast the argument to a number if necessary. For ENUM values, the index number is used in the calculation.

#### Handling of Enumeration Literals

Trailing spaces are automatically deleted from ENUM member values in the table definition when a table is created.

When retrieved, values stored into an ENUM column are displayed using the lettercase that was used in the column definition. Note that ENUM columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you store a number into an ENUM column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does not work with [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html), which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an ENUM column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of '0', '1', and '2', but numeric index values of 1, 2, and 3:

numbers ENUM('0','1','2')

If you store 2, it is interpreted as an index value, and becomes '1' (the value with index 2). If you store '2', it matches an enumeration value, so it is stored as '2'. If you store '3', it does not match any enumeration value, so it is treated as an index and becomes '2' (the value with index 3).

mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');

mysql> SELECT \* FROM t;

+---------+

| numbers |

+---------+

| 1 |

| 2 |

| 2 |

+---------+

To determine all possible values for an ENUM column, use [SHOW COLUMNS FROM ***tbl\_name*** LIKE '***enum\_col***'](https://dev.mysql.com/doc/refman/8.0/en/show-columns.html) and parse the ENUM definition in the Type column of the output.

In the C API, ENUM values are returned as strings. For information about using result set metadata to distinguish them from other strings, see [C API Data Structures](https://dev.mysql.com/doc/c-api/8.0/en/c-api-data-structures.html).

#### Empty or NULL Enumeration Values

An enumeration value can also be the empty string ('') or NULL under certain circumstances:

* If you insert an invalid value into an ENUM (that is, a string not present in the list of permitted values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numeric value 0. See [Index Values for Enumeration Literals](https://dev.mysql.com/doc/refman/8.0/en/enum.html#enum-indexes) for details about the numeric indexes for the enumeration values.

If strict SQL mode is enabled, attempts to insert invalid ENUM values result in an error.

* If an ENUM column is declared to permit NULL, the NULL value is a valid value for the column, and the default value is NULL. If an ENUM column is declared NOT NULL, its default value is the first element of the list of permitted values.

#### Enumeration Sorting

ENUM values are sorted based on their index numbers, which depend on the order in which the enumeration members were listed in the column specification. For example, 'b' sorts before 'a' for ENUM('b', 'a'). The empty string sorts before nonempty strings, and NULL values sort before all other enumeration values.

To prevent unexpected results when using the ORDER BY clause on an ENUM column, use one of these techniques:

* Specify the ENUM list in alphabetic order.
* Make sure that the column is sorted lexically rather than by index number by coding ORDER BY CAST(***col*** AS CHAR) or ORDER BY CONCAT(***col***).

#### Enumeration Limitations

An enumeration value cannot be an expression, even one that evaluates to a string value.

For example, this [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statement does not work because the CONCAT function cannot be used to construct an enumeration value:

CREATE TABLE sizes (

size ENUM('small', CONCAT('med','ium'), 'large')

);

You also cannot employ a user variable as an enumeration value. This pair of statements do not work:

SET @mysize = 'medium';

CREATE TABLE sizes (

size ENUM('small', @mysize, 'large')

);

We strongly recommend that you do not use numbers as enumeration values, because it does not save on storage over the appropriate [TINYINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) or [SMALLINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) type, and it is easy to mix up the strings and the underlying number values (which might not be the same) if you quote the ENUM values incorrectly. If you do use a number as an enumeration value, always enclose it in quotation marks. If the quotation marks are omitted, the number is regarded as an index. See [Handling of Enumeration Literals](https://dev.mysql.com/doc/refman/8.0/en/enum.html#enum-literals) to see how even a quoted number could be mistakenly used as a numeric index value.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

### The SET Type

A SET is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (,). A consequence of this is that SET member values should not themselves contain commas.

For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:

''

'one'

'two'

'one,two'

A [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) column can have a maximum of 64 distinct members.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

Trailing spaces are automatically deleted from SET member values in the table definition when a table is created.

See [String Type Storage Requirements](https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html#data-types-storage-reqs-strings) for storage requirements for the [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) type.

See [Section 11.3.1, “String Data Type Syntax”](https://dev.mysql.com/doc/refman/8.0/en/string-type-syntax.html) for [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) type syntax and length limits.

When retrieved, values stored in a SET column are displayed using the lettercase that was used in the column definition. Note that SET columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a SET value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a SET column like this:

mysql> SELECT set\_col+0 FROM tbl\_name;

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a','b','c','d'), the members have the following decimal and binary values.

| SET**Member** | **Decimal Value** | **Binary Value** |
| --- | --- | --- |
| 'a' | 1 | 0001 |
| 'b' | 2 | 0010 |
| 'c' | 4 | 0100 |
| 'd' | 8 | 1000 |

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. Suppose that a column is specified as SET('a','b','c','d'):

mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

mysql> INSERT INTO myset (col) VALUES

-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

Then all these values appear as 'a,d' when retrieved:

mysql> SELECT col FROM myset;

+------+

| col |

+------+

| a,d |

| a,d |

| a,d |

| a,d |

| a,d |

+------+

5 rows in set (0.04 sec)

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');

Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;

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

| Level | Code | Message |

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

| Warning | 1265 | Data truncated for column 'col' at row 1 |

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

1 row in set (0.04 sec)

mysql> SELECT col FROM myset;

+------+

| col |

+------+

| a,d |

| a,d |

| a,d |

| a,d |

| a,d |

| a,d |

+------+

6 rows in set (0.01 sec)

If strict SQL mode is enabled, attempts to insert invalid SET values result in an error.

SET values are sorted numerically. NULL values sort before non-NULL SET values.

Functions such as [SUM()](https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_sum) or [AVG()](https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_avg) that expect a numeric argument cast the argument to a number if necessary. For SET values, the cast operation causes the numeric value to be used.

Normally, you search for SET values using the [FIND\_IN\_SET()](https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_find-in-set) function or the [LIKE](https://dev.mysql.com/doc/refman/8.0/en/string-comparison-functions.html#operator_like) operator:

mysql> SELECT \* FROM tbl\_name WHERE FIND\_IN\_SET('value',set\_col)>0;

mysql> SELECT \* FROM tbl\_name WHERE set\_col LIKE '%value%';

The first statement finds rows where ***set\_col*** contains the ***value*** set member. The second is similar, but not the same: It finds rows where ***set\_col*** contains ***value*** anywhere, even as a substring of another set member.

The following statements also are permitted:

mysql> SELECT \* FROM tbl\_name WHERE set\_col & 1;

mysql> SELECT \* FROM tbl\_name WHERE set\_col = 'val1,val2';

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to '***val1***,***val2***' returns different results than comparing values to '***val2***,***val1***'. You should specify the values in the same order they are listed in the column definition.

To determine all possible values for a SET column, use SHOW COLUMNS FROM ***tbl\_name*** LIKE ***set\_col*** and parse the SET definition in the Type column of the output.

In the C API, SET values are returned as strings. For information about using result set metadata to distinguish them from other strings, see [C API Data Structures](https://dev.mysql.com/doc/c-api/8.0/en/c-api-data-structures.html).