数据类型

MySQL支持几个类别的SQL数据类型：数字类型、日期和时间类型、字符串（字符和字节）类型、空间类型和JSON数据类型。本章对每个类别中的类型的属性进行了概述和更详细的描述，并对数据类型的存储要求进行了总结。最初的概述是有意简略的。请查阅更详细的描述，以了解有关特定数据类型的额外信息，例如可以指定值的允许格式。

数据类型的描述使用这些约定。

对于整数类型，M表示最大显示宽度。对于浮点和定点类型，M是可以存储的总位数（精度）。对于字符串类型，M是最大长度。M的最大允许值取决于数据类型。***D*** applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale). The maximum possible value is 30, but should be no greater than ***M***−2.

***fsp*** applies to the [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) types and represents fractional seconds precision; that is, the number of digits following the decimal point for fractional parts of seconds. The ***fsp*** value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

Square brackets (**[** and **]**) indicate optional parts of type definitions.

## 11.1 Numeric Data Types

[11.1.1 Numeric Data Type Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#numeric-type-syntax)

[11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types)

[11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types)

[11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types)

[11.1.5 Bit-Value Type - BIT](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#bit-type)

[11.1.6 Numeric Type Attributes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#numeric-type-attributes)

[11.1.7 Out-of-Range and Overflow Handling](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#out-of-range-and-overflow)

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types ([**INTEGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), [**SMALLINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types), and [**NUMERIC**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types)), as well as the approximate numeric data types ([**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types), [**REAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types), and [**DOUBLE PRECISION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types)). The keyword [**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) is a synonym for [**INTEGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), and the keywords [**DEC**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) and [**FIXED**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) are synonyms for [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types). MySQL treats [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) as a synonym for [**DOUBLE PRECISION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) (a nonstandard extension). MySQL also treats [**REAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) as a synonym for [**DOUBLE PRECISION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) (a nonstandard variation), unless the [**REAL\_AS\_FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_real_as_float) SQL mode is enabled.

The [**BIT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#bit-type) data type stores bit values and is supported for **[MyISAM](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\storage-engines.html" \l "myisam-storage-engine" \o "16.2 The MyISAM Storage Engine)**, [**MEMORY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\storage-engines.html#memory-storage-engine), **[InnoDB](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\innodb-storage-engine.html" \o "Chapter 15 The InnoDB Storage Engine)**, and [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) tables.

For information about how MySQL handles assignment of out-of-range values to columns and overflow during expression evaluation, see [Section 11.1.7, “Out-of-Range and Overflow Handling”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#out-of-range-and-overflow).

For information about storage requirements of the numeric data types, see [Section 11.7, “Data Type Storage Requirements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements).

For descriptions of functions that operate on numeric values, see [Section 12.6, “Numeric Functions and Operators”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#numeric-functions). The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them. For more information, see [Section 12.6.1, “Arithmetic Operators”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#arithmetic-functions).

### 11.1.1 Numeric Data Type Syntax

For integer data types, ***M*** indicates the maximum display width. The maximum display width is 255. Display width is unrelated to the range of values a type can store, as described in [Section 11.1.6, “Numeric Type Attributes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#numeric-type-attributes).

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

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

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

As of MySQL 8.0.17, the **ZEROFILL** attribute is deprecated for numeric data types; you should expect support for it to be removed in a future version of MySQL. Consider using an alternative means of producing the effect of this attribute. For example, applications could use the [**LPAD()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_lpad) function to zero-pad numbers up to the desired width, or they could store the formatted numbers in [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) columns.

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

As of MySQL 8.0.17, the **UNSIGNED** attribute is deprecated for columns of type [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types), [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types), and [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) (and any synonyms); you should expect support for it to be removed in a future version of MySQL. Consider using a simple **CHECK** constraint instead for such columns.

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

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

**Warning**

When you use subtraction between integer values where one is of type **UNSIGNED**, the result is unsigned unless the [**NO\_UNSIGNED\_SUBTRACTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_unsigned_subtraction) SQL mode is enabled. See [Section 12.11, “Cast Functions and Operators”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#cast-functions).

**[BIT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "bit-type" \o "11.1.5 Bit-Value Type - BIT)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "bit-type" \o "11.1.5 Bit-Value Type - BIT)*[)]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "bit-type" \o "11.1.5 Bit-Value Type - BIT)**

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

**[TINYINT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)**

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

**[BOOL](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)**, [**BOOLEAN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types)

These types are synonyms for [**TINYINT(1)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types). A value of zero is considered false. Nonzero values are considered true:

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

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

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

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

| false |

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

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

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

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

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

| true |

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

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

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

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

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

| true |

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

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

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

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

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

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

| true |

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

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

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

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

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

| true |

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

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

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

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

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

| false |

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

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

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

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

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

| false |

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

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

**[SMALLINT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)**

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

**[MEDIUMINT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)**

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

**[INT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)**

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

**[INTEGER[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)**

This type is a synonym for [**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types).

**[BIGINT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "integer-types" \o "11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT)**

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

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

Some things you should be aware of with respect to [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) columns:

All arithmetic is done using signed [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) or [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) values, so you should not use unsigned big integers larger than **9223372036854775807** (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) value to a [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types).

MySQL can handle [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) in the following cases:

When using integers to store large unsigned values in a [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) column.

In [**MIN(*col\_name*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_min) or [**MAX(*col\_name*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_max), where ***col\_name*** refers to a [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) column.

When using operators ([**+**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_plus), [**-**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_minus), [**\***](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_times), and so on) where both operands are integers.

You can always store an exact integer value in a [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.

The [**-**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_minus), [**+**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_plus), and [**\***](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_times) operators use [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than **9223372036854775807**.

**[DECIMAL[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[[,](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[D](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[])] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)**

A packed “exact” fixed-point number. ***M*** is the total number of digits (the precision) and ***D*** is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the **-** sign are not counted in ***M***. If ***D*** is 0, values have no decimal point or fractional part. The maximum number of digits (***M***) for [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) is 65. The maximum number of supported decimals (***D***) is 30. If ***D*** is omitted, the default is 0. If ***M*** is omitted, the default is 10. (There is also a limit on how long the text of [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) literals can be; see [Section 12.25.3, “Expression Handling”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#precision-math-expressions).)

**UNSIGNED**, if specified, disallows negative values. As of MySQL 8.0.17, the **UNSIGNED** attribute is deprecated for columns of type [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) (and any synonyms); you should expect support for it to be removed in a future version of MySQL. Consider using a simple **CHECK** constraint instead for such columns.

All basic calculations (**+, -, \*, /**) with [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) columns are done with a precision of 65 digits.

**[DEC[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[[,](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[D](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)*[])] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "fixed-point-types" \o "11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC)**, [**NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types), [**FIXED[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types)

These types are synonyms for [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types). The [**FIXED**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) synonym is available for compatibility with other database systems.

**[FLOAT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[,](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[D](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)**

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

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

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

**UNSIGNED**, if specified, disallows negative values. As of MySQL 8.0.17, the **UNSIGNED** attribute is deprecated for columns of type [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple **CHECK** constraint instead for such columns.

Using [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) might give you some unexpected problems because all calculations in MySQL are done with double precision. See [Section B.3.4.7, “Solving Problems with No Matching Rows”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\error-handling.html#no-matching-rows).

**[FLOAT(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[p](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[) [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)**

A floating-point number. ***p*** represents the precision in bits, but MySQL uses this value only to determine whether to use [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) or [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) for the resulting data type. If ***p*** is from 0 to 24, the data type becomes [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) with no ***M*** or ***D*** values. If ***p*** is from 25 to 53, the data type becomes [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) with no ***M*** or ***D*** values. The range of the resulting column is the same as for the single-precision [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) or double-precision [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) data types described earlier in this section.

**UNSIGNED**, if specified, disallows negative values. As of MySQL 8.0.17, the **UNSIGNED** attribute is deprecated for columns of type [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple **CHECK** constraint instead for such columns.

**[FLOAT(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[p](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[)](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)** syntax is provided for ODBC compatibility.

**[DOUBLE[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[,](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[D](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)**

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

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

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

**UNSIGNED**, if specified, disallows negative values. As of MySQL 8.0.17, the **UNSIGNED** attribute is deprecated for columns of type [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple **CHECK** constraint instead for such columns.

**[DOUBLE PRECISION[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[,](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[D](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)*[)] [UNSIGNED] [ZEROFILL]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "floating-point-types" \o "11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE)**, [**REAL[(*M*,*D*)] [UNSIGNED] [ZEROFILL]**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types)

These types are synonyms for [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types). Exception: If the [**REAL\_AS\_FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_real_as_float) SQL mode is enabled, [**REAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) is a synonym for [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) rather than [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types).

### 11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

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

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

| **Type** | **Storage (Bytes)** | **Minimum Value Signed** | **Minimum Value Unsigned** | **Maximum Value Signed** | **Maximum Value Unsigned** |
| --- | --- | --- | --- | --- | --- |
| **TINYINT** | 1 | **-128** | **0** | **127** | **255** |
| **SMALLINT** | 2 | **-32768** | **0** | **32767** | **65535** |
| **MEDIUMINT** | 3 | **-8388608** | **0** | **8388607** | **16777215** |
| **INT** | 4 | **-2147483648** | **0** | **2147483647** | **4294967295** |
| **BIGINT** | 8 | **-2**63 | **0** | **2**63**-1** | **2**64**-1** |

### 11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC

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

MySQL stores **DECIMAL** values in binary format. See [Section 12.25, “Precision Math”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#precision-math).

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

salary DECIMAL(5,2)

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

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

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

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

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

### 11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

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

For **FLOAT**, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword **FLOAT** in parentheses; ; that is, [**FLOAT(*p*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types). MySQL also supports this optional precision specification, but the precision value in [**FLOAT(*p*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) is used only to determine storage size. A precision from 0 to 23 results in a 4-byte single-precision **FLOAT** column. A precision from 24 to 53 results in an 8-byte double-precision **DOUBLE** column.

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

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

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. For more information, see [Section B.3.4.8, “Problems with Floating-Point Values”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\error-handling.html#problems-with-float)

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

### 11.1.5 Bit-Value Type - BIT

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

To specify bit values, **b'*value*'** notation can be used. ***value*** is a binary value written using zeros and ones. For example, **b'111'** and **b'10000000'** represent 7 and 128, respectively. See [Section 9.1.5, “Bit-Value Literals”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#bit-value-literals).

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

**NDB Cluster.** The maximum combined size of all **BIT** columns used in a given [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) table must not exceed 4096 bits.

### 11.1.6 Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, [**INT(4)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) specifies an [**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used is up to the application.)

The display width does not constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as [**SMALLINT(3)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) has the usual [**SMALLINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) range of **-32768** to **32767**, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) **ZEROFILL** attribute, the default padding of spaces is replaced with zeros. For example, for a column declared as [**INT(4) ZEROFILL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), a value of **5** is retrieved as **0005**.

**Note**

The **ZEROFILL** attribute is ignored for columns involved in expressions or [**UNION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#union) queries.

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

As of MySQL 8.0.17, the **ZEROFILL** attribute is deprecated for numeric data types, as is the display width attribute for integer data types. You should expect support for **ZEROFILL** and display widths for integer data types to be removed in a future version of MySQL. Consider using an alternative means of producing the effect of these attributes. For example, applications can use the [**LPAD()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_lpad) function to zero-pad numbers up to the desired width, or they can store the formatted numbers in [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) columns.

All integer types can have an optional (nonstandard) **UNSIGNED** attribute. An unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the column. For example, if an [**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) column is **UNSIGNED**, the size of the column's range is the same but its endpoints shift up, from **-2147483648** and **2147483647** to **0** and **4294967295**.

Floating-point and fixed-point types also can be **UNSIGNED**. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same. As of MySQL 8.0.17, the **UNSIGNED** attribute is deprecated for columns of type [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types), [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types), and [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) (and any synonyms) and you should expect support for it to be removed in a future version of MySQL. Consider using a simple **CHECK** constraint instead for such columns.

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

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

Storing **0** into an **AUTO\_INCREMENT** column has the same effect as storing **NULL**, unless the [**NO\_AUTO\_VALUE\_ON\_ZERO**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_auto_value_on_zero) SQL mode is enabled.

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

Negative values for **AUTO\_INCREMENT** columns are not supported.

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

As of MySQL 8.0.17, **AUTO\_INCREMENT** support is deprecated for [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) and [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) columns; you should expect it to be removed in a future version of MySQL. Consider removing the **AUTO\_INCREMENT** attribute from such columns, or convert them to an integer type.

### 11.1.7 Out-of-Range and Overflow Handling

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

If strict SQL mode is enabled, MySQL rejects the out-of-range value with an error, and the insert fails, in accordance with the SQL standard.

If no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the column data type range and stores the resulting value instead.

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

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

Suppose that a table **t1** has this definition:

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

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

mysql> **SET sql\_mode = 'TRADITIONAL';**

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

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

mysql> **SELECT \* FROM t1;**

Empty set (0.00 sec)

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

mysql> **SET sql\_mode = '';**

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

mysql> **SHOW WARNINGS;**

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

| Level | Code | Message |

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

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

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

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

mysql> **SELECT \* FROM t1;**

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

| i1 | i2 |

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

| 127 | 255 |

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

When strict SQL mode is not enabled, column-assignment conversions that occur due to clipping are reported as warnings for [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table), [**LOAD DATA**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#load-data), [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update), and multiple-row [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) statements. In strict mode, these statements fail, and some or all the values are not inserted or changed, depending on whether the table is a transactional table and other factors. For details, see [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

Overflow during numeric expression evaluation results in an error. For example, the largest signed [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) value is 9223372036854775807, so the following expression produces an error:

mysql> **SELECT 9223372036854775807 + 1;**

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

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

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

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

| CAST(9223372036854775807 AS UNSIGNED) + 1 |

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

| 9223372036854775808 |

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

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) values have a larger range than integers:

mysql> **SELECT 9223372036854775807.0 + 1;**

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

| 9223372036854775807.0 + 1 |

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

| 9223372036854775808.0 |

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

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

mysql> **SET sql\_mode = '';**

Query OK, 0 rows affected (0.00 sec)

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

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

If the [**NO\_UNSIGNED\_SUBTRACTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_unsigned_subtraction) SQL mode is enabled, the result is negative:

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

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

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

| CAST(0 AS UNSIGNED) - 1 |

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

| -1 |

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

If the result of such an operation is used to update an **UNSIGNED** integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if [**NO\_UNSIGNED\_SUBTRACTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_unsigned_subtraction) is enabled. If strict SQL mode is enabled, an error occurs and the column remains unchanged.

## 11.2 Date and Time Data Types

[11.2.1 Date and Time Data Type Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#date-and-time-type-syntax)

[11.2.2 The DATE, DATETIME, and TIMESTAMP Types](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime)

[11.2.3 The TIME Type](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time)

[11.2.4 The YEAR Type](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year)

[11.2.5 Automatic Initialization and Updating for TIMESTAMP and DATETIME](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#timestamp-initialization)

[11.2.6 Fractional Seconds in Time Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fractional-seconds)

[11.2.7 Conversion Between Date and Time Types](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#date-and-time-type-conversion)

[11.2.8 2-Digit Years in Dates](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#two-digit-years)

The date and time data types for representing temporal values are [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year). Each temporal type has a range of valid values, as well as a “zero” value that may be used when you specify an invalid value that MySQL cannot represent. The [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) types have special automatic updating behavior, described in [Section 11.2.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#timestamp-initialization).

For information about storage requirements of the temporal data types, see [Section 11.7, “Data Type Storage Requirements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements).

For descriptions of functions that operate on temporal values, see [Section 12.7, “Date and Time Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#date-and-time-functions).

Keep in mind these general considerations when working with date and time types:

MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). For a description of the permitted formats for date and time types, see [Section 9.1.3, “Date and Time Literals”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#date-and-time-literals). It is expected that you supply valid values. Unpredictable results may occur if you use values in other formats.

Although MySQL tries to interpret values in several formats, date parts must always be given in year-month-day order (for example, **'98-09-04'**), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, **'09-04-98'**, **'04-09-98'**). To convert strings in other orders to year-month-day order, the [**STR\_TO\_DATE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_str-to-date) function may be useful.

Dates containing 2-digit year values are ambiguous because the century is unknown. MySQL interprets 2-digit year values using these rules:

Year values in the range **70-99** become **1970-1999**.

Year values in the range **00-69** become **2000-2069**.

See also [Section 11.2.8, “2-Digit Years in Dates”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#two-digit-years).

Conversion of values from one temporal type to another occurs according to the rules in [Section 11.2.7, “Conversion Between Date and Time Types”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#date-and-time-type-conversion).

MySQL automatically converts a date or time value to a number if the value is used in numeric context and vice versa.

By default, when MySQL encounters a value for a date or time type that is out of range or otherwise invalid for the type, it converts the value to the “zero” value for that type. The exception is that out-of-range [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) values are clipped to the appropriate endpoint of the [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) range.

By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).) You can get MySQL to accept certain dates, such as **'2009-11-31'**, by enabling the [**ALLOW\_INVALID\_DATES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_allow_invalid_dates) SQL mode. This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 1 to 12 and that the day is in the range from 1 to 31.

MySQL permits you to store dates where the day or month and day are zero in a [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column. This is useful for applications that need to store birthdates for which you may not know the exact date. In this case, you simply store the date as **'2009-00-00'** or **'2009-01-00'**. However, with dates such as these, you should not expect to get correct results for functions such as [**DATE\_SUB()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_date-sub) or [**DATE\_ADD()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_date-add) that require complete dates. To disallow zero month or day parts in dates, enable the [**NO\_ZERO\_IN\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_zero_in_date) mode.

MySQL permits you to store a “zero” value of **'0000-00-00'** as a “dummy date.” In some cases, this is more convenient than using **NULL** values, and uses less data and index space. To disallow **'0000-00-00'**, enable the [**NO\_ZERO\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_zero_date) mode.

“Zero” date or time values used through Connector/ODBC are converted automatically to **NULL** because ODBC cannot handle such values.

The following table shows the format of the “zero” value for each type. The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values **'0'** or **0**, which are easier to write. For temporal types that include a date part ([**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime)), use of these values may produce warning or errors. The precise behavior depends on which, if any, of the strict and [**NO\_ZERO\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_zero_date) SQL modes are enabled; see [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

| **Data Type** | **“Zero” Value** |
| --- | --- |
| [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) | **'0000-00-00'** |
| [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) | **'00:00:00'** |
| [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) | **'0000-00-00 00:00:00'** |
| [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) | **'0000-00-00 00:00:00'** |
| [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) | **0000** |

### 11.2.1 Date and Time Data Type Syntax

The date and time data types for representing temporal values are [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year).

For the [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) range descriptions, “supported” means that although earlier values might work, there is no guarantee.

MySQL permits fractional seconds for [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values, with up to microseconds (6 digits) precision. To define a column that includes a fractional seconds part, use the syntax ***type\_name*(*fsp*)**, where ***type\_name*** is [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), or [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and ***fsp*** is the fractional seconds precision. For example:

CREATE TABLE t1 (t TIME(3), dt DATETIME(6), ts TIMESTAMP(0));

The ***fsp*** value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

Any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column in a table can have automatic initialization and updating properties; see [Section 11.2.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#timestamp-initialization).

**[DATE](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)**

A date. The supported range is **'1000-01-01'** to **'9999-12-31'**. MySQL displays [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values in **'*YYYY-MM-DD*'** format, but permits assignment of values to [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns using either strings or numbers.

**[DATETIME[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)*[fsp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)*[)]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)**

A date and time combination. The supported range is **'1000-01-01 00:00:00.000000'** to **'9999-12-31 23:59:59.999999'**. MySQL displays [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values in **'*YYYY-MM-DD hh:mm:ss*[.*fraction*]'** format, but permits assignment of values to [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns using either strings or numbers.

An optional ***fsp*** value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

Automatic initialization and updating to the current date and time for [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns can be specified using **DEFAULT** and **ON UPDATE** column definition clauses, as described in [Section 11.2.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#timestamp-initialization).

**[TIMESTAMP[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)*[fsp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)*[)]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)**

A timestamp. The range is **'1970-01-01 00:00:01.000000'** UTC to **'2038-01-19 03:14:07.999999'** UTC. [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values are stored as the number of seconds since the epoch (**'1970-01-01 00:00:00'** UTC). A [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) cannot represent the value **'1970-01-01 00:00:00'** because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing **'0000-00-00 00:00:00'**, the “zero” [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value.

An optional ***fsp*** value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

The way the server handles **TIMESTAMP** definitions depends on the value of the **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** system variable (see [Section 5.1.8, “Server System Variables”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#server-system-variables)).

If **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** is enabled, there is no automatic assignment of the **DEFAULT CURRENT\_TIMESTAMP** or **ON UPDATE CURRENT\_TIMESTAMP** attributes to any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column. They must be included explicitly in the column definition. Also, any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) not explicitly declared as **NOT NULL** permits **NULL** values.

If **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** is disabled, the server handles **TIMESTAMP** as follows:

Unless specified otherwise, the first [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column in a table is defined to be automatically set to the date and time of the most recent modification if not explicitly assigned a value. This makes [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) useful for recording the timestamp of an [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) or [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) operation. You can also set any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column to the current date and time by assigning it a **NULL** value, unless it has been defined with the **NULL** attribute to permit **NULL** values.

Automatic initialization and updating to the current date and time can be specified using **DEFAULT CURRENT\_TIMESTAMP** and **ON UPDATE CURRENT\_TIMESTAMP** column definition clauses. By default, the first [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column has these properties, as previously noted. However, any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column in a table can be defined to have these properties.

**[TIME[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "time" \o "11.2.3 The TIME Type)*[fsp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "time" \o "11.2.3 The TIME Type)*[)]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "time" \o "11.2.3 The TIME Type)**

A time. The range is **'-838:59:59.000000'** to **'838:59:59.000000'**. MySQL displays [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) values in **'*hh:mm:ss*[.*fraction*]'** format, but permits assignment of values to [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) columns using either strings or numbers.

An optional ***fsp*** value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

**[YEAR[(4)]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "year" \o "11.2.4 The YEAR Type)**

A year in 4-digit format. MySQL displays [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) values in ***YYYY*** format, but permits assignment of values to [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) columns using either strings or numbers. Values display as **1901** to **2155**, or **0000**.

For additional information about [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) display format and interpretation of input values, see [Section 11.2.4, “The YEAR Type”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year).

**Note**

As of MySQL 8.0.19, the [**YEAR(4)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) data type with an explicit display width is deprecated; you should expect support for it to be removed in a future version of MySQL. Instead, use [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) without a display width, which has the same meaning.

MySQL 8.0 does not support the 2-digit [**YEAR(2)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) data type permitted in older versions of MySQL. For instructions on converting to 4-digit [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year), see [2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR](https://dev.mysql.com/doc/refman/5.7/en/migrating-from-year2.html), in [MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

The [**SUM()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_sum) and [**AVG()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_avg) aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

SELECT SEC\_TO\_TIME(SUM(TIME\_TO\_SEC(***time\_col***))) FROM ***tbl\_name***;

SELECT FROM\_DAYS(SUM(TO\_DAYS(***date\_col***))) FROM ***tbl\_name***;

### 11.2.2 The DATE, DATETIME, and TIMESTAMP Types

The **DATE**, **DATETIME**, and **TIMESTAMP** types are related. This section describes their characteristics, how they are similar, and how they differ. MySQL recognizes **DATE**, **DATETIME**, and **TIMESTAMP** values in several formats, described in [Section 9.1.3, “Date and Time Literals”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#date-and-time-literals). For the **DATE** and **DATETIME** range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The **DATE** type is used for values with a date part but no time part. MySQL retrieves and displays **DATE** values in **'*YYYY-MM-DD*'** format. The supported range is **'1000-01-01'** to **'9999-12-31'**.

The **DATETIME** type is used for values that contain both date and time parts. MySQL retrieves and displays **DATETIME** values in **'*YYYY-MM-DD hh:mm:ss*'** format. The supported range is **'1000-01-01 00:00:00'** to **'9999-12-31 23:59:59'**.

The **TIMESTAMP** data type is used for values that contain both date and time parts. **TIMESTAMP** has a range of **'1970-01-01 00:00:01'** UTC to **'2038-01-19 03:14:07'** UTC.

A **DATETIME** or **TIMESTAMP** value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. In particular, any fractional part in a value inserted into a **DATETIME** or **TIMESTAMP** column is stored rather than discarded. With the fractional part included, the format for these values is **'*YYYY-MM-DD hh:mm:ss*[.*fraction*]'**, the range for **DATETIME** values is **'1000-01-01 00:00:00.000000'** to **'9999-12-31 23:59:59.999999'**, and the range for **TIMESTAMP** values is **'1970-01-01 00:00:01.000000'** to **'2038-01-19 03:14:07.999999'**. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see [Section 11.2.6, “Fractional Seconds in Time Values”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fractional-seconds).

The **TIMESTAMP** and **DATETIME** data types offer automatic initialization and updating to the current date and time. For more information, see [Section 11.2.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#timestamp-initialization).

MySQL converts **TIMESTAMP** values from the current time zone to UTC for storage, and back from UTC to the current time zone for retrieval. (This does not occur for other types such as **DATETIME**.) By default, the current time zone for each connection is the server's time. The time zone can be set on a per-connection basis. As long as the time zone setting remains constant, you get back the same value you store. If you store a **TIMESTAMP** value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the **[time\_zone](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_time_zone)** system variable. For more information, see [Section 5.1.15, “MySQL Server Time Zone Support”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#time-zone-support).

As of MySQL 8.0.19, you can specify a time zone offset when inserting **TIMESTAMP** and **DATETIME** values into a table. The offset is appended to the time part of a datetime literal, with no intravening spaces, and uses the same format used for setting the **[time\_zone](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_time_zone)** system variable, with the following exceptions:

For hour values less than than 10, a leading zero is required.

The value **'-00:00'** is rejected.

Time zone names such as **'EET'** and **'Asia/Shanghai'** cannot be used; **'SYSTEM'** also cannot be used in this context.

The value inserted must not have a zero for the month part, the day part, or both parts. This is enforced beginning with MySQL 8.0.22, regardless of the server SQL mode setting.

This example illustrates inserting datetime values with time zone offsets into **TIMESTAMP** and **DATETIME** columns using different **[time\_zone](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_time_zone)** settings, and then retrieving them:

mysql> **CREATE TABLE ts (**

-> **id INTEGER NOT NULL AUTO\_INCREMENT PRIMARY KEY,**

-> **col TIMESTAMP NOT NULL**

-> **) AUTO\_INCREMENT = 1;**

mysql> **CREATE TABLE dt (**

-> **id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,**

-> **col DATETIME NOT NULL**

-> **) AUTO\_INCREMENT = 1;**

mysql> **SET @@time\_zone = 'SYSTEM';**

mysql> **INSERT INTO ts (col) VALUES ('2020-01-01 10:10:10'),**

-> **('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');**

mysql> **SET @@time\_zone = '+00:00';**

mysql> **INSERT INTO ts (col) VALUES ('2020-01-01 10:10:10'),**

-> **('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');**

mysql> **SET @@time\_zone = 'SYSTEM';**

mysql> **INSERT INTO dt (col) VALUES ('2020-01-01 10:10:10'),**

-> **('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');**

mysql> **SET @@time\_zone = '+00:00';**

mysql> **INSERT INTO dt (col) VALUES ('2020-01-01 10:10:10'),**

-> **('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');**

mysql> **SET @@time\_zone = 'SYSTEM';**

mysql> **SELECT @@system\_time\_zone;**

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

| @@system\_time\_zone |

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

| EST |

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

mysql> **SELECT col, UNIX\_TIMESTAMP(col) FROM dt ORDER BY id;**

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

| col | UNIX\_TIMESTAMP(col) |

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

| 2020-01-01 10:10:10 | 1577891410 |

| 2019-12-31 23:40:10 | 1577853610 |

| 2020-01-01 13:10:10 | 1577902210 |

| 2020-01-01 10:10:10 | 1577891410 |

| 2020-01-01 04:40:10 | 1577871610 |

| 2020-01-01 18:10:10 | 1577920210 |

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

mysql> **SELECT col, UNIX\_TIMESTAMP(col) FROM ts ORDER BY id;**

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

| col | UNIX\_TIMESTAMP(col) |

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

| 2020-01-01 10:10:10 | 1577891410 |

| 2019-12-31 23:40:10 | 1577853610 |

| 2020-01-01 13:10:10 | 1577902210 |

| 2020-01-01 05:10:10 | 1577873410 |

| 2019-12-31 23:40:10 | 1577853610 |

| 2020-01-01 13:10:10 | 1577902210 |

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

The offset is not displayed when selecting a datetime value, even if one was used when inserting it.

The range of supported offset values is **-14:00** to **+14:00**, inclusive.

Datetime literals that include time zone offsets are accepted as parameter values by prepared statements.

Invalid **DATE**, **DATETIME**, or **TIMESTAMP** values are converted to the “zero” value of the appropriate type (**'0000-00-00'** or **'0000-00-00 00:00:00'**), if the SQL mode permits this conversion. The precise behavior depends on which if any of strict SQL mode and the [**NO\_ZERO\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_zero_date) SQL mode are enabled; see [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

In MySQL 8.0.22 and later, you can convert **TIMESTAMP** values to UTC **DATETIME** values when retrieving them using [**CAST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_cast) with the **AT TIME ZONE** operator, as shown here:

mysql> **SELECT col,**

> **CAST(col AT TIME ZONE INTERVAL '+00:00' AS DATETIME) AS ut**

> **FROM ts ORDER BY id;**

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

| col | ut |

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

| 2020-01-01 10:10:10 | 2020-01-01 15:10:10 |

| 2019-12-31 23:40:10 | 2020-01-01 04:40:10 |

| 2020-01-01 13:10:10 | 2020-01-01 18:10:10 |

| 2020-01-01 10:10:10 | 2020-01-01 15:10:10 |

| 2020-01-01 04:40:10 | 2020-01-01 09:40:10 |

| 2020-01-01 18:10:10 | 2020-01-01 23:10:10 |

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

For complete information regarding syntax and additional examples, see the description of the [**CAST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_cast) function.

Be aware of certain properties of date value interpretation in MySQL:

MySQL permits a “relaxed” format for values specified as strings, in which any punctuation character may be used as the delimiter between date parts or time parts. In some cases, this syntax can be deceiving. For example, a value such as **'10:11:12'** might look like a time value because of the **:**, but is interpreted as the year **'2010-11-12'** if used in date context. The value **'10:45:15'** is converted to **'0000-00-00'** because **'45'** is not a valid month.

The only delimiter recognized between a date and time part and a fractional seconds part is the decimal point.

The server requires that month and day values be valid, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as **'2004-04-31'** are converted to **'0000-00-00'** and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable [**ALLOW\_INVALID\_DATES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_allow_invalid_dates). See [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode), for more information.

MySQL does not accept **TIMESTAMP** values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special “zero” value **'0000-00-00 00:00:00'**, if the SQL mode permits this value. The precise behavior depends on which if any of strict SQL mode and the [**NO\_ZERO\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_zero_date) SQL mode are enabled; see [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

Dates containing 2-digit year values are ambiguous because the century is unknown. MySQL interprets 2-digit year values using these rules:

Year values in the range **00-69** become **2000-2069**.

Year values in the range **70-99** become **1970-1999**.

See also [Section 11.2.8, “2-Digit Years in Dates”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#two-digit-years).

### 11.2.3 The TIME Type

MySQL retrieves and displays **TIME** values in ***'hh:mm:ss'*** format (or ***'hhh:mm:ss'*** format for large hours values). **TIME** values may range from **'-838:59:59'** to **'838:59:59'**. The hours part may be so large because the **TIME** type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

MySQL recognizes **TIME** values in several formats, some of which can include a trailing fractional seconds part in up to microseconds (6 digits) precision. See [Section 9.1.3, “Date and Time Literals”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#date-and-time-literals). For information about fractional seconds support in MySQL, see [Section 11.2.6, “Fractional Seconds in Time Values”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fractional-seconds). In particular, any fractional part in a value inserted into a **TIME** column is stored rather than discarded. With the fractional part included, the range for **TIME** values is **'-838:59:59.000000'** to **'838:59:59.000000'**.

Be careful about assigning abbreviated values to a **TIME** column. MySQL interprets abbreviated **TIME** values with colons as time of the day. That is, **'11:12'** means **'11:12:00'**, not **'00:11:12'**. MySQL interprets abbreviated values without colons using the assumption that the two rightmost digits represent seconds (that is, as elapsed time rather than as time of day). For example, you might think of **'1112'** and **1112** as meaning **'11:12:00'** (12 minutes after 11 o'clock), but MySQL interprets them as **'00:11:12'** (11 minutes, 12 seconds). Similarly, **'12'** and **12** are interpreted as **'00:00:12'**.

The only delimiter recognized between a time part and a fractional seconds part is the decimal point.

By default, values that lie outside the **TIME** range but are otherwise valid are clipped to the closest endpoint of the range. For example, **'-850:00:00'** and **'850:00:00'** are converted to **'-838:59:59'** and **'838:59:59'**. Invalid **TIME** values are converted to **'00:00:00'**. Note that because **'00:00:00'** is itself a valid **TIME** value, there is no way to tell, from a value of **'00:00:00'** stored in a table, whether the original value was specified as **'00:00:00'** or whether it was invalid.

For more restrictive treatment of invalid **TIME** values, enable strict SQL mode to cause errors to occur. See [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

### 11.2.4 The YEAR Type

The **YEAR** type is a 1-byte type used to represent year values. It can be declared as **YEAR** with an implicit display width of 4 characters, or equivalently as **YEAR(4)** with an explicit display width.

**Note**

As of MySQL 8.0.19, the [**YEAR(4)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) data type with an explicit display width is deprecated and you should expect support for it to be removed in a future version of MySQL. Instead, use [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) without a display width, which has the same meaning.

MySQL 8.0 does not support the 2-digit [**YEAR(2)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) data type permitted in older versions of MySQL. For instructions on converting to 4-digit [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year), see [2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR](https://dev.mysql.com/doc/refman/5.7/en/migrating-from-year2.html), in [MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

MySQL displays **YEAR** values in ***YYYY*** format, with a range of **1901** to **2155**, and **0000**.

**YEAR** accepts input values in a variety of formats:

As 4-digit strings in the range **'1901'** to **'2155'**.

As 4-digit numbers in the range **1901** to **2155**.

As 1- or 2-digit strings in the range **'0'** to **'99'**. MySQL converts values in the ranges **'0'** to **'69'** and **'70'** to **'99'** to **YEAR** values in the ranges **2000** to **2069** and **1970** to **1999**.

As 1- or 2-digit numbers in the range **0** to **99**. MySQL converts values in the ranges **1** to **69** and **70** to **99** to **YEAR** values in the ranges **2001** to **2069** and **1970** to **1999**.

The result of inserting a numeric **0** has a display value of **0000** and an internal value of **0000**. To insert zero and have it be interpreted as **2000**, specify it as a string **'0'** or **'00'**.

As the result of functions that return a value that is acceptable in **YEAR** context, such as [**NOW()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_now).

If strict SQL mode is not enabled, MySQL converts invalid **YEAR** values to **0000**. In strict SQL mode, attempting to insert an invalid **YEAR** value produces an error.

See also [Section 11.2.8, “2-Digit Years in Dates”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#two-digit-years).

### 11.2.5 Automatic Initialization and Updating for TIMESTAMP and DATETIME

**[TIMESTAMP](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "datetime" \o "11.2.2 The DATE, DATETIME, and TIMESTAMP Types)** and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns can be automatically initializated and updated to the current date and time (that is, the current timestamp).

For any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column in a table, you can assign the current timestamp as the default value, the auto-update value, or both:

An auto-initialized column is set to the current timestamp for inserted rows that specify no value for the column.

An auto-updated column is automatically updated to the current timestamp when the value of any other column in the row is changed from its current value. An auto-updated column remains unchanged if all other columns are set to their current values. To prevent an auto-updated column from updating when other columns change, explicitly set it to its current value. To update an auto-updated column even when other columns do not change, explicitly set it to the value it should have (for example, set it to [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp)).

In addition, if the **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** system variable is disabled, you can initialize or update any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) (but not **DATETIME**) column to the current date and time by assigning it a **NULL** value, unless it has been defined with the **NULL** attribute to permit **NULL** values.

To specify automatic properties, use the **DEFAULT CURRENT\_TIMESTAMP** and **ON UPDATE CURRENT\_TIMESTAMP** clauses in column definitions. The order of the clauses does not matter. If both are present in a column definition, either can occur first. Any of the synonyms for [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp) have the same meaning as [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp). These are [**CURRENT\_TIMESTAMP()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp), [**NOW()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_now), [**LOCALTIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_localtime), [**LOCALTIME()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_localtime), [**LOCALTIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_localtimestamp), and [**LOCALTIMESTAMP()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_localtimestamp).

Use of **DEFAULT CURRENT\_TIMESTAMP** and **ON UPDATE CURRENT\_TIMESTAMP** is specific to [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime). The **DEFAULT** clause also can be used to specify a constant (nonautomatic) default value (for example, **DEFAULT 0** or **DEFAULT '2000-01-01 00:00:00'**).

**Note**

The following examples use **DEFAULT 0**, a default that can produce warnings or errors depending on whether strict SQL mode or the [**NO\_ZERO\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_zero_date) SQL mode is enabled. Be aware that the [**TRADITIONAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_traditional) SQL mode includes strict mode and [**NO\_ZERO\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_zero_date). See [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

[**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column definitions can specify the current timestamp for both the default and auto-update values, for one but not the other, or for neither. Different columns can have different combinations of automatic properties. The following rules describe the possibilities:

With both **DEFAULT CURRENT\_TIMESTAMP** and **ON UPDATE CURRENT\_TIMESTAMP**, the column has the current timestamp for its default value and is automatically updated to the current timestamp.

CREATE TABLE t1 (

ts TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

dt DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

With a **DEFAULT** clause but no **ON UPDATE CURRENT\_TIMESTAMP** clause, the column has the given default value and is not automatically updated to the current timestamp.

The default depends on whether the **DEFAULT** clause specifies **CURRENT\_TIMESTAMP** or a constant value. With **CURRENT\_TIMESTAMP**, the default is the current timestamp.

CREATE TABLE t1 (

ts TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

dt DATETIME DEFAULT CURRENT\_TIMESTAMP

);

With a constant, the default is the given value. In this case, the column has no automatic properties at all.

CREATE TABLE t1 (

ts TIMESTAMP DEFAULT 0,

dt DATETIME DEFAULT 0

);

With an **ON UPDATE CURRENT\_TIMESTAMP** clause and a constant **DEFAULT** clause, the column is automatically updated to the current timestamp and has the given constant default value.

CREATE TABLE t1 (

ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT\_TIMESTAMP,

dt DATETIME DEFAULT 0 ON UPDATE CURRENT\_TIMESTAMP

);

With an **ON UPDATE CURRENT\_TIMESTAMP** clause but no **DEFAULT** clause, the column is automatically updated to the current timestamp but does not have the current timestamp for its default value.

The default in this case is type dependent. [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) has a default of 0 unless defined with the **NULL** attribute, in which case the default is **NULL**.

CREATE TABLE t1 (

ts1 TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP, -- default 0

ts2 TIMESTAMP NULL ON UPDATE CURRENT\_TIMESTAMP -- default NULL

);

[**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) has a default of **NULL** unless defined with the **NOT NULL** attribute, in which case the default is 0.

CREATE TABLE t1 (

dt1 DATETIME ON UPDATE CURRENT\_TIMESTAMP, -- default NULL

dt2 DATETIME NOT NULL ON UPDATE CURRENT\_TIMESTAMP -- default 0

);

[**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns have no automatic properties unless they are specified explicitly, with this exception: If the **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** system variable is disabled, the first [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column has both **DEFAULT CURRENT\_TIMESTAMP** and **ON UPDATE CURRENT\_TIMESTAMP** if neither is specified explicitly. To suppress automatic properties for the first [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column, use one of these strategies:

Enable the **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** system variable. In this case, the **DEFAULT CURRENT\_TIMESTAMP** and **ON UPDATE CURRENT\_TIMESTAMP** clauses that specify automatic initialization and updating are available, but are not assigned to any [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column unless explicitly included in the column definition.

Alternatively, if **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** is disabled, do either of the following:

Define the column with a **DEFAULT** clause that specifies a constant default value.

Specify the **NULL** attribute. This also causes the column to permit **NULL** values, which means that you cannot assign the current timestamp by setting the column to **NULL**. Assigning **NULL** sets the column to **NULL**, not the current timestamp. To assign the current timestamp, set the column to [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp) or a synonym such as [**NOW()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_now).

Consider these table definitions:

CREATE TABLE t1 (

ts1 TIMESTAMP DEFAULT 0,

ts2 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP);

CREATE TABLE t2 (

ts1 TIMESTAMP NULL,

ts2 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP);

CREATE TABLE t3 (

ts1 TIMESTAMP NULL DEFAULT 0,

ts2 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP);

The tables have these properties:

In each table definition, the first [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column has no automatic initialization or updating.

The tables differ in how the **ts1** column handles **NULL** values. For **t1**, **ts1** is **NOT NULL** and assigning it a value of **NULL** sets it to the current timestamp. For **t2** and **t3**, **ts1** permits **NULL** and assigning it a value of **NULL** sets it to **NULL**.

**t2** and **t3** differ in the default value for **ts1**. For **t2**, **ts1** is defined to permit **NULL**, so the default is also **NULL** in the absence of an explicit **DEFAULT** clause. For **t3**, **ts1** permits **NULL** but has an explicit default of 0.

If a [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column definition includes an explicit fractional seconds precision value anywhere, the same value must be used throughout the column definition. This is permitted:

CREATE TABLE t1 (

ts TIMESTAMP(6) DEFAULT CURRENT\_TIMESTAMP(6) ON UPDATE CURRENT\_TIMESTAMP(6)

);

This is not permitted:

CREATE TABLE t1 (

ts TIMESTAMP(6) DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP(3)

);

#### TIMESTAMP Initialization and the NULL Attribute

If the **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** system variable is disabled, [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns by default are **NOT NULL**, cannot contain **NULL** values, and assigning **NULL** assigns the current timestamp. To permit a [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column to contain **NULL**, explicitly declare it with the **NULL** attribute. In this case, the default value also becomes **NULL** unless overridden with a **DEFAULT** clause that specifies a different default value. **DEFAULT NULL** can be used to explicitly specify **NULL** as the default value. (For a [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column not declared with the **NULL** attribute, **DEFAULT NULL** is invalid.) If a [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column permits **NULL** values, assigning **NULL** sets it to **NULL**, not to the current timestamp.

The following table contains several [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns that permit **NULL** values:

CREATE TABLE t

(

ts1 TIMESTAMP NULL DEFAULT NULL,

ts2 TIMESTAMP NULL DEFAULT 0,

ts3 TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP

);

A [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column that permits **NULL** values does not take on the current timestamp at insert time except under one of the following conditions:

Its default value is defined as [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp) and no value is specified for the column

[**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp) or any of its synonyms such as [**NOW()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_now) is explicitly inserted into the column

In other words, a [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column defined to permit **NULL** values auto-initializes only if its definition includes **DEFAULT CURRENT\_TIMESTAMP**:

CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP);

If the [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column permits **NULL** values but its definition does not include **DEFAULT CURRENT\_TIMESTAMP**, you must explicitly insert a value corresponding to the current date and time. Suppose that tables **t1** and **t2** have these definitions:

CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');

CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT NULL);

To set the [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column in either table to the current timestamp at insert time, explicitly assign it that value. For example:

INSERT INTO t2 VALUES (CURRENT\_TIMESTAMP);

INSERT INTO t1 VALUES (NOW());

If the **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** system variable is enabled, [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns permit **NULL** values only if declared with the **NULL** attribute. Also, [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns do not permit assigning **NULL** to assign the current timestamp, whether declared with the **NULL** or **NOT NULL** attribute. To assign the current timestamp, set the column to [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp) or a synonym such as [**NOW()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_now).

### 11.2.6 Fractional Seconds in Time Values

MySQL has fractional seconds support for [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values, with up to microseconds (6 digits) precision:

To define a column that includes a fractional seconds part, use the syntax ***type\_name*(*fsp*)**, where ***type\_name*** is [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), or [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and ***fsp*** is the fractional seconds precision. For example:

CREATE TABLE t1 (t TIME(3), dt DATETIME(6));

The ***fsp*** value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

Inserting a [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), or [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value with a fractional seconds part into a column of the same type but having fewer fractional digits results in rounding. Consider a table created and populated as follows:

CREATE TABLE fractest( c1 TIME(2), c2 DATETIME(2), c3 TIMESTAMP(2) );

INSERT INTO fractest VALUES

('17:51:04.777', '2018-09-08 17:51:04.777', '2018-09-08 17:51:04.777');

The temporal values are inserted into the table with rounding:

mysql> **SELECT \* FROM fractest;**

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

| c1 | c2 | c3 |

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

| 17:51:04.78 | 2018-09-08 17:51:04.78 | 2018-09-08 17:51:04.78 |

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

No warning or error is given when such rounding occurs. This behavior follows the SQL standard.

To insert the values with truncation instead, enable the [**TIME\_TRUNCATE\_FRACTIONAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_time_truncate_fractional) SQL mode:

SET @@sql\_mode = sys.list\_add(@@sql\_mode, 'TIME\_TRUNCATE\_FRACTIONAL');

With that SQL mode enabled, the temporal values are inserted with truncation:

mysql> **SELECT \* FROM fractest;**

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

| c1 | c2 | c3 |

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

| 17:51:04.77 | 2018-09-08 17:51:04.77 | 2018-09-08 17:51:04.77 |

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

Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate. For example, [**NOW()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_now) with no argument returns the current date and time with no fractional part, but takes an optional argument from 0 to 6 to specify that the return value includes a fractional seconds part of that many digits.

Syntax for temporal literals produces temporal values: **DATE '*str*'**, **TIME '*str*'**, and **TIMESTAMP '*str*'**, and the ODBC-syntax equivalents. The resulting value includes a trailing fractional seconds part if specified. Previously, the temporal type keyword was ignored and these constructs produced the string value. See [Standard SQL and ODBC Date and Time Literals](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#date-and-time-standard-sql-literals)

### 11.2.7 Conversion Between Date and Time Types

To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information. In all cases, conversion between temporal types is subject to the range of valid values for the resulting type. For example, although [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values all can be specified using the same set of formats, the types do not all have the same range of values. [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values cannot be earlier than **1970** UTC or later than **'2038-01-19 03:14:07'** UTC. This means that a date such as **'1968-01-01'**, while valid as a [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value, is not valid as a [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value and is converted to **0**.

Conversion of [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values:

Conversion to a [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value adds a time part of **'00:00:00'** because the [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value contains no time information.

Conversion to a [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) value is not useful; the result is **'00:00:00'**.

Conversion of [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values:

Conversion to a [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value takes fractional seconds into account and rounds the time part. For example, **'1999-12-31 23:59:59.499'** becomes **'1999-12-31'**, whereas **'1999-12-31 23:59:59.500'** becomes **'2000-01-01'**.

Conversion to a [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) value discards the date part because the [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) type contains no date information.

For conversion of [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) values to other temporal types, the value of [**CURRENT\_DATE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-date) is used for the date part. The [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) is interpreted as elapsed time (not time of day) and added to the date. This means that the date part of the result differs from the current date if the time value is outside the range from **'00:00:00'** to **'23:59:59'**.

Suppose that the current date is **'2012-01-01'**. [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) values of **'12:00:00'**, **'24:00:00'**, and **'-12:00:00'**, when converted to [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) or [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values, result in **'2012-01-01 12:00:00'**, **'2012-01-02 00:00:00'**, and **'2011-12-31 12:00:00'**, respectively.

Conversion of [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) to [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) is similar but discards the time part from the result: **'2012-01-01'**, **'2012-01-02'**, and **'2011-12-31'**, respectively.

Explicit conversion can be used to override implicit conversion. For example, in comparison of [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values, the [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value is coerced to the [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) type by adding a time part of **'00:00:00'**. To perform the comparison by ignoring the time part of the [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) value instead, use the [**CAST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_cast) function in the following way:

***date\_col*** = CAST(***datetime\_col*** AS DATE)

Conversion of [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) values to numeric form (for example, by adding **+0**) depends on whether the value contains a fractional seconds part. [**TIME(*N*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) or [**DATETIME(*N*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) is converted to integer when ***N*** is 0 (or omitted) and to a **DECIMAL** value with ***N*** decimal digits when ***N*** is greater than 0:

mysql> **SELECT CURTIME(), CURTIME()+0, CURTIME(3)+0;**

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

| CURTIME() | CURTIME()+0 | CURTIME(3)+0 |

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

| 09:28:00 | 92800 | 92800.887 |

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

mysql> **SELECT NOW(), NOW()+0, NOW(3)+0;**

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

| NOW() | NOW()+0 | NOW(3)+0 |

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

| 2012-08-15 09:28:00 | 20120815092800 | 20120815092800.889 |

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

### 11.2.8 2-Digit Years in Dates

Date values with 2-digit years are ambiguous because the century is unknown. Such values must be interpreted into 4-digit form because MySQL stores years internally using 4 digits.

For [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) types, MySQL interprets dates specified with ambiguous year values using these rules:

Year values in the range **00-69** become **2000-2069**.

Year values in the range **70-99** become **1970-1999**.

For **YEAR**, the rules are the same, with this exception: A numeric **00** inserted into **YEAR** results in **0000** rather than **2000**. To specify zero for **YEAR** and have it be interpreted as **2000**, specify it as a string **'0'** or **'00'**.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the values you require, you must provide unambiguous input containing 4-digit year values.

**ORDER BY** properly sorts [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) values that have 2-digit years.

Some functions like [**MIN()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_min) and [**MAX()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_max) convert a [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) to a number. This means that a value with a 2-digit year does not work properly with these functions. The fix in this case is to convert the [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) to 4-digit year format.

## 11.3 String Data Types

[11.3.1 String Data Type Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#string-type-syntax)

[11.3.2 The CHAR and VARCHAR Types](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char)

[11.3.3 The BINARY and VARBINARY Types](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary)

[11.3.4 The BLOB and TEXT Types](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob)

[11.3.5 The ENUM Type](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum)

[11.3.6 The SET Type](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set)

The string data types are [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**BINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum), and [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set).

For information about storage requirements of the string data types, see [Section 11.7, “Data Type Storage Requirements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements).

For descriptions of functions that operate on string values, see [Section 12.8, “String Functions and Operators”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#string-functions).

### 11.3.1 String Data Type Syntax

The string data types are [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**BINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum), and [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set).

In some cases, MySQL may change a string column to a type different from that given in a [**CREATE TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table) or [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table) statement. See [Section 13.1.20.7, “Silent Column Specification Changes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#silent-column-changes).

For definitions of character string columns ([**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), and the [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) types), MySQL interprets length specifications in character units. For definitions of binary string columns ([**BINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), and the [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) types), MySQL interprets length specifications in byte units.

Column definitions for character string data types [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), the [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) types, [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum), [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set), 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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-column).

**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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) becomes [**BINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) becomes [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), and [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) becomes [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob). For the [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum) and [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum), and [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) 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*](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) values are retrieved unless the [**PAD\_CHAR\_TO\_FULL\_LENGTH**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_pad_char_to_full_length) SQL mode is enabled.

[**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) is shorthand for [**CHARACTER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char). [**NATIONAL CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) (or its equivalent short form, [**NCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char)) is the standard SQL way to define that a [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) column should use some predefined character set. MySQL uses **utf8** as this predefined character set. [Section 10.3.7, “The National Character Set”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-national).

The [**CHAR BYTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary) data type is an alias for the [**BINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) column that uses the **utf8** character set can be declared to be a maximum of 21,844 characters. See [Section 8.4.7, “Limits on Table Column Count and Row Size”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#column-count-limit).

MySQL stores [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) values.

[**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) is shorthand for [**CHARACTER VARYING**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char). [**NATIONAL VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) is the standard SQL way to define that a [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) column should use some predefined character set. MySQL uses **utf8** as this predefined character set. [Section 10.3.7, “The National Character Set”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-national). [**NVARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) is shorthand for [**NATIONAL VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char).

**[BINARY[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "binary-varbinary" \o "11.3.3 The BINARY and VARBINARY Types)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "binary-varbinary" \o "11.3.3 The BINARY and VARBINARY Types)*[)]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "binary-varbinary" \o "11.3.3 The BINARY and VARBINARY Types)**

The [**BINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary) type is similar to the [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) 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(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "binary-varbinary" \o "11.3.3 The BINARY and VARBINARY Types)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "binary-varbinary" \o "11.3.3 The BINARY and VARBINARY Types)*[)](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "binary-varbinary" \o "11.3.3 The BINARY and VARBINARY Types)**

The [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary) type is similar to the [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) type, but stores binary byte strings rather than nonbinary character strings. ***M*** represents the maximum column length in bytes.

**[TINYBLOB](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 255 (28 − 1) bytes. Each [**TINYBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

**[TINYTEXT [CHARACTER SET](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[charset\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[] [COLLATE](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[collation\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 255 (28 − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each [**TINYTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

**[BLOB[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[)]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 65,535 (216 − 1) bytes. Each [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) type large enough to hold values ***M*** bytes long.

**[TEXT[(](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[M](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[)] [CHARACTER SET](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[charset\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[] [COLLATE](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[collation\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 65,535 (216 − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) type large enough to hold values ***M*** characters long.

**[MEDIUMBLOB](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 16,777,215 (224 − 1) bytes. Each [**MEDIUMBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

**[MEDIUMTEXT [CHARACTER SET](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[charset\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[] [COLLATE](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[collation\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 16,777,215 (224 − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each [**MEDIUMTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

**[LONGBLOB](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 4,294,967,295 or 4GB (232 − 1) bytes. The effective maximum length of [**LONGBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) columns depends on the configured maximum packet size in the client/server protocol and available memory. Each [**LONGBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

**[LONGTEXT [CHARACTER SET](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[charset\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[] [COLLATE](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[collation\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)*[]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "blob" \o "11.3.4 The BLOB and TEXT Types)**

A [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column with a maximum length of 4,294,967,295 or 4GB (232 − 1) characters. The effective maximum length is less if the value contains multibyte characters. The effective maximum length of [**LONGTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each [**LONGTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

**[ENUM('](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[value1](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[','](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[value2](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[',...) [CHARACTER SET](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[charset\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[] [COLLATE](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[collation\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \o "11.3.5 The ENUM Type)*[]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "enum" \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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum) values are represented internally as integers.

An [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum) 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('](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[value1](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[','](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[value2](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[',...) [CHARACTER SET](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[charset\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[] [COLLATE](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[collation\_name](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \o "11.3.6 The SET Type)*[]](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "set" \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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) values are represented internally as integers.

A [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) 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.

### 11.3.2 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#column-count-limit).

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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-unicode-sets)).

To determine the pad attribute for a collation, use the **INFORMATION\_SCHEMA** [**COLLATIONS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-collations-table) 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](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.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*](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html). For additional information about storage requirements, see [Section 11.7, “Data Type Storage Requirements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements).

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.

### 11.3.3 The BINARY and VARBINARY Types

The **BINARY** and **VARBINARY** types are similar to [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) and [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) and [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), 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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-binary-collations).

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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) data types instead.

### 11.3.4 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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements).

**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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

Truncation of excess trailing spaces from values to be inserted into [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary) column that can be as large as you like. Similarly, you can regard a **TEXT** column as a [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) column. **BLOB** and **TEXT** differ from [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary) and [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) in the following ways:

For indexes on **BLOB** and **TEXT** columns, you must specify an index prefix length. For [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) and [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), a prefix length is optional. See [Section 8.3.5, “Column Indexes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#column-indexes).

**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](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_sort_length)** bytes of the column are used when sorting. The default value of **[max\_sort\_length](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.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](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_sort_length)** at server startup or runtime. Any client can change the value of its session **[max\_sort\_length](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#internal-temporary-tables)). 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 \***](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select), 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](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_allowed_packet)** variable, but you must do so for both the server and your client program. For example, both **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** and **[mysqldump](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysqldump" \o "4.5.4 mysqldump — A Database Backup Program)** enable you to change the client-side **[max\_allowed\_packet](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_allowed_packet)** value. See [Section 5.1.1, “Configuring the Server”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#server-configuration), [Section 4.5.1, “mysql — The MySQL Command-Line Client”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\programs.html#mysql), and [Section 4.5.4, “mysqldump — A Database Backup Program”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\programs.html#mysqldump). 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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements)

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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#string-functions). For security and other reasons, it is usually preferable to do so using application code rather than giving application users the [**FILE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_file) privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (<http://forums.mysql.com/>).

### 11.3.5 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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#string-type-syntax) for [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum) type syntax and length limits.

The [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum) 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”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements) 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](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-limits).

Using **ENUM** columns in **ORDER BY** clauses requires extra care, as explained in [Enumeration Sorting](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-sorting).

[Creating and Using ENUM Columns](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-using)

[Index Values for Enumeration Literals](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-indexes)

[Handling of Enumeration Literals](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-literals)

[Empty or NULL Enumeration Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-nulls)

[Enumeration Sorting](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-sorting)

[Enumeration Limitations](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum-limits)

#### 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum) 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()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_sum) or [**AVG()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#load-data), 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*'**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-columns) 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 Basic 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](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table) 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) or [**SMALLINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) 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](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.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.

### 11.3.6 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**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) 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](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-types-storage-reqs-strings) for storage requirements for the [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) type.

See [Section 11.3.1, “String Data Type Syntax”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#string-type-syntax) for [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) 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()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_sum) or [**AVG()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\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()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_find-in-set) function or the [**LIKE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\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 Basic Data Structures](https://dev.mysql.com/doc/c-api/8.0/en/c-api-data-structures.html).

## 11.4 Spatial Data Types

[11.4.1 Spatial Data Types](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-type-overview)

[11.4.2 The OpenGIS Geometry Model](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#opengis-geometry-model)

[11.4.3 Supported Spatial Data Formats](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-data-formats)

[11.4.4 Geometry Well-Formedness and Validity](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#geometry-well-formedness-validity)

[11.4.5 Spatial Reference System Support](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-reference-systems)

[11.4.6 Creating Spatial Columns](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#creating-spatial-columns)

[11.4.7 Populating Spatial Columns](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#populating-spatial-columns)

[11.4.8 Fetching Spatial Data](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fetching-spatial-data)

[11.4.9 Optimizing Spatial Analysis](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#optimizing-spatial-analysis)

[11.4.10 Creating Spatial Indexes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#creating-spatial-indexes)

[11.4.11 Using Spatial Indexes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#using-spatial-indexes)

The [Open Geospatial Consortium](http://www.opengeospatial.org) (OGC) is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data.

The Open Geospatial Consortium publishes the OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC website at <http://www.opengeospatial.org/standards/sfs>.

Following the OGC specification, MySQL implements spatial extensions as a subset of the ***SQL with Geometry Types*** environment. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describes a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

MySQL spatial extensions enable the generation, storage, and analysis of geographic features:

Data types for representing spatial values

Functions for manipulating spatial values

Spatial indexing for improved access times to spatial columns

The spatial data types and functions are available for **[MyISAM](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\storage-engines.html" \l "myisam-storage-engine" \o "16.2 The MyISAM Storage Engine)**, **[InnoDB](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\innodb-storage-engine.html" \o "Chapter 15 The InnoDB Storage Engine)**, [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html), and [**ARCHIVE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\storage-engines.html#archive-storage-engine) tables. For indexing spatial columns, **MyISAM** and **InnoDB** support both **SPATIAL** and non-**SPATIAL** indexes. The other storage engines support non-**SPATIAL** indexes, as described in [Section 13.1.15, “CREATE INDEX Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-index).

A ***geographic feature*** is anything in the world that has a location. A feature can be:

An entity. For example, a mountain, a pond, a city.

A space. For example, town district, the tropics.

A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term ***geospatial feature*** to refer to geographic features.

***Geometry*** is another word that denotes a geographic feature. Originally the word ***geometry*** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

The discussion here considers these terms synonymous: ***geographic feature***, ***geospatial feature***, ***feature***, or ***geometry***. The term most commonly used is ***geometry***, defined as a point or an aggregate of points representing anything in the world that has a location.

The following material covers these topics:

The spatial data types implemented in MySQL model

The basis of the spatial extensions in the OpenGIS geometry model

Data formats for representing spatial data

How to use spatial data in MySQL

Use of indexing for spatial data

MySQL differences from the OpenGIS specification

For information about functions that operate on spatial data, see [Section 12.17, “Spatial Analysis Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#spatial-analysis-functions).

### Additional Resources

These standards are important for the MySQL implementation of spatial operations:

SQL/MM Part 3: Spatial.

The [Open Geospatial Consortium](http://www.opengeospatial.org) publishes the OpenGIS® Implementation Standard for Geographic information, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. See in particular Simple Feature Access - Part 1: Common Architecture, and Simple Feature Access - Part 2: SQL Option. The Open Geospatial Consortium (OGC) maintains a website at <http://www.opengeospatial.org/>. The specification is available there at <http://www.opengeospatial.org/standards/sfs>. It contains additional information relevant to the material here.

The grammar for [spatial reference system](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-reference-systems) (SRS) definitions is based on the grammar defined in OpenGIS Implementation Specification: Coordinate Transformation Services, Revision 1.00, OGC 01-009, January 12, 2001, Section 7.2. This specification is available at <http://www.opengeospatial.org/standards/ct>. For differences from that specification in SRS definitions as implemented in MySQL, see [Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-spatial-reference-system).

If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: <https://forums.mysql.com/list.php?23>.

### 11.4.1 Spatial Data Types

MySQL has spatial data types that correspond to OpenGIS classes. The basis for these types is described in [Section 11.4.2, “The OpenGIS Geometry Model”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#opengis-geometry-model).

Some spatial data types hold single geometry values:

**GEOMETRY**

**POINT**

**LINESTRING**

**POLYGON**

**GEOMETRY** can store geometry values of any type. The other single-value types (**POINT**, **LINESTRING**, and **POLYGON**) restrict their values to a particular geometry type.

The other spatial data types hold collections of values:

**MULTIPOINT**

**MULTILINESTRING**

**MULTIPOLYGON**

**GEOMETRYCOLLECTION**

**GEOMETRYCOLLECTION** can store a collection of objects of any type. The other collection types (**MULTIPOINT**, **MULTILINESTRING**, and **MULTIPOLYGON**) restrict collection members to those having a particular geometry type.

Example: To create a table named **geom** that has a column named **g** that can store values of any geometry type, use this statement:

CREATE TABLE geom (g GEOMETRY);

Columns with a spatial data type can have an **SRID** attribute, to explicitly indicate the spatial reference system (SRS) for values stored in the column. For example:

CREATE TABLE geom (

p POINT SRID 0,

g GEOMETRY NOT NULL SRID 4326

);

**SPATIAL** indexes can be created on spatial columns if they are **NOT NULL** and have a specific SRID, so if you plan to index the column, declare it with the **NOT NULL** and **SRID** attributes:

CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);

**InnoDB** tables permit **SRID** values for Cartesian and geographic SRSs. **MyISAM** tables permit **SRID** values for Cartesian SRSs.

The **SRID** attribute makes a spatial column SRID-restricted, which has these implications:

The column can contain only values with the given SRID. Attempts to insert values with a different SRID produce an error.

The optimizer can use **SPATIAL** indexes on the column. See [Section 8.3.3, “SPATIAL Index Optimization”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#spatial-index-optimization).

Spatial columns with no **SRID** attribute are not SRID-restricted and accept values with any SRID. However, the optimizer cannot use **SPATIAL** indexes on them until the column definition is modified to include an **SRID** attribute, which may require that the column contents first be modified so that all values have the same SRID.

For other examples showing how to use spatial data types in MySQL, see [Section 11.4.6, “Creating Spatial Columns”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#creating-spatial-columns). For information about spatial reference systems, see [Section 11.4.5, “Spatial Reference System Support”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-reference-systems).

### 11.4.2 The OpenGIS Geometry Model

[11.4.2.1 The Geometry Class Hierarchy](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-geometry-class-hierarchy)

[11.4.2.2 Geometry Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-geometry)

[11.4.2.3 Point Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-point)

[11.4.2.4 Curve Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-curve)

[11.4.2.5 LineString Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-linestring)

[11.4.2.6 Surface Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-surface)

[11.4.2.7 Polygon Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-polygon)

[11.4.2.8 GeometryCollection Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-geometrycollection)

[11.4.2.9 MultiPoint Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-multipoint)

[11.4.2.10 MultiCurve Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-multicurve)

[11.4.2.11 MultiLineString Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-multilinestring)

[11.4.2.12 MultiSurface Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-multisurface)

[11.4.2.13 MultiPolygon Class](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-class-multipolygon)

The set of geometry types proposed by OGC's ***SQL with Geometry Types*** environment is based on the ***OpenGIS Geometry Model***. In this model, each geometric object has the following general properties:

It is associated with a spatial reference system, which describes the coordinate space in which the object is defined.

It belongs to some geometry class.

#### 11.4.2.1 The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

**Geometry** (noninstantiable)

**Point** (instantiable)

**Curve** (noninstantiable)

**LineString** (instantiable)

**Line**

**LinearRing**

**Surface** (noninstantiable)

**Polygon** (instantiable)

**GeometryCollection** (instantiable)

**MultiPoint** (instantiable)

**MultiCurve** (noninstantiable)

**MultiLineString** (instantiable)

**MultiSurface** (noninstantiable)

**MultiPolygon** (instantiable)

It is not possible to create objects in noninstantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

**Geometry** is the base class. It is an abstract class. The instantiable subclasses of **Geometry** are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base **Geometry** class has subclasses for **Point**, **Curve**, **Surface**, and **GeometryCollection**:

**Point** represents zero-dimensional objects.

**Curve** represents one-dimensional objects, and has subclass **LineString**, with sub-subclasses **Line** and **LinearRing**.

**Surface** is designed for two-dimensional objects and has subclass **Polygon**.

**GeometryCollection** has specialized zero-, one-, and two-dimensional collection classes named **MultiPoint**, **MultiLineString**, and **MultiPolygon** for modeling geometries corresponding to collections of **Points**, **LineStrings**, and **Polygons**, respectively. **MultiCurve** and **MultiSurface** are introduced as abstract superclasses that generalize the collection interfaces to handle **Curves** and **Surfaces**.

**Geometry**, **Curve**, **Surface**, **MultiCurve**, and **MultiSurface** are defined as noninstantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

**Point**, **LineString**, **Polygon**, **GeometryCollection**, **MultiPoint**, **MultiLineString**, and **MultiPolygon** are instantiable classes.

#### 11.4.2.2 Geometry Class

**Geometry** is the root class of the hierarchy. It is a noninstantiable class but has a number of properties, described in the following list, that are common to all geometry values created from any of the **Geometry** subclasses. Particular subclasses have their own specific properties, described later.

***Geometry Properties***

A geometry value has the following properties:

Its ***type***. Each geometry belongs to one of the instantiable classes in the hierarchy.

Its ***SRID***, or spatial reference identifier. This value identifies the geometry's associated spatial reference system that describes the coordinate space in which the geometry object is defined.

In MySQL, the SRID value is an integer associated with the geometry value. The maximum usable SRID value is 232−1. If a larger value is given, only the lower 32 bits are used.

SRID 0 represents an infinite flat Cartesian plane with no units assigned to its axes. To ensure SRID 0 behavior, create geometry values using SRID 0. SRID 0 is the default for new geometry values if no SRID is specified.

For computations on multiple geometry values, all values must have the same SRID or an error occurs.

Its ***coordinates*** in its spatial reference system, represented as double-precision (8-byte) numbers. All nonempty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the ***planar*** coordinate system and the distance on the ***geodetic*** system (coordinates on the Earth's surface) are different things.

Its ***interior***, ***boundary***, and ***exterior***.

Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

Its ***MBR*** (minimum bounding rectangle), or envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))

Whether the value is ***simple*** or ***nonsimple***. Geometry values of types (**LineString**, **MultiPoint**, **MultiLineString**) are either simple or nonsimple. Each type determines its own assertions for being simple or nonsimple.

Whether the value is ***closed*** or ***not closed***. Geometry values of types (**LineString**, **MultiString**) are either closed or not closed. Each type determines its own assertions for being closed or not closed.

Whether the value is ***empty*** or ***nonempty*** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a **NULL** value). An empty geometry is defined to be always simple and has an area of 0.

Its ***dimension***. A geometry can have a dimension of −1, 0, 1, or 2:

−1 for an empty geometry.

0 for a geometry with no length and no area.

1 for a geometry with nonzero length and zero area.

2 for a geometry with nonzero area.

**Point** objects have a dimension of zero. **LineString** objects have a dimension of 1. **Polygon** objects have a dimension of 2. The dimensions of **MultiPoint**, **MultiLineString**, and **MultiPolygon** objects are the same as the dimensions of the elements they consist of.

#### 11.4.2.3 Point Class

A **Point** is a geometry that represents a single location in coordinate space.

***Point Examples***

Imagine a large-scale map of the world with many cities. A **Point** object could represent each city.

On a city map, a **Point** object could represent a bus stop.

***Point Properties***

X-coordinate value.

Y-coordinate value.

**Point** is defined as a zero-dimensional geometry.

The boundary of a **Point** is the empty set.

#### 11.4.2.4 Curve Class

A **Curve** is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of **Curve** define the type of interpolation between points. **Curve** is a noninstantiable class.

***Curve Properties***

A **Curve** has the coordinates of its points.

A **Curve** is defined as a one-dimensional geometry.

A **Curve** is simple if it does not pass through the same point twice, with the exception that a curve can still be simple if the start and end points are the same.

A **Curve** is closed if its start point is equal to its endpoint.

The boundary of a closed **Curve** is empty.

The boundary of a nonclosed **Curve** consists of its two endpoints.

A **Curve** that is simple and closed is a **LinearRing**.

#### 11.4.2.5 LineString Class

A **LineString** is a **Curve** with linear interpolation between points.

***LineString Examples***

On a world map, **LineString** objects could represent rivers.

In a city map, **LineString** objects could represent streets.

***LineString Properties***

A **LineString** has coordinates of segments, defined by each consecutive pair of points.

A **LineString** is a **Line** if it consists of exactly two points.

A **LineString** is a **LinearRing** if it is both closed and simple.

#### 11.4.2.6 Surface Class

A **Surface** is a two-dimensional geometry. It is a noninstantiable class. Its only instantiable subclass is **Polygon**.

***Surface Properties***

A **Surface** is defined as a two-dimensional geometry.

The OpenGIS specification defines a simple **Surface** as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.

The boundary of a simple **Surface** is the set of closed curves corresponding to its exterior and interior boundaries.

#### 11.4.2.7 Polygon Class

A **Polygon** is a planar **Surface** representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the **Polygon**.

***Polygon Examples***

On a region map, **Polygon** objects could represent forests, districts, and so on.

***Polygon Assertions***

The boundary of a **Polygon** consists of a set of **LinearRing** objects (that is, **LineString** objects that are both simple and closed) that make up its exterior and interior boundaries.

A **Polygon** has no rings that cross. The rings in the boundary of a **Polygon** may intersect at a **Point**, but only as a tangent.

A **Polygon** has no lines, spikes, or punctures.

A **Polygon** has an interior that is a connected point set.

A **Polygon** may have holes. The exterior of a **Polygon** with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a **Polygon** a simple geometry.

#### 11.4.2.8 GeometryCollection Class

A **GeomCollection** is a geometry that is a collection of zero or more geometries of any class.

**GeomCollection** and **GeometryCollection** are synonymous, with **GeomCollection** the preferred type name.

All the elements in a geometry collection must be in the same spatial reference system (that is, in the same coordinate system). There are no other constraints on the elements of a geometry collection, although the subclasses of **GeomCollection** described in the following sections may restrict membership. Restrictions may be based on:

Element type (for example, a **MultiPoint** may contain only **Point** elements)

Dimension

Constraints on the degree of spatial overlap between elements

#### 11.4.2.9 MultiPoint Class

A **MultiPoint** is a geometry collection composed of **Point** elements. The points are not connected or ordered in any way.

***MultiPoint Examples***

On a world map, a **MultiPoint** could represent a chain of small islands.

On a city map, a **MultiPoint** could represent the outlets for a ticket office.

***MultiPoint Properties***

A **MultiPoint** is a zero-dimensional geometry.

A **MultiPoint** is simple if no two of its **Point** values are equal (have identical coordinate values).

The boundary of a **MultiPoint** is the empty set.

#### 11.4.2.10 MultiCurve Class

A **MultiCurve** is a geometry collection composed of **Curve** elements. **MultiCurve** is a noninstantiable class.

***MultiCurve Properties***

A **MultiCurve** is a one-dimensional geometry.

A **MultiCurve** is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.

A **MultiCurve** boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a **MultiCurve** if it is in the boundaries of an odd number of **Curve** elements.

A **MultiCurve** is closed if all of its elements are closed.

The boundary of a closed **MultiCurve** is always empty.

#### 11.4.2.11 MultiLineString Class

A **MultiLineString** is a **MultiCurve** geometry collection composed of **LineString** elements.

***MultiLineString Examples***

On a region map, a **MultiLineString** could represent a river system or a highway system.

#### 11.4.2.12 MultiSurface Class

A **MultiSurface** is a geometry collection composed of surface elements. **MultiSurface** is a noninstantiable class. Its only instantiable subclass is **MultiPolygon**.

***MultiSurface Assertions***

Surfaces within a **MultiSurface** have no interiors that intersect.

Surfaces within a **MultiSurface** have boundaries that intersect at most at a finite number of points.

#### 11.4.2.13 MultiPolygon Class

A **MultiPolygon** is a **MultiSurface** object composed of **Polygon** elements.

***MultiPolygon Examples***

On a region map, a **MultiPolygon** could represent a system of lakes.

***MultiPolygon Assertions***

A **MultiPolygon** has no two **Polygon** elements with interiors that intersect.

A **MultiPolygon** has no two **Polygon** elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.

A **MultiPolygon** may not have cut lines, spikes, or punctures. A **MultiPolygon** is a regular, closed point set.

A **MultiPolygon** that has more than one **Polygon** has an interior that is not connected. The number of connected components of the interior of a **MultiPolygon** is equal to the number of **Polygon** values in the **MultiPolygon**.

***MultiPolygon Properties***

A **MultiPolygon** is a two-dimensional geometry.

A **MultiPolygon** boundary is a set of closed curves (**LineString** values) corresponding to the boundaries of its **Polygon** elements.

Each **Curve** in the boundary of the **MultiPolygon** is in the boundary of exactly one **Polygon** element.

Every **Curve** in the boundary of an **Polygon** element is in the boundary of the **MultiPolygon**.

### 11.4.3 Supported Spatial Data Formats

Two standard spatial data formats are used to represent geometry objects in queries:

Well-Known Text (WKT) format

Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format. (Internal format is like WKB but with an initial 4 bytes to indicate the SRID.)

There are functions available to convert between different data formats; see [Section 12.17.6, “Geometry Format Conversion Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#gis-format-conversion-functions).

The following sections describe the spatial data formats MySQL uses:

[Well-Known Text (WKT) Format](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-wkt-format)

[Well-Known Binary (WKB) Format](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-wkb-format)

[Internal Geometry Storage Format](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-internal-format)

#### Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of geometry values is designed for exchanging geometry data in ASCII form. The OpenGIS specification provides a Backus-Naur grammar that specifies the formal production rules for writing WKT values (see [Section 11.4, “Spatial Data Types”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-types)).

Examples of WKT representations of geometry objects:

A **Point**:

POINT(15 20)

The point coordinates are specified with no separating comma. This differs from the syntax for the SQL [**Point()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_point) function, which requires a comma between the coordinates. Take care to use the syntax appropriate to the context of a given spatial operation. For example, the following statements both use [**ST\_X()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_st-x) to extract the X-coordinate from a **Point** object. The first produces the object directly using the [**Point()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_point) function. The second uses a WKT representation converted to a **Point** with **[ST\_GeomFromText()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-geomfromtext)**.

mysql> **SELECT ST\_X(Point(15, 20));**

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

| ST\_X(POINT(15, 20)) |

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

| 15 |

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

mysql> **SELECT ST\_X(ST\_GeomFromText('POINT(15 20)'));**

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

| ST\_X(ST\_GeomFromText('POINT(15 20)')) |

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

| 15 |

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

A **LineString** with four points:

LINESTRING(0 0, 10 10, 20 25, 50 60)

The point coordinate pairs are separated by commas.

A **Polygon** with one exterior ring and one interior ring:

POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))

A **MultiPoint** with three **Point** values:

MULTIPOINT(0 0, 20 20, 60 60)

Spatial functions such as **[ST\_MPointFromText()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-mpointfromtext)** and **[ST\_GeomFromText()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-geomfromtext)** that accept WKT-format representations of **MultiPoint** values permit individual points within values to be surrounded by parentheses. For example, both of the following function calls are valid:

ST\_MPointFromText('MULTIPOINT (1 1, 2 2, 3 3)')

ST\_MPointFromText('MULTIPOINT ((1 1), (2 2), (3 3))')

A **MultiLineString** with two **LineString** values:

MULTILINESTRING((10 10, 20 20), (15 15, 30 15))

A **MultiPolygon** with two **Polygon** values:

MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))

A **GeometryCollection** consisting of two **Point** values and one **LineString**:

GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))

#### Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation of geometric values is used for exchanging geometry data as binary streams represented by [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) values containing geometric WKB information. This format is defined by the OpenGIS specification (see [Section 11.4, “Spatial Data Types”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-types)). It is also defined in the ISO SQL/MM Part 3: Spatial standard.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to **POINT(1 -1)** consists of this sequence of 21 bytes, each represented by two hexadecimal digits:

0101000000000000000000F03F000000000000F0BF

The sequence consists of the components shown in the following table.

**Table 11.2 WKB Components Example**

| **Component** | **Size** | **Value** |
| --- | --- | --- |
| **Byte order** | 1 byte | **01** |
| **WKB type** | 4 bytes | **01000000** |
| **X coordinate** | 8 bytes | **000000000000F03F** |
| **Y coordinate** | 8 bytes | **000000000000F0BF** |

Component representation is as follows:

The byte order indicator is either 1 or 0 to signify little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.

The WKB type is a code that indicates the geometry type. MySQL uses values from 1 through 7 to indicate **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, and **GeometryCollection**.

A **Point** value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values have more complex data structures, as detailed in the OpenGIS specification.

#### Internal Geometry Storage Format

MySQL stores geometry values using 4 bytes to indicate the SRID followed by the WKB representation of the value. For a description of WKB format, see [Well-Known Binary (WKB) Format](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-wkb-format).

For the WKB part, these MySQL-specific considerations apply:

The byte-order indicator byte is 1 because MySQL stores geometries as little-endian values.

MySQL supports geometry types of **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, and **GeometryCollection**. Other geometry types are not supported.

Only **GeometryCollection** can be empty. Such a value is stored with 0 elements.

Polygon rings can be specified both clockwise and counterclockwise. MySQL flips the rings automatically when reading data.

Cartesian coordinates are stored in the length unit of the spatial reference system, with X values in the X coordinates and Y values in the Y coordinates. Axis directions are those specified by the spatial reference system.

Geographic coordinates are stored in the angle unit of the spatial reference system, with longitudes in the X coordinates and latitudes in the Y coordinates. Axis directions and the meridian are those specified by the spatial reference system.

The [**LENGTH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_length) function returns the space in bytes required for value storage. Example:

mysql> **SET @g = ST\_GeomFromText('POINT(1 -1)');**

mysql> **SELECT LENGTH(@g);**

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

| LENGTH(@g) |

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

| 25 |

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

mysql> **SELECT HEX(@g);**

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

| HEX(@g) |

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

| 000000000101000000000000000000F03F000000000000F0BF |

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

The value length is 25 bytes, made up of these components (as can be seen from the hexadecimal value):

4 bytes for integer SRID (0)

1 byte for integer byte order (1 = little-endian)

4 bytes for integer type information (1 = **Point**)

8 bytes for double-precision X coordinate (1)

8 bytes for double-precision Y coordinate (−1)

### 11.4.4 Geometry Well-Formedness and Validity

For geometry values, MySQL distinguishes between the concepts of syntactically well-formed and geometrically valid.

A geometry is syntactically well-formed if it satisfies conditions such as those in this (nonexhaustive) list:

Linestrings have at least two points

Polygons have at least one ring

Polygon rings are closed (first and last points the same)

Polygon rings have at least 4 points (minimum polygon is a triangle with first and last points the same)

Collections are not empty (except **GeometryCollection**)

A geometry is geometrically valid if it is syntactically well-formed and satisfies conditions such as those in this (nonexhaustive) list:

Polygons are not self-intersecting

Polygon interior rings are inside the exterior ring

Multipolygons do not have overlapping polygons

Spatial functions fail if a geometry is not syntactically well-formed. Spatial import functions that parse WKT or WKB values raise an error for attempts to create a geometry that is not syntactically well-formed. Syntactic well-formedness is also checked for attempts to store geometries into tables.

It is permitted to insert, select, and update geometrically invalid geometries, but they must be syntactically well-formed. Due to the computational expense, MySQL does not check explicitly for geometric validity. Spatial computations may detect some cases of invalid geometries and raise an error, but they may also return an undefined result without detecting the invalidity. Applications that require geometically valid geometries should check them using the **[ST\_IsValid()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-isvalid)** function.

### 11.4.5 Spatial Reference System Support

A spatial reference system (SRS) for spatial data is a coordinate-based system for geographic locations.

There are different types of spatial reference systems:

A projected SRS is a projection of a globe onto a flat surface; that is, a flat map. For example, a light bulb inside a globe that shines on a paper cylinder surrounding the globe projects a map onto the paper. The result is georeferenced: Each point maps to a place on the globe. The coordinate system on that plane is Cartesian using a length unit (meters, feet, and so forth), rather than degrees of longitude and latitude.

The globes in this case are ellipsoids; that is, flattened spheres. Earth is a bit shorter in its North-South axis than its East-West axis, so a slightly flattened sphere is more correct, but perfect spheres permit faster calculations.

A geographic SRS is a nonprojected SRS representing longitude-latitude (or latitude-longitude) coordinates on an ellipsoid, in any angular unit.

The SRS denoted in MySQL by SRID 0 represents an infinite flat Cartesian plane with no units assigned to its axes. Unlike projected SRSs, it is not georeferenced and it does not necessarily represent Earth. It is an abstract plane that can be used for anything. SRID 0 is the default SRID for spatial data in MySQL.

MySQL maintains information about available spatial reference systems for spatial data in the data dictionary **mysql.st\_spatial\_reference\_systems** table, which can store entries for projected and geographic SRSs. This data dictionary table is invisible, but SRS entry contents are available through the **INFORMATION\_SCHEMA** [**ST\_SPATIAL\_REFERENCE\_SYSTEMS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-st-spatial-reference-systems-table) table, implemented as a view on **mysql.st\_spatial\_reference\_systems** (see [Section 26.3.36, “The INFORMATION\_SCHEMA ST\_SPATIAL\_REFERENCE\_SYSTEMS Table”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-st-spatial-reference-systems-table)).

The following example shows what an SRS entry looks like:

mysql> **SELECT \***

**FROM INFORMATION\_SCHEMA.ST\_SPATIAL\_REFERENCE\_SYSTEMS**

**WHERE SRS\_ID = 4326\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SRS\_NAME: WGS 84

SRS\_ID: 4326

ORGANIZATION: EPSG

ORGANIZATION\_COORDSYS\_ID: 4326

DEFINITION: GEOGCS["WGS 84",DATUM["World Geodetic System 1984",

SPHEROID["WGS 84",6378137,298.257223563,

AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],

PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],

UNIT["degree",0.017453292519943278,

AUTHORITY["EPSG","9122"]],

AXIS["Lat",NORTH],AXIS["Long",EAST],

AUTHORITY["EPSG","4326"]]

DESCRIPTION:

This entry describes the SRS used for GPS systems. It has a name (**SRS\_NAME**) of WGS 84 and an ID (**SRS\_ID**) of 4326, which is the ID used by the [European Petroleum Survey Group](http://epsg.org) (EPSG).

SRS definitions in the **DEFINITION** column are WKT values, represented as specified in the [Open Geospatial Consortium](http://www.opengeospatial.org) document [OGC 12-063r5](http://docs.opengeospatial.org/is/12-063r5/12-063r5.html).

**SRS\_ID** values represent the same kind of values as the SRID of geometry values or passed as the SRID argument to spatial functions. SRID 0 (the unitless Cartesian plane) is special. It is always a legal spatial reference system ID and can be used in any computations on spatial data that depend on SRID values.

For computations on multiple geometry values, all values must have the same SRID or an error occurs.

SRS definition parsing occurs on demand when definitions are needed by GIS functions. Parsed definitions are stored in the data dictionary cache to enable reuse and avoid incurring parsing overhead for every statement that needs SRS information.

To enable manipulation of SRS entries stored in the data dictionary, MySQL provides these SQL statements:

[**CREATE SPATIAL REFERENCE SYSTEM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-spatial-reference-system): See [Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-spatial-reference-system). The description for this statement includes additional information about SRS components.

[**DROP SPATIAL REFERENCE SYSTEM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-spatial-reference-system): See [Section 13.1.31, “DROP SPATIAL REFERENCE SYSTEM Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-spatial-reference-system).

### 11.4.6 Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with [**CREATE TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table) or [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table). Spatial columns are supported for **[MyISAM](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\storage-engines.html" \l "myisam-storage-engine" \o "16.2 The MyISAM Storage Engine)**, **[InnoDB](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\innodb-storage-engine.html" \o "Chapter 15 The InnoDB Storage Engine)**, [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html), and [**ARCHIVE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\storage-engines.html#archive-storage-engine) tables. See also the notes about spatial indexes under [Section 11.4.10, “Creating Spatial Indexes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#creating-spatial-indexes).

Columns with a spatial data type can have an SRID attribute, to explicitly indicate the spatial reference system (SRS) for values stored in the column. For implications of an SRID-restricted column, see [Section 11.4.1, “Spatial Data Types”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-type-overview).

Use the [**CREATE TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table) statement to create a table with a spatial column:

CREATE TABLE geom (g GEOMETRY);

Use the [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table) statement to add or drop a spatial column to or from an existing table:

ALTER TABLE geom ADD pt POINT;

ALTER TABLE geom DROP pt;

### 11.4.7 Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values to internal geometry format:

Perform the conversion directly in the [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) statement:

INSERT INTO geom VALUES (ST\_GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';

INSERT INTO geom VALUES (ST\_GeomFromText(@g));

Perform the conversion prior to the [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert):

SET @g = ST\_GeomFromText('POINT(1 1)');

INSERT INTO geom VALUES (@g);

The following examples insert more complex geometries into the table:

SET @g = 'LINESTRING(0 0,1 1,2 2)';

INSERT INTO geom VALUES (ST\_GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';

INSERT INTO geom VALUES (ST\_GeomFromText(@g));

SET @g =

'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';

INSERT INTO geom VALUES (ST\_GeomFromText(@g));

The preceding examples use **[ST\_GeomFromText()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-geomfromtext)** to create geometry values. You can also use type-specific functions:

SET @g = 'POINT(1 1)';

INSERT INTO geom VALUES (ST\_PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';

INSERT INTO geom VALUES (ST\_LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';

INSERT INTO geom VALUES (ST\_PolygonFromText(@g));

SET @g =

'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';

INSERT INTO geom VALUES (ST\_GeomCollFromText(@g));

A client application program that wants to use WKB representations of geometry values is responsible for sending correctly formed WKB in queries to the server. There are several ways to satisfy this requirement. For example:

Inserting a **POINT(1 1)** value with hex literal syntax:

INSERT INTO geom VALUES

(ST\_GeomFromWKB(X'0101000000000000000000F03F000000000000F03F'));

An ODBC application can send a WKB representation, binding it to a placeholder using an argument of [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) type:

INSERT INTO geom VALUES (ST\_GeomFromWKB(?))

Other programming interfaces may support a similar placeholder mechanism.

In a C program, you can escape a binary value using **[mysql\_real\_escape\_string\_quote()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-escape-string-quote.html" \t "_top)** and include the result in a query string that is sent to the server. See [mysql\_real\_escape\_string\_quote()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-escape-string-quote.html" \t "_top).

### 11.4.8 Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them to WKT or WKB format.

Fetching spatial data in internal format:

Fetching geometry values using internal format can be useful in table-to-table transfers:

CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;

Fetching spatial data in WKT format:

The **[ST\_AsText()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-astext)** function converts a geometry from internal format to a WKT string.

SELECT ST\_AsText(g) FROM geom;

Fetching spatial data in WKB format:

The **[ST\_AsBinary()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-asbinary)** function converts a geometry from internal format to a [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) containing the WKB value.

SELECT ST\_AsBinary(g) FROM geom;

### 11.4.9 Optimizing Spatial Analysis

For **[MyISAM](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\storage-engines.html" \l "myisam-storage-engine" \o "16.2 The MyISAM Storage Engine)** and **InnoDB** tables, search operations in columns containing spatial data can be optimized using **SPATIAL** indexes. The most typical operations are:

Point queries that search for all objects that contain a given point

Region queries that search for all objects that overlap a given region

MySQL uses ***R-Trees with quadratic splitting*** for **SPATIAL** indexes on spatial columns. A **SPATIAL** index is built using the minimum bounding rectangle (MBR) of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. In a non-**SPATIAL** index, you must declare a prefix for any spatial column except for **POINT** columns.

**MyISAM** and **InnoDB** support both **SPATIAL** and non-**SPATIAL** indexes. Other storage engines support non-**SPATIAL** indexes, as described in [Section 13.1.15, “CREATE INDEX Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-index).

### 11.4.10 Creating Spatial Indexes

For **InnoDB** and **MyISAM** tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but using the **SPATIAL** keyword. Columns in spatial indexes must be declared **NOT NULL**. The following examples demonstrate how to create spatial indexes:

With [**CREATE TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table):

CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326, SPATIAL INDEX(g));

With [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table):

CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);

ALTER TABLE geom ADD SPATIAL INDEX(g);

With [**CREATE INDEX**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-index):

CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);

CREATE SPATIAL INDEX g ON geom (g);

**SPATIAL INDEX** creates an R-tree index. For storage engines that support nonspatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values is useful for exact-value lookups, but not for range scans.

The optimizer can use spatial indexes defined on columns that are SRID-restricted. For more information, see [Section 11.4.1, “Spatial Data Types”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#spatial-type-overview), and [Section 8.3.3, “SPATIAL Index Optimization”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#spatial-index-optimization).

For more information on indexing spatial columns, see [Section 13.1.15, “CREATE INDEX Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-index).

To drop spatial indexes, use [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table) or [**DROP INDEX**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-index):

With [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table):

ALTER TABLE geom DROP INDEX g;

With [**DROP INDEX**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-index):

DROP INDEX g ON geom;

Example: Suppose that a table **geom** contains more than 32,000 geometries, which are stored in the column **g** of type **GEOMETRY**. The table also has an **AUTO\_INCREMENT** column **fid** for storing object ID values.

mysql> **DESCRIBE geom;**

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

| Field | Type | Null | Key | Default | Extra |

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

| fid | int(11) | | PRI | NULL | auto\_increment |

| g | geometry | | | | |

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

2 rows in set (0.00 sec)

mysql> **SELECT COUNT(\*) FROM geom;**

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

| count(\*) |

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

| 32376 |

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

1 row in set (0.00 sec)

To add a spatial index on the column **g**, use this statement:

mysql> **ALTER TABLE geom ADD SPATIAL INDEX(g);**

Query OK, 32376 rows affected (4.05 sec)

Records: 32376 Duplicates: 0 Warnings: 0

### 11.4.11 Using Spatial Indexes

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as **[MBRContains()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_mbrcontains)** or **[MBRWithin()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_mbrwithin)** in the **WHERE** clause. The following query finds all objects that are in the given rectangle:

mysql> **SET @poly =**

-> **'Polygon((30000 15000,**

**31000 15000,**

**31000 16000,**

**30000 16000,**

**30000 15000))';**

mysql> **SELECT fid,ST\_AsText(g) FROM geom WHERE**

-> **MBRContains(ST\_GeomFromText(@poly),g);**

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

| fid | ST\_AsText(g) |

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

| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |

| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |

| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |

| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |

| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |

| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |

| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |

| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |

| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |

| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |

| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |

| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |

| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |

| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |

| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |

| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |

| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |

| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |

| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |

| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |

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

20 rows in set (0.00 sec)

Use [**EXPLAIN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#explain) to check the way this query is executed:

mysql> **SET @poly =**

-> **'Polygon((30000 15000,**

**31000 15000,**

**31000 16000,**

**30000 16000,**

**30000 15000))';**

mysql> **EXPLAIN SELECT fid,ST\_AsText(g) FROM geom WHERE**

-> **MBRContains(ST\_GeomFromText(@poly),g)\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: SIMPLE

table: geom

type: range

possible\_keys: g

key: g

key\_len: 32

ref: NULL

rows: 50

Extra: Using where

1 row in set (0.00 sec)

Check what would happen without a spatial index:

mysql> **SET @poly =**

-> **'Polygon((30000 15000,**

**31000 15000,**

**31000 16000,**

**30000 16000,**

**30000 15000))';**

mysql> **EXPLAIN SELECT fid,ST\_AsText(g) FROM g IGNORE INDEX (g) WHERE**

-> **MBRContains(ST\_GeomFromText(@poly),g)\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: SIMPLE

table: geom

type: ALL

possible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 32376

Extra: Using where

1 row in set (0.00 sec)

Executing the [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

mysql> **SET @poly =**

-> **'Polygon((30000 15000,**

**31000 15000,**

**31000 16000,**

**30000 16000,**

**30000 15000))';**

mysql> **SELECT fid,ST\_AsText(g) FROM geom IGNORE INDEX (g) WHERE**

-> **MBRContains(ST\_GeomFromText(@poly),g);**

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

| fid | ST\_AsText(g) |

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

| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |

| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |

| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |

| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |

| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |

| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |

| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |

| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |

| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |

| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |

| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |

| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |

| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |

| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |

| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |

| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |

| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |

| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |

| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |

| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |

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

20 rows in set (0.46 sec)

## 11.5 The JSON Data Type

[Creating JSON Values](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "json-values" \o "Creating JSON Values)

[Normalization, Merging, and Autowrapping of JSON Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-normalization)

[Searching and Modifying JSON Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-paths)

[JSON Path Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-path-syntax)

[Comparison and Ordering of JSON Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-comparison)

[Converting between JSON and non-JSON values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-converting-between-types)

[Aggregation of JSON Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-aggregation)

MySQL supports a native **JSON** data type defined by [RFC 7159](https://tools.ietf.org/html/rfc7159) that enables efficient access to data in JSON (JavaScript Object Notation) documents. The **JSON** data type provides these advantages over storing JSON-format strings in a string column:

Automatic validation of JSON documents stored in **JSON** columns. Invalid documents produce an error.

Optimized storage format. JSON documents stored in **JSON** columns are converted to an internal format that permits quick read access to document elements. When the server later must read a JSON value stored in this binary format, the value need not be parsed from a text representation. The binary format is structured to enable the server to look up subobjects or nested values directly by key or array index without reading all values before or after them in the document.

MySQL 8.0 also supports the JSON Merge Patch format defined in [RFC 7396](https://tools.ietf.org/html/rfc7396), using the [**JSON\_MERGE\_PATCH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-patch) function. See the description of this function, as well as [Normalization, Merging, and Autowrapping of JSON Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-normalization), for examples and further information.

**Note**

This discussion uses **JSON** in monotype to indicate specifically the JSON data type and “JSON” in regular font to indicate JSON data in general.

The space required to store a **JSON** document is roughly the same as for [**LONGBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) or [**LONGTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob); see [Section 11.7, “Data Type Storage Requirements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#storage-requirements), for more information. It is important to keep in mind that the size of any JSON document stored in a **JSON** column is limited to the value of the **[max\_allowed\_packet](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_allowed_packet)** system variable. (When the server is manipulating a JSON value internally in memory, it can be larger than this; the limit applies when the server stores it.) You can obtain the amount of space required to store a JSON document using the [**JSON\_STORAGE\_SIZE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-storage-size) function; note that for a [**JSON**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json) column, the storage size—and thus the value returned by this function—is that used by the column prior to any partial updates that may have been performed on it (see the discussion of the JSON partial update optimization later in this section).

Prior to MySQL 8.0.13, a **JSON** column cannot have a non-**NULL** default value.

Along with the **JSON** data type, a set of SQL functions is available to enable operations on JSON values, such as creation, manipulation, and searching. The following discussion shows examples of these operations. For details about individual functions, see [Section 12.18, “JSON Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#json-functions).

A set of spatial functions for operating on GeoJSON values is also available. See [Section 12.17.11, “Spatial GeoJSON Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#spatial-geojson-functions).

**JSON** columns, like columns of other binary types, are not indexed directly; instead, you can create an index on a generated column that extracts a scalar value from the **JSON** column. See [Indexing a Generated Column to Provide a JSON Column Index](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#json-column-indirect-index), for a detailed example.

The MySQL optimizer also looks for compatible indexes on virtual columns that match JSON expressions.

In MySQL 8.0.17 and later, the **[InnoDB](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\innodb-storage-engine.html" \o "Chapter 15 The InnoDB Storage Engine)** storage engine supports multi-valued indexes on JSON arrays. See [Multi-Valued Indexes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-index-multi-valued).

MySQL NDB Cluster 8.0 supports **JSON** columns and MySQL JSON functions, including creation of an index on a column generated from a **JSON** column as a workaround for being unable to index a **JSON** column. A maximum of 3 **JSON** columns per [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) table is supported.

### Partial Updates of JSON Values

In MySQL 8.0, the optimizer can perform a partial, in-place update of a **JSON** column instead of removing the old document and writing the new document in its entirety to the column. This optimization can be performed for an update that meets the following conditions:

The column being updated was declared as **JSON**.

The [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) statement uses any of the three functions [**JSON\_SET()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-set), [**JSON\_REPLACE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-replace), or [**JSON\_REMOVE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-remove) to update the column. A direct assignment of the column value (for example, **UPDATE mytable SET jcol = '{"a": 10, "b": 25}'**) cannot be performed as a partial update.

Updates of multiple **JSON** columns in a single **UPDATE** statement can be optimized in this fashion; MySQL can perform partial updates of only those columns whose values are updated using the three functions just listed.

The input column and the target column must be the same column; a statement such as **UPDATE mytable SET jcol1 = JSON\_SET(jcol2, '$.a', 100)** cannot be performed as a partial update.

The update can use nested calls to any of the functions listed in the previous item, in any combination, as long as the input and target columns are the same.

All changes replace existing array or object values with new ones, and do not add any new elements to the parent object or array.

The value being replaced must be at least as large as the replacement value. In other words, the new value cannot be any larger than the old one.

A possible exception to this requirement occurs when a previous partial update has left sufficient space for the larger value. You can use the function [**JSON\_STORAGE\_FREE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-storage-free) see how much space has been freed by any partial updates of a **JSON** column.

Such partial updates can be written to the binary log using a compact format that saves space; this can be enabled by setting the **[binlog\_row\_value\_options](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_binlog_row_value_options)** system variable to **PARTIAL\_JSON**. See the description of this variable for more information.

The next few sections provide basic information regarding the creation and manipulation of JSON values.

### Creating JSON Values

A JSON array contains a list of values separated by commas and enclosed within **[** and **]** characters:

["abc", 10, null, true, false]

A JSON object contains a set of key-value pairs separated by commas and enclosed within **{** and **}** characters:

{"k1": "value", "k2": 10}

As the examples illustrate, JSON arrays and objects can contain scalar values that are strings or numbers, the JSON null literal, or the JSON boolean true or false literals. Keys in JSON objects must be strings. Temporal (date, time, or datetime) scalar values are also permitted:

["12:18:29.000000", "2015-07-29", "2015-07-29 12:18:29.000000"]

Nesting is permitted within JSON array elements and JSON object key values:

[99, {"id": "HK500", "cost": 75.99}, ["hot", "cold"]]

{"k1": "value", "k2": [10, 20]}

You can also obtain JSON values from a number of functions supplied by MySQL for this purpose (see [Section 12.18.2, “Functions That Create JSON Values”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#json-creation-functions)) as well as by casting values of other types to the **JSON** type using [**CAST(*value* AS JSON)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_cast) (see [Converting between JSON and non-JSON values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-converting-between-types)). The next several paragraphs describe how MySQL handles JSON values provided as input.

In MySQL, JSON values are written as strings. MySQL parses any string used in a context that requires a JSON value, and produces an error if it is not valid as JSON. These contexts include inserting a value into a column that has the **JSON** data type and passing an argument to a function that expects a JSON value (usually shown as ***json\_doc*** or ***json\_val*** in the documentation for MySQL JSON functions), as the following examples demonstrate:

Attempting to insert a value into a **JSON** column succeeds if the value is a valid JSON value, but fails if it is not:

mysql> **CREATE TABLE t1 (jdoc JSON);**

Query OK, 0 rows affected (0.20 sec)

mysql> **INSERT INTO t1 VALUES('{"key1": "value1", "key2": "value2"}');**

Query OK, 1 row affected (0.01 sec)

mysql> **INSERT INTO t1 VALUES('[1, 2,');**

ERROR 3140 (22032) at line 2: Invalid JSON text:

"Invalid value." at position 6 in value (or column) '[1, 2,'.

Positions for “at position ***N***” in such error messages are 0-based, but should be considered rough indications of where the problem in a value actually occurs.

The [**JSON\_TYPE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-type) function expects a JSON argument and attempts to parse it into a JSON value. It returns the value's JSON type if it is valid and produces an error otherwise:

mysql> **SELECT JSON\_TYPE('["a", "b", 1]');**

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

| JSON\_TYPE('["a", "b", 1]') |

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

| ARRAY |

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

mysql> **SELECT JSON\_TYPE('"hello"');**

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

| JSON\_TYPE('"hello"') |

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

| STRING |

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

mysql> **SELECT JSON\_TYPE('hello');**

ERROR 3146 (22032): Invalid data type for JSON data in argument 1

to function json\_type; a JSON string or JSON type is required.

MySQL handles strings used in JSON context using the **utf8mb4** character set and **utf8mb4\_bin** collation. Strings in other character sets are converted to **utf8mb4** as necessary. (For strings in the **ascii** or **utf8** character sets, no conversion is needed because **ascii** and **utf8** are subsets of **utf8mb4**.)

As an alternative to writing JSON values using literal strings, functions exist for composing JSON values from component elements. [**JSON\_ARRAY()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-array) takes a (possibly empty) list of values and returns a JSON array containing those values:

mysql> **SELECT JSON\_ARRAY('a', 1, NOW());**

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

| JSON\_ARRAY('a', 1, NOW()) |

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

| ["a", 1, "2015-07-27 09:43:47.000000"] |

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

[**JSON\_OBJECT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-object) takes a (possibly empty) list of key-value pairs and returns a JSON object containing those pairs:

mysql> **SELECT JSON\_OBJECT('key1', 1, 'key2', 'abc');**

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

| JSON\_OBJECT('key1', 1, 'key2', 'abc') |

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

| {"key1": 1, "key2": "abc"} |

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

[**JSON\_MERGE\_PRESERVE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-preserve) takes two or more JSON documents and returns the combined result:

mysql> **SELECT JSON\_MERGE\_PRESERVE('["a", 1]', '{"key": "value"}');**

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

| JSON\_MERGE\_PRESERVE('["a", 1]', '{"key": "value"}') |

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

| ["a", 1, {"key": "value"}] |

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

1 row in set (0.00 sec)

For information about the merging rules, see [Normalization, Merging, and Autowrapping of JSON Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-normalization).

(MySQL 8.0.3 and later also support [**JSON\_MERGE\_PATCH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-patch), which has somewhat different behavior. See [JSON\_MERGE\_PATCH() compared with JSON\_MERGE\_PRESERVE()](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#json-merge-patch-json-merge-preserve-compared), for information about the differences between these two functions.)

JSON values can be assigned to user-defined variables:

mysql> **SET @j = JSON\_OBJECT('key', 'value');**

mysql> **SELECT @j;**

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

| @j |

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

| {"key": "value"} |

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

However, user-defined variables cannot be of **JSON** data type, so although **@j** in the preceding example looks like a JSON value and has the same character set and collation as a JSON value, it does not have the **JSON** data type. Instead, the result from [**JSON\_OBJECT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-object) is converted to a string when assigned to the variable.

Strings produced by converting JSON values have a character set of **utf8mb4** and a collation of **utf8mb4\_bin**:

mysql> **SELECT CHARSET(@j), COLLATION(@j);**

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

| CHARSET(@j) | COLLATION(@j) |

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

| utf8mb4 | utf8mb4\_bin |

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

Because **utf8mb4\_bin** is a binary collation, comparison of JSON values is case-sensitive.

mysql> **SELECT JSON\_ARRAY('x') = JSON\_ARRAY('X');**

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

| JSON\_ARRAY('x') = JSON\_ARRAY('X') |

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

| 0 |

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

Case sensitivity also applies to the JSON **null**, **true**, and **false** literals, which always must be written in lowercase:

mysql> **SELECT JSON\_VALID('null'), JSON\_VALID('Null'), JSON\_VALID('NULL');**

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

| JSON\_VALID('null') | JSON\_VALID('Null') | JSON\_VALID('NULL') |

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

| 1 | 0 | 0 |

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

mysql> **SELECT CAST('null' AS JSON);**

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

| CAST('null' AS JSON) |

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

| null |

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

1 row in set (0.00 sec)

mysql> **SELECT CAST('NULL' AS JSON);**

ERROR 3141 (22032): Invalid JSON text in argument 1 to function cast\_as\_json:

"Invalid value." at position 0 in 'NULL'.

Case sensitivity of the JSON literals differs from that of the SQL **NULL**, **TRUE**, and **FALSE** literals, which can be written in any lettercase:

mysql> **SELECT ISNULL(null), ISNULL(Null), ISNULL(NULL);**

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

| ISNULL(null) | ISNULL(Null) | ISNULL(NULL) |

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

| 1 | 1 | 1 |

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

Sometimes it may be necessary or desirable to insert quote characters (**"** or **'**) into a JSON document. Assume for this example that you want to insert some JSON objects containing strings representing sentences that state some facts about MySQL, each paired with an appropriate keyword, into a table created using the SQL statement shown here:

mysql> **CREATE TABLE facts (sentence JSON);**

Among these keyword-sentence pairs is this one:

mascot: The MySQL mascot is a dolphin named "Sakila".

One way to insert this as a JSON object into the **facts** table is to use the MySQL [**JSON\_OBJECT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-object) function. In this case, you must escape each quote character using a backslash, as shown here:

mysql> **INSERT INTO facts VALUES**

> **(JSON\_OBJECT("mascot", "Our mascot is a dolphin named \"Sakila\"."));**

This does not work in the same way if you insert the value as a JSON object literal, in which case, you must use the double backslash escape sequence, like this:

mysql> **INSERT INTO facts VALUES**

> **('{"mascot": "Our mascot is a dolphin named \\"Sakila\\"."}');**

Using the double backslash keeps MySQL from performing escape sequence processing, and instead causes it to pass the string literal to the storage engine for processing. After inserting the JSON object in either of the ways just shown, you can see that the backslashes are present in the JSON column value by doing a simple [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select), like this:

mysql> **SELECT sentence FROM facts;**

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

| sentence |

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

| {"mascot": "Our mascot is a dolphin named \"Sakila\"."} |

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

To look up this particular sentence employing **mascot** as the key, you can use the column-path operator [**->**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_json-column-path), as shown here:

mysql> SELECT col->"$.mascot" FROM qtest;

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

| col->"$.mascot" |

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

| "Our mascot is a dolphin named \"Sakila\"." |

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

1 row in set (0.00 sec)

This leaves the backslashes intact, along with the surrounding quote marks. To display the desired value using **mascot** as the key, but without including the surrounding quote marks or any escapes, use the inline path operator [**->>**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_json-inline-path), like this:

mysql> **SELECT sentence->>"$.mascot" FROM facts;**

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

| sentence->>"$.mascot" |

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

| Our mascot is a dolphin named "Sakila". |

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

**Note**

The previous example does not work as shown if the [**NO\_BACKSLASH\_ESCAPES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sqlmode_no_backslash_escapes) server SQL mode is enabled. If this mode is set, a single backslash instead of double backslashes can be used to insert the JSON object literal, and the backslashes are preserved. If you use the **JSON\_OBJECT()** function when performing the insert and this mode is set, you must alternate single and double quotes, like this:

mysql> **INSERT INTO facts VALUES**

> **(JSON\_OBJECT('mascot', 'Our mascot is a dolphin named "Sakila".'));**

See the description of the [**JSON\_UNQUOTE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-unquote) function for more information about the effects of this mode on escaped characters in JSON values.

### Normalization, Merging, and Autowrapping of JSON Values

When a string is parsed and found to be a valid JSON document, it is also normalized. This means that members with keys that duplicate a key found later in the document, reading from left to right, are discarded. The object value produced by the following [**JSON\_OBJECT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-object) call includes only the second **key1** element because that key name occurs earlier in the value, as shown here:

mysql> **SELECT JSON\_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');**

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

| JSON\_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def') |

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

| {"key1": "def", "key2": "abc"} |

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

Normalization is also performed when values are inserted into JSON columns, as shown here:

mysql> **CREATE TABLE t1 (c1 JSON);**

mysql> **INSERT INTO t1 VALUES**

> **('{"x": 17, "x": "red"}'),**

> **('{"x": 17, "x": "red", "x": [3, 5, 7]}');**

mysql> **SELECT c1 FROM t1;**

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

| c1 |

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

| {"x": "red"} |

| {"x": [3, 5, 7]} |

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

This “last duplicate key wins” behavior is suggested by [RFC 7159](https://tools.ietf.org/html/rfc7159) and is implemented by most JavaScript parsers. (Bug #86866, Bug #26369555)

In versions of MySQL prior to 8.0.3, members with keys that duplicated a key found earlier in the document were discarded. The object value produced by the following [**JSON\_OBJECT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-object) call does not include the second **key1** element because that key name occurs earlier in the value:

mysql> **SELECT JSON\_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');**

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

| JSON\_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def') |

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

| {"key1": 1, "key2": "abc"} |

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

Prior to MySQL 8.0.3, this “first duplicate key wins” normalization was also performed when inserting values into JSON columns.

mysql> **CREATE TABLE t1 (c1 JSON);**

mysql> **INSERT INTO t1 VALUES**

> **('{"x": 17, "x": "red"}'),**

> **('{"x": 17, "x": "red", "x": [3, 5, 7]}');**

mysql> **SELECT c1 FROM t1;**

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

| c1 |

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

| {"x": 17} |

| {"x": 17} |

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

MySQL also discards extra whitespace between keys, values, or elements in the original JSON document, and leaves (or inserts, when necessary) a single space following each comma (**,**) or colon (**:**) when displaying it. This is done to enhance readibility.

MySQL functions that produce JSON values (see [Section 12.18.2, “Functions That Create JSON Values”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#json-creation-functions)) always return normalized values.

To make lookups more efficient, MySQL also sorts the keys of a JSON object. You should be aware that the result of this ordering is subject to change and not guaranteed to be consistent across releases.

#### Merging JSON Values

Two merging algorithms are supported in MySQL 8.0.3 (and later), implemented by the functions [**JSON\_MERGE\_PRESERVE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-preserve) and [**JSON\_MERGE\_PATCH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-patch). These differ in how they handle duplicate keys: [**JSON\_MERGE\_PRESERVE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-preserve) retains values for duplicate keys, while [**JSON\_MERGE\_PATCH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-patch) discards all but the last value. The next few paragraphs explain how each of these two functions handles the merging of different combinations of JSON documents (that is, of objects and arrays).

**Note**

[**JSON\_MERGE\_PRESERVE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-merge-preserve) is the same as the **JSON\_MERGE()** function found in previous versions of MySQL (renamed in MySQL 8.0.3). **JSON\_MERGE()** is still supported as an alias for **JSON\_MERGE\_PRESERVE()** in MySQL 8.0, but is deprecated and subject to removal in a future release.

**Merging arrays.** In contexts that combine multiple arrays, the arrays are merged into a single array. **JSON\_MERGE\_PRESERVE()** does this by concatenating arrays named later to the end of the first array. **JSON\_MERGE\_PATCH()** considers each argument as an array consisting of a single element (thus having 0 as its index) and then applies “last duplicate key wins” logic to select only the last argument. You can compare the results shown by this query:

mysql> **SELECT**

-> **JSON\_MERGE\_PRESERVE('[1, 2]', '["a", "b", "c"]', '[true, false]') AS Preserve,**

-> **JSON\_MERGE\_PATCH('[1, 2]', '["a", "b", "c"]', '[true, false]') AS Patch\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Preserve: [1, 2, "a", "b", "c", true, false]

Patch: [true, false]

Multiple objects when merged produce a single object. **JSON\_MERGE\_PRESERVE()** handles multiple objects having the same key by combining all unique values for that key in an array; this array is then used as the value for that key in the result. **JSON\_MERGE\_PATCH()** discards values for which duplicate keys are found, working from left to right, so that the result contains only the last value for that key. The following query illustrates the difference in the results for the duplicate key **a**:

mysql> **SELECT**

-> **JSON\_MERGE\_PRESERVE('{"a": 1, "b": 2}', '{"c": 3, "a": 4}', '{"c": 5, "d": 3}') AS Preserve,**

-> **JSON\_MERGE\_PATCH('{"a": 3, "b": 2}', '{"c": 3, "a": 4}', '{"c": 5, "d": 3}') AS Patch\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Preserve: {"a": [1, 4], "b": 2, "c": [3, 5], "d": 3}

Patch: {"a": 4, "b": 2, "c": 5, "d": 3}

Nonarray values used in a context that requires an array value are autowrapped: The value is surrounded by **[** and **]** characters to convert it to an array. In the following statement, each argument is autowrapped as an array (**[1]**, **[2]**). These are then merged to produce a single result array; as in the previous two cases, **JSON\_MERGE\_PRESERVE()** combines values having the same key while **JSON\_MERGE\_PATCH()** discards values for all duplicate keys except the last, as shown here:

mysql> **SELECT**

-> **JSON\_MERGE\_PRESERVE('1', '2') AS Preserve,**

-> **JSON\_MERGE\_PATCH('1', '2') AS Patch\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Preserve: [1, 2]

Patch: 2

Array and object values are merged by autowrapping the object as an array and merging the arrays by combining values or by “last duplicate key wins” according to the choice of merging function (**JSON\_MERGE\_PRESERVE()** or **JSON\_MERGE\_PATCH()**, respectively), as can be seen in this example:

mysql> **SELECT**

-> **JSON\_MERGE\_PRESERVE('[10, 20]', '{"a": "x", "b": "y"}') AS Preserve,**

-> **JSON\_MERGE\_PATCH('[10, 20]', '{"a": "x", "b": "y"}') AS Patch\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Preserve: [10, 20, {"a": "x", "b": "y"}]

Patch: {"a": "x", "b": "y"}

### Searching and Modifying JSON Values

A JSON path expression selects a value within a JSON document.

Path expressions are useful with functions that extract parts of or modify a JSON document, to specify where within that document to operate. For example, the following query extracts from a JSON document the value of the member with the **name** key:

mysql> **SELECT JSON\_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name');**

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

| JSON\_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name') |

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

| "Aztalan" |

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

Path syntax uses a leading **$** character to represent the JSON document under consideration, optionally followed by selectors that indicate successively more specific parts of the document:

A period followed by a key name names the member in an object with the given key. The key name must be specified within double quotation marks if the name without quotes is not legal within path expressions (for example, if it contains a space).

**[*N*]** appended to a ***path*** that selects an array names the value at position ***N*** within the array. Array positions are integers beginning with zero. If ***path*** does not select an array value, ***path***[0] evaluates to the same value as ***path***:

mysql> **SELECT JSON\_SET('"x"', '$[0]', 'a');**

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

| JSON\_SET('"x"', '$[0]', 'a') |

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

| "a" |

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

1 row in set (0.00 sec)

**[*M* to *N*]** specifies a subset or range of array values starting with the value at position ***M***, and ending with the value at position ***N***.

**last** is supported as a synonym for the index of the rightmost array element. Relative addressing of array elements is also supported. If ***path*** does not select an array value, ***path***[last] evaluates to the same value as ***path***, as shown later in this section (see [Rightmost array element](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-paths-last)).

Paths can contain **\*** or **\*\*** wildcards:

**.[\*]** evaluates to the values of all members in a JSON object.

**[\*]** evaluates to the values of all elements in a JSON array.

***prefix*\*\**suffix*** evaluates to all paths that begin with the named prefix and end with the named suffix.

A path that does not exist in the document (evaluates to nonexistent data) evaluates to **NULL**.

Let **$** refer to this JSON array with three elements:

[3, {"a": [5, 6], "b": 10}, [99, 100]]

Then:

**$[0]** evaluates to **3**.

**$[1]** evaluates to **{"a": [5, 6], "b": 10}**.

**$[2]** evaluates to **[99, 100]**.

**$[3]** evaluates to **NULL** (it refers to the fourth array element, which does not exist).

Because **$[1]** and **$[2]** evaluate to nonscalar values, they can be used as the basis for more-specific path expressions that select nested values. Examples:

**$[1].a** evaluates to **[5, 6]**.

**$[1].a[1]** evaluates to **6**.

**$[1].b** evaluates to **10**.

**$[2][0]** evaluates to **99**.

As mentioned previously, path components that name keys must be quoted if the unquoted key name is not legal in path expressions. Let **$** refer to this value:

{"a fish": "shark", "a bird": "sparrow"}

The keys both contain a space and must be quoted:

**$."a fish"** evaluates to **shark**.

**$."a bird"** evaluates to **sparrow**.

Paths that use wildcards evaluate to an array that can contain multiple values:

mysql> **SELECT JSON\_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.\*');**

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

| JSON\_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.\*') |

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

| [1, 2, [3, 4, 5]] |

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

mysql> **SELECT JSON\_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.c[\*]');**

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

| JSON\_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.c[\*]') |

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

| [3, 4, 5] |

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

In the following example, the path **$\*\*.b** evaluates to multiple paths (**$.a.b** and **$.c.b**) and produces an array of the matching path values:

mysql> **SELECT JSON\_EXTRACT('{"a": {"b": 1}, "c": {"b": 2}}', '$\*\*.b');**

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

| JSON\_EXTRACT('{"a": {"b": 1}, "c": {"b": 2}}', '$\*\*.b') |

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

| [1, 2] |

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

**Ranges from JSON arrays.** You can use ranges with the **to** keyword to specify subsets of JSON arrays. For example, **$[1 to 3]** includes the second, third, and fourth elements of an array, as shown here:

mysql> **SELECT JSON\_EXTRACT('[1, 2, 3, 4, 5]', '$[1 to 3]');**

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

| JSON\_EXTRACT('[1, 2, 3, 4, 5]', '$[1 to 3]') |

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

| [2, 3, 4] |

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

1 row in set (0.00 sec)

The syntax is ***M* to *N***, where ***M*** and ***N*** are, respectively, the first and last indexes of a range of elements from a JSON array. ***N*** must be greater than ***M***; ***M*** must be greater than or equal to 0. Array elements are indexed beginning with 0.

You can use ranges in contexts where wildcards are supported.

**Rightmost array element.** The **last** keyword is supported as a synonym for the index of the last element in an array. Expressions of the form **last - *N*** can be used for relative addressing, and within range definitions, like this:

mysql> **SELECT JSON\_EXTRACT('[1, 2, 3, 4, 5]', '$[last-3 to last-1]');**

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

| JSON\_EXTRACT('[1, 2, 3, 4, 5]', '$[last-3 to last-1]') |

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

| [2, 3, 4] |

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

1 row in set (0.01 sec)

If the path is evaluated against a value that is not an array, the result of the evaluation is the same as if the value had been wrapped in a single-element array:

mysql> SELECT JSON\_REPLACE('"Sakila"', '$[last]', 10);

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

| JSON\_REPLACE('"Sakila"', '$[last]', 10) |

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

| 10 |

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

1 row in set (0.00 sec)

You can use [***column*->*path***](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_json-column-path) with a JSON column identifier and JSON path expression as a synonym for [**JSON\_EXTRACT(*column*, *path*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-extract). See [Section 12.18.3, “Functions That Search JSON Values”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#json-search-functions), for more information. See also [Indexing a Generated Column to Provide a JSON Column Index](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#json-column-indirect-index).

Some functions take an existing JSON document, modify it in some way, and return the resulting modified document. Path expressions indicate where in the document to make changes. For example, the [**JSON\_SET()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-set), [**JSON\_INSERT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-insert), and [**JSON\_REPLACE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-replace) functions each take a JSON document, plus one or more path-value pairs that describe where to modify the document and the values to use. The functions differ in how they handle existing and nonexisting values within the document.

Consider this document:

mysql> SET @j = '["a", {"b": [true, false]}, [10, 20]]';

[**JSON\_SET()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-set) replaces values for paths that exist and adds values for paths that do not exist:.

mysql> **SELECT JSON\_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2);**

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

| JSON\_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2) |

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

| ["a", {"b": [1, false]}, [10, 20, 2]] |

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

In this case, the path **$[1].b[0]** selects an existing value (**true**), which is replaced with the value following the path argument (**1**). The path **$[2][2]** does not exist, so the corresponding value (**2**) is added to the value selected by **$[2]**.

[**JSON\_INSERT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-insert) adds new values but does not replace existing values:

mysql> **SELECT JSON\_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2);**

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

| JSON\_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2) |

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

| ["a", {"b": [true, false]}, [10, 20, 2]] |

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

[**JSON\_REPLACE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-replace) replaces existing values and ignores new values:

mysql> **SELECT JSON\_REPLACE(@j, '$[1].b[0]', 1, '$[2][2]', 2);**

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

| JSON\_REPLACE(@j, '$[1].b[0]', 1, '$[2][2]', 2) |

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

| ["a", {"b": [1, false]}, [10, 20]] |

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

The path-value pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

**JSON\_REMOVE()** takes a JSON document and one or more paths that specify values to be removed from the document. The return value is the original document minus the values selected by paths that exist within the document:

mysql> **SELECT JSON\_REMOVE(@j, '$[2]', '$[1].b[1]', '$[1].b[1]');**

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

| JSON\_REMOVE(@j, '$[2]', '$[1].b[1]', '$[1].b[1]') |

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

| ["a", {"b": [true]}] |

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

The paths have these effects:

**$[2]** matches **[10, 20]** and removes it.

The first instance of **$[1].b[1]** matches **false** in the **b** element and removes it.

The second instance of **$[1].b[1]** matches nothing: That element has already been removed, the path no longer exists, and has no effect.

### JSON Path Syntax

Many of the JSON functions supported by MySQL and described elsewhere in this Manual (see [Section 12.18, “JSON Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#json-functions)) require a path expression in order to identify a specific element in a JSON document. A path consists of the path's scope followed by one or more path legs. For paths used in MySQL JSON functions, the scope is always the document being searched or otherwise operated on, represented by a leading **$** character. Path legs are separated by period characters (**.**). Cells in arrays are represented by **[*N*]**, where ***N*** is a non-negative integer. Names of keys must be double-quoted strings or valid ECMAScript identifiers (see [Identifier Names and Identifiers](http://www.ecma-international.org/ecma-262/5.1/#sec-7.6), in the ECMAScript Language Specification). Path expressions, like JSON text, should be encoded using the **ascii**, **utf8**, or **utf8mb4** character set. Other character encodings are implicitly coerced to **utf8mb4**. The complete syntax is shown here:

***pathExpression***:

***scope***[(***pathLeg***)\*]

***pathLeg***:

***member*** | ***arrayLocation*** | ***doubleAsterisk***

***member***:

***period*** ( ***keyName*** | ***asterisk*** )

***arrayLocation***:

***leftBracket*** ( ***nonNegativeInteger*** | ***asterisk*** ) ***rightBracket***

***keyName***:

[***ESIdentifier***](http://www.ecma-international.org/ecma-262/5.1/#sec-7.6) | ***doubleQuotedString***

***doubleAsterisk***:

'\*\*'

***period***:

'.'

***asterisk***:

'\*'

***leftBracket***:

'['

***rightBracket***:

']'

As noted previously, in MySQL, the scope of the path is always the document being operated on, represented as **$**. You can use **'$'** as a synonynm for the document in JSON path expressions.

**Note**

Some implementations support column references for scopes of JSON paths; currently, MySQL does not support these.

The wildcard **\*** and **\*\*** tokens are used as follows:

**.\*** represents the values of all members in the object.

**[\*]** represents the values of all cells in the array.

**[*prefix*]\*\**suffix*** represents all paths beginning with ***prefix*** and ending with ***suffix***. ***prefix*** is optional, while ***suffix*** is required; in other words, a path may not end in **\*\***.

In addition, a path may not contain the sequence **\*\*\***.

For path syntax examples, see the descriptions of the various JSON functions that take paths as arguments, such as [**JSON\_CONTAINS\_PATH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-contains-path), [**JSON\_SET()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-set), and [**JSON\_REPLACE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-replace). For examples which include the use of the **\*** and **\*\*** wildcards, see the description of the [**JSON\_SEARCH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-search) function.

MySQL 8.0.2 and later also supports range notation for subsets of JSON arrays using the **to** keyword (such as **$[2 to 10]**), as well as the **last** keyword as a synonym for the rightmost element of an array. See [Searching and Modifying JSON Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json-paths), for more information and examples.

### Comparison and Ordering of JSON Values

JSON values can be compared using the [**=**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_equal), [**<**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_less-than), [**<=**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_less-than-or-equal), [**>**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_greater-than), [**>=**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_greater-than-or-equal), [**<>**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_not-equal), [**!=**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_not-equal), and [**<=>**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_equal-to) operators.

The following comparison operators and functions are not yet supported with JSON values:

[**BETWEEN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_between)

[**IN()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_in)

[**GREATEST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_greatest)

[**LEAST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_least)

A workaround for the comparison operators and functions just listed is to cast JSON values to a native MySQL numeric or string data type so they have a consistent non-JSON scalar type.

Comparison of JSON values takes place at two levels. The first level of comparison is based on the JSON types of the compared values. If the types differ, the comparison result is determined solely by which type has higher precedence. If the two values have the same JSON type, a second level of comparison occurs using type-specific rules.

The following list shows the precedences of JSON types, from highest precedence to the lowest. (The type names are those returned by the [**JSON\_TYPE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_json-type) function.) Types shown together on a line have the same precedence. Any value having a JSON type listed earlier in the list compares greater than any value having a JSON type listed later in the list.

BLOB

BIT

OPAQUE

DATETIME

TIME

DATE

BOOLEAN

ARRAY

OBJECT

STRING

INTEGER, DOUBLE

NULL

For JSON values of the same precedence, the comparison rules are type specific:

**BLOB**

The first ***N*** bytes of the two values are compared, where ***N*** is the number of bytes in the shorter value. If the first ***N*** bytes of the two values are identical, the shorter value is ordered before the longer value.

**BIT**

Same rules as for **BLOB**.

**OPAQUE**

Same rules as for **BLOB**. **OPAQUE** values are values that are not classified as one of the other types.

**DATETIME**

A value that represents an earlier point in time is ordered before a value that represents a later point in time. If two values originally come from the MySQL **DATETIME** and **TIMESTAMP** types, respectively, they are equal if they represent the same point in time.

**TIME**

The smaller of two time values is ordered before the larger one.

**DATE**

The earlier date is ordered before the more recent date.

**ARRAY**

Two JSON arrays are equal if they have the same length and values in corresponding positions in the arrays are equal.

If the arrays are not equal, their order is determined by the elements in the first position where there is a difference. The array with the smaller value in that position is ordered first. If all values of the shorter array are equal to the corresponding values in the longer array, the shorter array is ordered first.

Example:

[] < ["a"] < ["ab"] < ["ab", "cd", "ef"] < ["ab", "ef"]

**BOOLEAN**

The JSON false literal is less than the JSON true literal.

**OBJECT**

Two JSON objects are equal if they have the same set of keys, and each key has the same value in both objects.

Example:

{"a": 1, "b": 2} = {"b": 2, "a": 1}

The order of two objects that are not equal is unspecified but deterministic.

**STRING**

Strings are ordered lexically on the first ***N*** bytes of the **utf8mb4** representation of the two strings being compared, where ***N*** is the length of the shorter string. If the first ***N*** bytes of the two strings are identical, the shorter string is considered smaller than the longer string.

Example:

"a" < "ab" < "b" < "bc"

This ordering is equivalent to the ordering of SQL strings with collation **utf8mb4\_bin**. Because **utf8mb4\_bin** is a binary collation, comparison of JSON values is case-sensitive:

"A" < "a"

**INTEGER**, **DOUBLE**

JSON values can contain exact-value numbers and approximate-value numbers. For a general discussion of these types of numbers, see [Section 9.1.2, “Numeric Literals”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#number-literals).

The rules for comparing native MySQL numeric types are discussed in [Section 12.3, “Type Conversion in Expression Evaluation”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#type-conversion), but the rules for comparing numbers within JSON values differ somewhat:

In a comparison between two columns that use the native MySQL [**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) and [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) numeric types, respectively, it is known that all comparisons involve an integer and a double, so the integer is converted to double for all rows. That is, exact-value numbers are converted to approximate-value numbers.

On the other hand, if the query compares two JSON columns containing numbers, it cannot be known in advance whether numbers are integer or double. To provide the most consistent behavior across all rows, MySQL converts approximate-value numbers to exact-value numbers. The resulting ordering is consistent and does not lose precision for the exact-value numbers. For example, given the scalars 9223372036854775805, 9223372036854775806, 9223372036854775807 and 9.223372036854776e18, the order is such as this:

9223372036854775805 < 9223372036854775806 < 9223372036854775807

< 9.223372036854776e18 = 9223372036854776000 < 9223372036854776001

Were JSON comparisons to use the non-JSON numeric comparison rules, inconsistent ordering could occur. The usual MySQL comparison rules for numbers yield these orderings:

Integer comparison:

9223372036854775805 < 9223372036854775806 < 9223372036854775807

(not defined for 9.223372036854776e18)

Double comparison:

9223372036854775805 = 9223372036854775806 = 9223372036854775807 = 9.223372036854776e18

For comparison of any JSON value to SQL **NULL**, the result is **UNKNOWN**.

For comparison of JSON and non-JSON values, the non-JSON value is converted to JSON according to the rules in the following table, then the values compared as described previously.

### Converting between JSON and non-JSON values

The following table provides a summary of the rules that MySQL follows when casting between JSON values and values of other types:

**Table 11.3 JSON Conversion Rules**

| **other type** | **CAST(other type AS JSON)** | **CAST(JSON AS other type)** |
| --- | --- | --- |
| **JSON** | No change | No change |
| **utf8 character type (utf8mb4, utf8, ascii)** | The string is parsed into a JSON value. | The JSON value is serialized into a **utf8mb4** string. |
| **Other character types** | Other character encodings are implicitly converted to **utf8mb4** and treated as described for utf8 character type. | The JSON value is serialized into a **utf8mb4** string, then cast to the other character encoding. The result may not be meaningful. |
| **NULL** | Results in a **NULL** value of type JSON. | Not applicable. |
| **Geometry types** | The geometry value is converted into a JSON document by calling **[ST\_AsGeoJSON()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-asgeojson)**. | Illegal operation. Workaround: Pass the result of [**CAST(*json\_val* AS CHAR)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_cast) to **[ST\_GeomFromGeoJSON()](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\functions.html" \l "function_st-geomfromgeojson)**. |
| **All other types** | Results in a JSON document consisting of a single scalar value. | Succeeds if the JSON document consists of a single scalar value of the target type and that scalar value can be cast to the target type. Otherwise, returns **NULL** and produces a warning. |

**ORDER BY** and **GROUP BY** for JSON values works according to these principles:

Ordering of scalar JSON values uses the same rules as in the preceding discussion.

For ascending sorts, SQL **NULL** orders before all JSON values, including the JSON null literal; for descending sorts, SQL **NULL** orders after all JSON values, including the JSON null literal.

Sort keys for JSON values are bound by the value of the **[max\_sort\_length](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_sort_length)** system variable, so keys that differ only after the first **[max\_sort\_length](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_sort_length)** bytes compare as equal.

Sorting of nonscalar values is not currently supported and a warning occurs.

For sorting, it can be beneficial to cast a JSON scalar to some other native MySQL type. For example, if a column named **jdoc** contains JSON objects having a member consisting of an **id** key and a nonnegative value, use this expression to sort by **id** values:

ORDER BY CAST(JSON\_EXTRACT(jdoc, '$.id') AS UNSIGNED)

If there happens to be a generated column defined to use the same expression as in the **ORDER BY**, the MySQL optimizer recognizes that and considers using the index for the query execution plan. See [Section 8.3.11, “Optimizer Use of Generated Column Indexes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#generated-column-index-optimizations).

### Aggregation of JSON Values

For aggregation of JSON values, SQL **NULL** values are ignored as for other data types. Non-**NULL** values are converted to a numeric type and aggregated, except for [**MIN()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_min), [**MAX()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_max), and [**GROUP\_CONCAT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_group-concat). The conversion to number should produce a meaningful result for JSON values that are numeric scalars, although (depending on the values) truncation and loss of precision may occur. Conversion to number of other JSON values may not produce a meaningful result.

## 11.6 Data Type Default Values

Data type specifications can have explicit or implicit default values.

A **DEFAULT *value*** clause in a data type specification explicitly indicates a default value for a column. Examples:

CREATE TABLE t1 (

i INT DEFAULT -1,

c VARCHAR(10) DEFAULT '',

price DOUBLE(16,2) DEFAULT 0.00

);

**SERIAL DEFAULT VALUE** is a special case. In the definition of an integer column, it is an alias for **NOT NULL AUTO\_INCREMENT UNIQUE**.

Some aspects of explicit **DEFAULT** clause handling are version dependent, as described following.

[Explicit Default Handling as of MySQL 8.0.13](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-type-defaults-explicit)

[Explicit Default Handling Prior to MySQL 8.0.13](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-type-defaults-explicit-old)

[Implicit Default Handling](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-type-defaults-implicit)

### Explicit Default Handling as of MySQL 8.0.13

The default value specified in a **DEFAULT** clause can be a literal constant or an expression. With one exception, enclose expression default values within parentheses to distinguish them from literal constant default values. Examples:

CREATE TABLE t1 (

-- literal defaults

i INT DEFAULT 0,

c VARCHAR(10) DEFAULT '',

-- expression defaults

f FLOAT DEFAULT (RAND() \* RAND()),

b BINARY(16) DEFAULT (UUID\_TO\_BIN(UUID())),

d DATE DEFAULT (CURRENT\_DATE + INTERVAL 1 YEAR),

p POINT DEFAULT (Point(0,0)),

j JSON DEFAULT (JSON\_ARRAY())

);

The exception is that, for [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns, you can specify the [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp) function as the default, without enclosing parentheses. See [Section 11.2.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#timestamp-initialization).

The [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), **GEOMETRY**, and [**JSON**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json) data types can be assigned a default value only if the value is written as an expression, even if the expression value is a literal:

This is permitted (literal default specified as expression):

CREATE TABLE t2 (b BLOB DEFAULT ('abc'));

This produces an error (literal default not specified as expression):

CREATE TABLE t2 (b BLOB DEFAULT 'abc');

Expression default values must adhere to the following rules. An error occurs if an expression contains disallowed constructs.

Literals, built-in functions (both deterministic and nondeterministic), and operators are permitted.

Subqueries, parameters, variables, stored functions, and user-defined functions are not permitted.

An expression default value cannot depend on a column that has the **AUTO\_INCREMENT** attribute.

An expression default value for one column can refer to other table columns, with the exception that references to generated columns or columns with expression default values must be to columns that occur earlier in the table definition. That is, expression default values cannot contain forward references to generated columns or columns with expression default values.

The ordering constraint also applies to the use of [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table) to reorder table columns. If the resulting table would have an expression default value that contains a forward reference to a generated column or column with an expression default value, the statement fails.

**Note**

If any component of an expression default value depends on the SQL mode, different results may occur for different uses of the table unless the SQL mode is the same during all uses.

For [**CREATE TABLE ... LIKE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table-like) and [**CREATE TABLE ... SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table-select), the destination table preserves expression default values from the original table.

If an expression default value refers to a nondeterministic function, any statement that causes the expression to be evaluated is unsafe for statement-based replication. This includes statements such as [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) and [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update). In this situation, if binary logging is disabled, the statement is executed as normal. If binary logging is enabled and **[binlog\_format](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_binlog_format)** is set to **STATEMENT**, the statement is logged and executed but a warning message is written to the error log, because replication slaves might diverge. When **[binlog\_format](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_binlog_format)** is set to **MIXED** or **ROW**, the statement is executed as normal.

When inserting a new row, the default value for a column with an expression default can be inserted either by omitting the column name or by specifying the column as **DEFAULT** (just as for columns with literal defaults):

mysql> **CREATE TABLE t4 (uid BINARY(16) DEFAULT (UUID\_TO\_BIN(UUID())));**

mysql> **INSERT INTO t4 () VALUES();**

mysql> **INSERT INTO t4 () VALUES(DEFAULT);**

mysql> **SELECT BIN\_TO\_UUID(uid) AS uid FROM t4;**

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

| uid |

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

| f1109174-94c9-11e8-971d-3bf1095aa633 |

| f110cf9a-94c9-11e8-971d-3bf1095aa633 |

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

However, the use of [**DEFAULT(*col\_name*)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_default) to specify the default value for a named column is permitted only for columns that have a literal default value, not for columns that have an expression default value.

Not all storage engines permit expression default values. For those that do not, an [**ER\_UNSUPPORTED\_ACTION\_ON\_DEFAULT\_VAL\_GENERATED**](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unsupported_action_on_default_val_generated) error occurs.

If a default value evaluates to a data type that differs from the declared column type, implicit coercion to the declared type occurs according to the usual MySQL type-conversion rules. See [Section 12.3, “Type Conversion in Expression Evaluation”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#type-conversion).

### Explicit Default Handling Prior to MySQL 8.0.13

With one exception, the default value specified in a **DEFAULT** clause must be a literal constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as [**NOW()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_now) or [**CURRENT\_DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-date). The exception is that, for [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) and [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns, you can specify [**CURRENT\_TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-timestamp) as the default. See [Section 11.2.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#timestamp-initialization).

The [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), **GEOMETRY**, and [**JSON**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json) data types cannot be assigned a default value.

If a default value evaluates to a data type that differs from the declared column type, implicit coercion to the declared type occurs according to the usual MySQL type-conversion rules. See [Section 12.3, “Type Conversion in Expression Evaluation”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#type-conversion).

### Implicit Default Handling

If a data type specification includes no explicit **DEFAULT** value, MySQL determines the default value as follows:

If the column can take **NULL** as a value, the column is defined with an explicit **DEFAULT NULL** clause.

If the column cannot take **NULL** as a value, MySQL defines the column with no explicit **DEFAULT** clause.

For data entry into a **NOT NULL** column that has no explicit **DEFAULT** clause, if an [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) or [**REPLACE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#replace) statement includes no value for the column, or an [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) statement sets the column to **NULL**, MySQL handles the column according to the SQL mode in effect at the time:

If strict SQL mode is enabled, an error occurs for transactional tables and the statement is rolled back. For nontransactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows are inserted.

If strict mode is not enabled, MySQL sets the column to the implicit default value for the column data type.

Suppose that a table **t** is defined as follows:

CREATE TABLE t (i INT NOT NULL);

In this case, **i** has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit default is inserted for the first two statements, but the third fails because [**DEFAULT(i)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_default) cannot produce a value:

INSERT INTO t VALUES();

INSERT INTO t VALUES(DEFAULT);

INSERT INTO t VALUES(DEFAULT(i));

See [Section 5.1.11, “Server SQL Modes”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sql-mode).

For a given table, the [**SHOW CREATE TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-table) statement displays which columns have an explicit **DEFAULT** clause.

Implicit defaults are defined as follows:

For numeric types, the default is **0**, with the exception that for integer or floating-point types declared with the **AUTO\_INCREMENT** attribute, the default is the next value in the sequence.

For date and time types other than [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), the default is the appropriate “zero” value for the type. This is also true for [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) if the **[explicit\_defaults\_for\_timestamp](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_explicit_defaults_for_timestamp)** system variable is enabled (see [Section 5.1.8, “Server System Variables”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#server-system-variables)). Otherwise, for the first [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) column in a table, the default value is the current date and time. See [Section 11.2, “Date and Time Data Types”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#date-and-time-types).

For string types other than [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum), the default value is the empty string. For [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum), the default is the first enumeration value.

## 11.7 Data Type Storage Requirements

[InnoDB Table Storage Requirements](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "data-types-storage-reqs-innodb" \o "InnoDB Table Storage Requirements)

[NDB Table Storage Requirements](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-types-storage-reqs-ndb)

[Numeric Type Storage Requirements](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-types-storage-reqs-numeric)

[Date and Time Type Storage Requirements](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-types-storage-reqs-date-time)

[String Type Storage Requirements](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-types-storage-reqs-strings)

[Spatial Type Storage Requirements](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-types-storage-reqs-gis)

[JSON Storage Requirements](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#data-types-storage-reqs-json)

The storage requirements for table data on disk depend on several factors. Different storage engines represent data types and store raw data differently. Table data might be compressed, either for a column or an entire row, complicating the calculation of storage requirements for a table or column.

Despite differences in storage layout on disk, the internal MySQL APIs that communicate and exchange information about table rows use a consistent data structure that applies across all storage engines.

This section includes guidelines and information for the storage requirements for each data type supported by MySQL, including the internal format and size for storage engines that use a fixed-size representation for data types. Information is listed by category or storage engine.

The internal representation of a table has a maximum row size of 65,535 bytes, even if the storage engine is capable of supporting larger rows. This figure excludes [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) or [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) columns, which contribute only 9 to 12 bytes toward this size. For [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) and [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) data, the information is stored internally in a different area of memory than the row buffer. Different storage engines handle the allocation and storage of this data in different ways, according to the method they use for handling the corresponding types. For more information, see [Chapter 16, *Alternative Storage Engines*](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\storage-engines.html), and [Section 8.4.7, “Limits on Table Column Count and Row Size”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#column-count-limit).

### InnoDB Table Storage Requirements

See [Section 15.10, “InnoDB Row Formats”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\innodb-storage-engine.html#innodb-row-format) for information about storage requirements for **InnoDB** tables.

### NDB Table Storage Requirements

**Important**

[**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) tables use 4-byte alignment; all [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) data storage is done in multiples of 4 bytes. Thus, a column value that would typically take 15 bytes requires 16 bytes in an [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) table. For example, in [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) tables, the [**TINYINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), [**SMALLINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), [**MEDIUMINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), and [**INTEGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) ([**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types)) column types each require 4 bytes storage per record due to the alignment factor.

Each **BIT(*M*)** column takes ***M*** bits of storage space. Although an individual [**BIT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#bit-type) column is not 4-byte aligned, [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) reserves 4 bytes (32 bits) per row for the first 1-32 bits needed for **BIT** columns, then another 4 bytes for bits 33-64, and so on.

While a **NULL** itself does not require any storage space, [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) reserves 4 bytes per row if the table definition contains any columns allowing **NULL**, up to 32 **NULL** columns. (If an NDB Cluster table is defined with more than 32 **NULL** columns up to 64 **NULL** columns, then 8 bytes per row are reserved.)

Every table using the [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) storage engine requires a primary key; if you do not define a primary key, a “hidden” primary key is created by [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html). This hidden primary key consumes 31-35 bytes per table record.

You can use the [**ndb\_size.pl**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html#mysql-cluster-programs-ndb-size-pl) Perl script to estimate [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) storage requirements. It connects to a current MySQL (not NDB Cluster) database and creates a report on how much space that database would require if it used the [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) storage engine. See [Section 23.4.28, “ndb\_size.pl — NDBCLUSTER Size Requirement Estimator”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html#mysql-cluster-programs-ndb-size-pl) for more information.

### Numeric Type Storage Requirements

| **Data Type** | **Storage Required** |
| --- | --- |
| [**TINYINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | 1 byte |
| [**SMALLINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | 2 bytes |
| [**MEDIUMINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | 3 bytes |
| [**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types), [**INTEGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | 4 bytes |
| [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | 8 bytes |
| **FLOAT(*p*)** | 4 bytes if 0 <= ***p*** <= 24, 8 bytes if 25 <= ***p*** <= 53 |
| [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) | 4 bytes |
| **DOUBLE [PRECISION]**, [**REAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) | 8 bytes |
| **DECIMAL(*M*,*D*)**, **NUMERIC(*M*,*D*)** | Varies; see following discussion |
| **BIT(*M*)** | approximately (***M***+7)/8 bytes |

Values for [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) (and [**NUMERIC**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types)) columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table.

| **Leftover Digits** | **Number of Bytes** |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 3 |
| 6 | 3 |
| 7 | 4 |
| 8 | 4 |

### Date and Time Type Storage Requirements

For [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) columns, the storage required for tables created before MySQL 5.6.4 differs from tables created from 5.6.4 on. This is due to a change in 5.6.4 that permits these types to have a fractional part, which requires from 0 to 3 bytes.

| **Data Type** | **Storage Required Before MySQL 5.6.4** | **Storage Required as of MySQL 5.6.4** |
| --- | --- | --- |
| [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) | 1 byte | 1 byte |
| [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) | 3 bytes | 3 bytes |
| [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) | 3 bytes | 3 bytes + fractional seconds storage |
| [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) | 8 bytes | 5 bytes + fractional seconds storage |
| [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) | 4 bytes | 4 bytes + fractional seconds storage |

As of MySQL 5.6.4, storage for [**YEAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#year) and [**DATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) remains unchanged. However, [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime), and [**TIMESTAMP**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) are represented differently. [**DATETIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#datetime) is packed more efficiently, requiring 5 rather than 8 bytes for the nonfractional part, and all three parts have a fractional part that requires from 0 to 3 bytes, depending on the fractional seconds precision of stored values.

| **Fractional Seconds Precision** | **Storage Required** |
| --- | --- |
| 0 | 0 bytes |
| 1, 2 | 1 byte |
| 3, 4 | 2 bytes |
| 5, 6 | 3 bytes |

For example, [**TIME(0)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**TIME(2)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), [**TIME(4)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time), and [**TIME(6)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) use 3, 4, 5, and 6 bytes, respectively. [**TIME**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) and [**TIME(0)**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#time) are equivalent and require the same storage.

For details about internal representation of temporal values, see [MySQL Internals: Important Algorithms and Structures](https://dev.mysql.com/doc/internals/en/algorithms.html).

### String Type Storage Requirements

In the following table, ***M*** represents the declared column length in characters for nonbinary string types and bytes for binary string types. ***L*** represents the actual length in bytes of a given string value.

| **Data Type** | **Storage Required** |
| --- | --- |
| **CHAR(*M*)** | The compact family of InnoDB row formats optimize storage for variable-length character sets. See [COMPACT Row Format Storage Characteristics](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\innodb-storage-engine.html#innodb-compact-row-format-characteristics). Otherwise, ***M*** × ***w*** bytes, **<= *M* <=** 255, where ***w*** is the number of bytes required for the maximum-length character in the character set. |
| **BINARY(*M*)** | ***M*** bytes, 0 **<= *M* <=** 255 |
| **VARCHAR(*M*)**, **VARBINARY(*M*)** | ***L*** + 1 bytes if column values require 0 − 255 bytes, ***L*** + 2 bytes if values may require more than 255 bytes |
| [**TINYBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**TINYTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) | ***L*** + 1 bytes, where ***L*** < 28 |
| [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) | ***L*** + 2 bytes, where ***L*** < 216 |
| [**MEDIUMBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**MEDIUMTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) | ***L*** + 3 bytes, where ***L*** < 224 |
| [**LONGBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob), [**LONGTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) | ***L*** + 4 bytes, where ***L*** < 232 |
| **ENUM('*value1*','*value2*',...)** | 1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum) |
| **SET('*value1*','*value2*',...)** | 1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum) |

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is ***L*** (the byte length of the string). For example, storage for a [**MEDIUMTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) value requires ***L*** bytes to store the value plus three bytes to store the length of the value.

To calculate the number of bytes used to store a particular [**CHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char), or [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) column value, you must take into account the character set used for that column and whether the value contains multibyte characters. In particular, when using a **utf8** Unicode character set, you must keep in mind that not all characters use the same number of bytes. **utf8mb3** and **utf8mb4** character sets can require up to three and four bytes per character, respectively. For a breakdown of the storage used for different categories of **utf8mb3** or **utf8mb4** characters, see [Section 10.9, “Unicode Support”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-unicode).

**[VARCHAR](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\data-types.html" \l "char" \o "11.3.2 The CHAR and VARCHAR Types)**, [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary), and the [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) and [**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) types are variable-length types. For each, the storage requirements depend on these factors:

The actual length of the column value

The column's maximum possible length

The character set used for the column, because some character sets contain multibyte characters

For example, a **VARCHAR(255)** column can hold a string with a maximum length of 255 characters. Assuming that the column uses the **latin1** character set (one byte per character), the actual storage required is the length of the string (***L***), plus one byte to record the length of the string. For the string **'abcd'**, ***L*** is 4 and the storage requirement is five bytes. If the same column is instead declared to use the **ucs2** double-byte character set, the storage requirement is 10 bytes: The length of **'abcd'** is eight bytes and the column requires two bytes to store lengths because the maximum length is greater than 255 (up to 510 bytes).

The effective maximum number of bytes that can be stored in a [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) or [**VARBINARY**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#binary-varbinary) column is subject to the maximum row size of 65,535 bytes, which is shared among all columns. For a [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) column that stores multibyte characters, the effective maximum number of characters is less. For example, **utf8mb4** characters can require up to four bytes per character, so a [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) column that uses the **utf8mb4** character set can be declared to be a maximum of 16,383 characters. See [Section 8.4.7, “Limits on Table Column Count and Row Size”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#column-count-limit).

**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**.

The [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) storage engine supports variable-width columns. This means that a [**VARCHAR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#char) column in an NDB Cluster table requires the same amount of storage as would any other storage engine, with the exception that such values are 4-byte aligned. Thus, the string **'abcd'** stored in a **VARCHAR(50)** column using the **latin1** character set requires 8 bytes (rather than 5 bytes for the same column value in a **MyISAM** table).

[**TEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) and [**BLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) columns are implemented differently in [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html); each row in a **TEXT** column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The rows in this second table are always 2000 bytes long. This means that the size of a **TEXT** column is 256 if ***size*** <= 256 (where ***size*** represents the size of the row); otherwise, the size is 256 + ***size*** + (2000 × (***size*** − 256) % 2000).

The size of an [**ENUM**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum) object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See [Section 11.3.5, “The ENUM Type”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#enum).

The size of a [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) object is determined by the number of different set members. If the set size is ***N***, the object occupies **(*N*+7)/8** bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set) can have a maximum of 64 members. See [Section 11.3.6, “The SET Type”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#set).

### Spatial Type Storage Requirements

MySQL stores geometry values using 4 bytes to indicate the SRID followed by the WKB representation of the value. The [**LENGTH()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_length) function returns the space in bytes required for value storage.

For descriptions of WKB and internal storage formats for spatial values, see [Section 11.4.3, “Supported Spatial Data Formats”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#gis-data-formats).

### JSON Storage Requirements

In general, the storage requirement for a [**JSON**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#json) column is approximately the same as for a **LONGBLOB** or **LONGTEXT** column; that is, the space consumed by a JSON document is roughly the same as it would be for the document's string representation stored in a column of one of these types. However, there is an overhead imposed by the binary encoding, including metadata and dictionaries needed for lookup, of the individual values stored in the JSON document. For example, a string stored in a JSON document requires 4 to 10 bytes additional storage, depending on the length of the string and the size of the object or array in which it is stored.

In addition, MySQL imposes a limit on the size of any JSON document stored in a **JSON** column such that it cannot be any larger than the value of **[max\_allowed\_packet](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_allowed_packet)**.

## 11.8 Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from **1** to **99999**, **MEDIUMINT UNSIGNED** is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

All basic calculations (**+**, **-**, **\***, and **/**) with [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) columns are done with precision of 65 decimal (base 10) digits. See [Section 11.1.1, “Numeric Data Type Syntax”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#numeric-type-syntax).

If accuracy is not too important or if speed is the highest priority, the [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) type may be good enough. For high precision, you can always convert to a fixed-point type stored in a [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types). This enables you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

## 11.9 Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL.

| **Other Vendor Type** | **MySQL Type** |
| --- | --- |
| [**BOOL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | [**TINYINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| [**BOOLEAN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | [**TINYINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| **CHARACTER VARYING(*M*)** | **VARCHAR(*M*)** |
| [**FIXED**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) | [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) |
| [**FLOAT4**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) | [**FLOAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) |
| [**FLOAT8**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) | [**DOUBLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#floating-point-types) |
| [**INT1**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | [**TINYINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| [**INT2**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | [**SMALLINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| [**INT3**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | [**MEDIUMINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| [**INT4**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | [**INT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| **INT8** | [**BIGINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| **LONG VARBINARY** | [**MEDIUMBLOB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) |
| **LONG VARCHAR** | [**MEDIUMTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) |
| **LONG** | [**MEDIUMTEXT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#blob) |
| [**MIDDLEINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) | [**MEDIUMINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#integer-types) |
| [**NUMERIC**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) | [**DECIMAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\data-types.html#fixed-point-types) |

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a **DESCRIBE *tbl\_name*** statement, MySQL reports the table structure using the equivalent MySQL types. For example:

mysql> **CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);**

Query OK, 0 rows affected (0.00 sec)

mysql> **DESCRIBE t;**

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

| Field | Type | Null | Key | Default | Extra |

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

| a | tinyint(1) | YES | | NULL | |

| b | double | YES | | NULL | |

| c | mediumtext | YES | | NULL | |

| d | decimal(10,0) | YES | | NULL | |

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

4 rows in set (0.01 sec)