语言结构

[9.7 Comments](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#comments)

This chapter discusses the rules for writing the following elements of [SQL](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\glossary.html#glos_sql) statements when using MySQL:

Literal values such as strings and numbers

Identifiers such as database, table, and column names

Keywords and reserved words

User-defined and system variables

Expressions

Query attributes

Comments

## 9.1 Literal Values

[9.1.1 String 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#string-literals)

[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)

[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)

[9.1.4 Hexadecimal 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#hexadecimal-literals)

[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)

[9.1.6 Boolean 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#boolean-literals)

[9.1.7 NULL Values](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#null-values)

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal and bit values, boolean values, and **NULL**. The section also covers various nuances that you may encounter when dealing with these basic types in MySQL.

### 9.1.1 String Literals

A string is a sequence of bytes or characters, enclosed within either single quote (**'**) or double quote (**"**) characters. Examples:

'a string'

"another string"

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

'a string'

'a' ' ' 'string'

If the [**ANSI\_QUOTES**](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_ansi_quotes) SQL mode is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

A binary string is a string of bytes. Every binary string has a character set and collation named **binary**. A nonbinary string is a string of characters. It has a character set other than **binary** and a collation that is compatible with the character set.

For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte; comparisons use numeric byte values. For nonbinary strings, the unit is the character and some character sets support multibyte characters; comparisons use numeric character code values. Character code ordering is a function of the string collation. (For more information, 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).)

A character string literal may have an optional character set introducer and **COLLATE** clause, to designate it as a string that uses a particular character set and collation:

[\_***charset\_name***]'***string***' [COLLATE ***collation\_name***]

Examples:

SELECT \_latin1'***string***';

SELECT \_binary'***string***';

SELECT \_utf8'***string***' COLLATE utf8\_danish\_ci;

You can use **N'*literal*'** (or **n'*literal*'**) to create a string in the national character set. These statements are equivalent:

SELECT N'some text';

SELECT n'some text';

SELECT \_utf8'some text';

For information about these forms of string syntax, see [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), and [Section 10.3.8, “Character Set Introducers”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-introducer).

Within a string, certain sequences have special meaning unless 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) SQL mode is enabled. Each of these sequences begins with a backslash (**\**), known as the escape character. MySQL recognizes the escape sequences shown in [Table 9.1, “Special Character Escape Sequences”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#character-escape-sequences). For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, **\x** is just **x**. These sequences are case-sensitive. For example, **\b** is interpreted as a backspace, but **\B** is interpreted as **B**. Escape processing is done according to the character set indicated by the **[character\_set\_connection](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_character_set_connection)** system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in [Section 10.3.6, “Character String Literal 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-literal).

**Table 9.1 Special Character Escape Sequences**

| **Escape Sequence** | **Character Represented by Sequence** |
| --- | --- |
| **\0** | An ASCII NUL (**X'00'**) character |
| **\'** | A single quote (**'**) character |
| **\"** | A double quote (**"**) character |
| **\b** | A backspace character |
| **\n** | A newline (linefeed) character |
| **\r** | A carriage return character |
| **\t** | A tab character |
| **\Z** | ASCII 26 (Control+Z); see note following the table |
| **\\** | A backslash (**\**) character |
| **\%** | A **%** character; see note following the table |
| **\\_** | A **\_** character; see note following the table |

The ASCII 26 character can be encoded as **\Z** to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use **mysql *db\_name* < *file\_name***.

The **\%** and **\\_** sequences are used to search for literal instances of **%** and **\_** in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of 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 in [Section 12.8.1, “String Comparison 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-comparison-functions). If you use **\%** or **\\_** outside of pattern-matching contexts, they evaluate to the strings **\%** and **\\_**, not to **%** and **\_**.

There are several ways to include quote characters within a string:

A **'** inside a string quoted with **'** may be written as **''**.

A **"** inside a string quoted with **"** may be written as **""**.

Precede the quote character by an escape character (**\**).

A **'** inside a string quoted with **"** needs no special treatment and need not be doubled or escaped. In the same way, **"** inside a string quoted with **'** needs no special treatment.

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) statements demonstrate how quoting and escaping work:

mysql> **SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';**

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

| hello | "hello" | ""hello"" | hel'lo | 'hello |

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

mysql> **SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";**

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

| hello | 'hello' | ''hello'' | hel"lo | "hello |

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

mysql> **SELECT 'This\nIs\nFour\nLines';**

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

| This

Is

Four

Lines |

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

mysql> **SELECT 'disappearing\ backslash';**

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

| disappearing backslash |

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

To insert binary data into a string column (such as 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), you should represent certain characters by escape sequences. Backslash (**\**) and the quote character used to quote the string must be escaped. In certain client environments, it may also be necessary to escape **NUL** or Control+Z. The **[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)** client truncates quoted strings containing **NUL** characters if they are not escaped, and Control+Z may be taken for END-OF-FILE on Windows if not escaped. For the escape sequences that represent each of these characters, see [Table 9.1, “Special Character Escape Sequences”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#character-escape-sequences).

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

Process the string with a function that escapes the special characters. In a C program, you can use the **[mysql\_real\_escape\_string\_quote()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-escape-string-quote.html" \t "_top)** C API function to escape characters. See [mysql\_real\_escape\_string\_quote()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-escape-string-quote.html" \t "_top). Within SQL statements that construct other SQL statements, you can use the [**QUOTE()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_quote) function. The Perl DBI interface provides a **quote** method to convert special characters to the proper escape sequences. See [Section 29.9, “MySQL Perl API”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\connectors-apis.html#apis-perl). Other language interfaces may provide a similar capability.

As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

### 9.1.2 Numeric Literals

Number literals include exact-value (integer 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)) literals and approximate-value (floating-point) literals.

Integers are represented as a sequence of digits. Numbers may include **.** as a decimal separator. Numbers may be preceded by **-** or **+** to indicate a negative or positive value, respectively. Numbers represented in scientific notation with a mantissa and exponent are approximate-value numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: **1**, **.2**, **3.4**, **-5**, **-6.78**, **+9.10**.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: **1.2E3**, **1.2E-3**, **-1.2E3**, **-1.2E-3**.

Two numbers that look similar may be treated differently. For example, **2.34** is an exact-value (fixed-point) number, whereas **2.34E0** is an approximate-value (floating-point) number.

The [**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 is a fixed-point type and calculations are exact. In MySQL, the [**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) type has several synonyms: [**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), [**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), [**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). The integer types also are exact-value types. For more information about exact-value calculations, 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).

The [**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) data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with [**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) are [**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) and [**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).

An integer may be used in floating-point context; it is interpreted as the equivalent floating-point number.

### 9.1.3 Date and Time Literals

Date and time values can be represented in several formats, such as quoted strings or as numbers, depending on the exact type of the value and other factors. For example, in contexts where MySQL expects a date, it interprets any of **'2015-07-21'**, **'20150721'**, and **20150721** as a date.

This section describes the acceptable formats for date and time literals. For more information about the temporal data types, such as the range of permitted values, 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).

**Standard SQL and ODBC Date and Time Literals.** Standard SQL requires temporal literals to be specified using a type keyword and a string. The space between the keyword and string is optional.

DATE '***str***'

TIME '***str***'

TIMESTAMP '***str***'

MySQL recognizes but, unlike standard SQL, does not require the type keyword. Applications that are to be standard-compliant should include the type keyword for temporal literals.

MySQL also recognizes the ODBC syntax corresponding to the standard SQL syntax:

{ d '***str***' }

{ t '***str***' }

{ ts '***str***' }

MySQL uses the type keywords and the ODBC constructions to produce [**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), 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, respectively, including a trailing fractional seconds part if specified. 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) syntax produces 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) value in MySQL because [**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 range that more closely corresponds to the standard SQL [**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) type, which has a year range from **0001** to **9999**. (The MySQL [**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) year range is **1970** to **2038**.)

**String and Numeric Literals in Date and Time Context.** MySQL recognizes [**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 these formats:

As a string in either **'*YYYY-MM-DD*'** or **'*YY-MM-DD*'** format. A “relaxed” syntax is permitted: Any punctuation character may be used as the delimiter between date parts. For example, **'2012-12-31'**, **'2012/12/31'**, **'2012^12^31'**, and **'2012@12@31'** are equivalent.

As a string with no delimiters in either **'*YYYYMMDD*'** or **'*YYMMDD*'** format, provided that the string makes sense as a date. For example, **'20070523'** and **'070523'** are interpreted as **'2007-05-23'**, but **'071332'** is illegal (it has nonsensical month and day parts) and becomes **'0000-00-00'**.

As a number in either ***YYYYMMDD*** or ***YYMMDD*** format, provided that the number makes sense as a date. For example, **19830905** and **830905** are interpreted as **'1983-09-05'**.

MySQL recognizes [**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 in these formats:

As a string in either **'*YYYY-MM-DD hh:mm:ss*'** or **'*YY-MM-DD hh:mm:ss*'** format. A “relaxed” syntax is permitted here, too: Any punctuation character may be used as the delimiter between date parts or time parts. For example, **'2012-12-31 11:30:45'**, **'2012^12^31 11+30+45'**, **'2012/12/31 11\*30\*45'**, and **'2012@12@31 11^30^45'** are equivalent.

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

The date and time parts can be separated by **T** rather than a space. For example, **'2012-12-31 11:30:45'** **'2012-12-31T11:30:45'** are equivalent.

As a string with no delimiters in either **'*YYYYMMDDhhmmss*'** or **'*YYMMDDhhmmss*'** format, provided that the string makes sense as a date. For example, **'20070523091528'** and **'070523091528'** are interpreted as **'2007-05-23 09:15:28'**, but **'071122129015'** is illegal (it has a nonsensical minute part) and becomes **'0000-00-00 00:00:00'**.

As a number in either ***YYYYMMDDhhmmss*** or ***YYMMDDhhmmss*** format, provided that the number makes sense as a date. For example, **19830905132800** and **830905132800** are interpreted as **'1983-09-05 13:28:00'**.

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 can include a trailing fractional seconds part in up to microseconds (6 digits) precision. 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).

Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-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).

For values specified as strings that include date part delimiters, it is unnecessary to specify two digits for month or day values that are less than **10**. **'2015-6-9'** is the same as **'2015-06-09'**. Similarly, for values specified as strings that include time part delimiters, it is unnecessary to specify two digits for hour, minute, or second values that are less than **10**. **'2015-10-30 1:2:3'** is the same as **'2015-10-30 01:02:03'**.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in ***YYYYMMDD*** or ***YYYYMMDDhhmmss*** format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in ***YYMMDD*** or ***YYMMDDhhmmss*** format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as nondelimited strings are interpreted according their length. For a string 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify **'9903'**, thinking that represents March, 1999, MySQL converts it to the “zero” date value. This occurs because the year and month values are **99** and **03**, but the day part is completely missing. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, to insert the value **'1999-03-00'**, use **'990300'**.

MySQL recognizes [**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 these formats:

As a string in ***'D hh:mm:ss'*** format. You can also use one of the following “relaxed” syntaxes: ***'hh:mm:ss'***, ***'hh:mm'***, ***'D hh:mm'***, ***'D hh'***, or ***'ss'***. Here ***D*** represents days and can have a value from 0 to 34.

As a string with no delimiters in ***'hhmmss'*** format, provided that it makes sense as a time. For example, **'101112'** is understood as **'10:11:12'**, but **'109712'** is illegal (it has a nonsensical minute part) and becomes **'00:00:00'**.

As a number in ***hhmmss*** format, provided that it makes sense as a time. For example, **101112** is understood as **'10:11:12'**. The following alternative formats are also understood: ***ss***, ***mmss***, or ***hhmmss***.

A trailing fractional seconds part is recognized in the ***'D hh:mm:ss.fraction'***, ***'hh:mm:ss.fraction'***, ***'hhmmss.fraction'***, and ***hhmmss.fraction*** time formats, where **fraction** is the fractional part in up to microseconds (6 digits) precision. 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).

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) values specified as strings that include a time part delimiter, it is unnecessary to specify two digits for hours, minutes, or seconds values that are less than **10**. **'8:3:2'** is the same as **'08:03:02'**.

### 9.1.4 Hexadecimal Literals

Hexadecimal literal values are written using **X'*val*'** or **0x*val*** notation, where ***val*** contains hexadecimal digits (**0..9**, **A..F**). Lettercase of the digits and of any leading **X** does not matter. A leading **0x** is case-sensitive and cannot be written as **0X**.

Legal hexadecimal literals:

X'01AF'

X'01af'

x'01AF'

x'01af'

0x01AF

0x01af

Illegal hexadecimal literals:

X'0G' (G is not a hexadecimal digit)

0X01AF (0X must be written as 0x)

Values written using **X'*val*'** notation must contain an even number of digits or a syntax error occurs. To correct the problem, pad the value with a leading zero:

mysql> **SET @s = X'FFF';**

ERROR 1064 (42000): You have an error in your SQL syntax;

check the manual that corresponds to your MySQL server

version for the right syntax to use near 'X'FFF''

mysql> **SET @s = X'0FFF';**

Query OK, 0 rows affected (0.00 sec)

Values written using **0x*val*** notation that contain an odd number of digits are treated as having an extra leading **0**. For example, **0xaaa** is interpreted as **0x0aaa**.

By default, a hexadecimal literal is a binary string, where each pair of hexadecimal digits represents a character:

mysql> **SELECT X'4D7953514C', CHARSET(X'4D7953514C');**

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

| X'4D7953514C' | CHARSET(X'4D7953514C') |

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

| MySQL | binary |

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

mysql> **SELECT 0x5461626c65, CHARSET(0x5461626c65);**

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

| 0x5461626c65 | CHARSET(0x5461626c65) |

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

| Table | binary |

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

A hexadecimal literal may have an optional character set introducer and **COLLATE** clause, to designate it as a string that uses a particular character set and collation:

[\_***charset\_name***] X'***val***' [COLLATE ***collation\_name***]

Examples:

SELECT \_latin1 X'4D7953514C';

SELECT \_utf8 0x4D7953514C COLLATE utf8\_danish\_ci;

The examples use **X'*val*'** notation, but **0x*val*** notation permits introducers as well. For information about introducers, see [Section 10.3.8, “Character Set Introducers”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-introducer).

In numeric contexts, MySQL treats a hexadecimal literal like 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) (64-bit integer). To ensure numeric treatment of a hexadecimal literal, use it in numeric context. Ways to do this include adding 0 or using [**CAST(... AS UNSIGNED)**](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). For example, a hexadecimal literal assigned to a user-defined variable is a binary string by default. To assign the value as a number, use it in numeric context:

mysql> **SET @v1 = X'41';**

mysql> **SET @v2 = X'41'+0;**

mysql> **SET @v3 = CAST(X'41' AS UNSIGNED);**

mysql> **SELECT @v1, @v2, @v3;**

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

| @v1 | @v2 | @v3 |

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

| A | 65 | 65 |

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

An empty hexadecimal value (**X''**) evaluates to a zero-length binary string. Converted to a number, it produces 0:

mysql> **SELECT CHARSET(X''), LENGTH(X'');**

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

| CHARSET(X'') | LENGTH(X'') |

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

| binary | 0 |

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

mysql> **SELECT X''+0;**

+-------+

| X''+0 |

+-------+

| 0 |

+-------+

The **X'*val*'** notation is based on standard SQL. The **0x** notation is based on ODBC, for which hexadecimal strings are often used to supply values 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) columns.

To convert a string or a number to a string in hexadecimal format, use the [**HEX()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_hex) function:

mysql> **SELECT HEX('cat');**

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

| HEX('cat') |

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

| 636174 |

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

mysql> **SELECT X'636174';**

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

| X'636174' |

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

| cat |

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

For hexadecimal literals, bit operations are considered numeric context, but bit operations permit numeric or binary string arguments in MySQL 8.0 and higher. To explicitly specify binary string context for hexadecimal literals, use a **\_binary** introducer for at least one of the arguments:

mysql> **SET @v1 = X'000D' | X'0BC0';**

mysql> **SET @v2 = \_binary X'000D' | X'0BC0';**

mysql> **SELECT HEX(@v1), HEX(@v2);**

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

| HEX(@v1) | HEX(@v2) |

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

| BCD | 0BCD |

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

The displayed result appears similar for both bit operations, but the result without **\_binary** is a **BIGINT** value, whereas the result with **\_binary** is a binary string. Due to the difference in result types, the displayed values differ: High-order 0 digits are not displayed for the numeric result.

### 9.1.5 Bit-Value Literals

Bit-value literals are written using **b'*val*'** or **0b*val*** notation. ***val*** is a binary value written using zeros and ones. Lettercase of any leading **b** does not matter. A leading **0b** is case-sensitive and cannot be written as **0B**.

Legal bit-value literals:

b'01'

B'01'

0b01

Illegal bit-value literals:

b'2' (2 is not a binary digit)

0B01 (0B must be written as 0b)

By default, a bit-value literal is a binary string:

mysql> **SELECT b'1000001', CHARSET(b'1000001');**

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

| b'1000001' | CHARSET(b'1000001') |

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

| A | binary |

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

mysql> **SELECT 0b1100001, CHARSET(0b1100001);**

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

| 0b1100001 | CHARSET(0b1100001) |

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

| a | binary |

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

A bit-value literal may have an optional character set introducer and **COLLATE** clause, to designate it as a string that uses a particular character set and collation:

[\_***charset\_name***] b'***val***' [COLLATE ***collation\_name***]

Examples:

SELECT \_latin1 b'1000001';

SELECT \_utf8 0b1000001 COLLATE utf8\_danish\_ci;

The examples use **b'*val*'** notation, but **0b*val*** notation permits introducers as well. For information about introducers, see [Section 10.3.8, “Character Set Introducers”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-introducer).

In numeric contexts, MySQL treats a bit literal like an integer. To ensure numeric treatment of a bit literal, use it in numeric context. Ways to do this include adding 0 or using [**CAST(... AS UNSIGNED)**](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). For example, a bit literal assigned to a user-defined variable is a binary string by default. To assign the value as a number, use it in numeric context:

mysql> **SET @v1 = b'1100001';**

mysql> **SET @v2 = b'1100001'+0;**

mysql> **SET @v3 = CAST(b'1100001' AS UNSIGNED);**

mysql> **SELECT @v1, @v2, @v3;**

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

| @v1 | @v2 | @v3 |

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

| a | 97 | 97 |

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

An empty bit value (**b''**) evaluates to a zero-length binary string. Converted to a number, it produces 0:

mysql> **SELECT CHARSET(b''), LENGTH(b'');**

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

| CHARSET(b'') | LENGTH(b'') |

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

| binary | 0 |

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

mysql> **SELECT b''+0;**

+-------+

| b''+0 |

+-------+

| 0 |

+-------+

Bit-value notation is convenient for specifying values to be assigned to [**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) columns:

mysql> **CREATE TABLE t (b BIT(8));**

mysql> **INSERT INTO t SET b = b'11111111';**

mysql> **INSERT INTO t SET b = b'1010';**

mysql> **INSERT INTO t SET b = b'0101';**

Bit values in result sets are returned as binary values, which may not display well. To convert a bit value to printable form, use it in numeric context or use a conversion function such as [**BIN()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_bin) or [**HEX()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_hex). High-order 0 digits are not displayed in the converted value.

mysql> **SELECT b+0, BIN(b), OCT(b), HEX(b) FROM t;**

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

| b+0 | BIN(b) | OCT(b) | HEX(b) |

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

| 255 | 11111111 | 377 | FF |

| 10 | 1010 | 12 | A |

| 5 | 101 | 5 | 5 |

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

For bit literals, bit operations are considered numeric context, but bit operations permit numeric or binary string arguments in MySQL 8.0 and higher. To explicitly specify binary string context for bit literals, use a **\_binary** introducer for at least one of the arguments:

mysql> **SET @v1 = b'000010101' | b'000101010';**

mysql> **SET @v2 = \_binary b'000010101' | \_binary b'000101010';**

mysql> **SELECT HEX(@v1), HEX(@v2);**

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

| HEX(@v1) | HEX(@v2) |

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

| 3F | 003F |

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

The displayed result appears similar for both bit operations, but the result without **\_binary** is a **BIGINT** value, whereas the result with **\_binary** is a binary string. Due to the difference in result types, the displayed values differ: High-order 0 digits are not displayed for the numeric result.

### 9.1.6 Boolean Literals

The constants **TRUE** and **FALSE** evaluate to **1** and **0**, respectively. The constant names can be written in any lettercase.

mysql> **SELECT TRUE, true, FALSE, false;**

-> 1, 1, 0, 0

### 9.1.7 NULL Values

The **NULL** value means “no data.” **NULL** can be written in any lettercase.

Be aware that the **NULL** value is different from values such as **0** for numeric types or the empty string for string types. For more information, see [Section B.3.4.3, “Problems with NULL 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-null).

For text file import or export operations performed 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) or [**SELECT ... INTO OUTFILE**](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-into), **NULL** is represented by the **\N** sequence. See [Section 13.2.7, “LOAD DATA 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#load-data).

For sorting with **ORDER BY**, **NULL** values sort before other values for ascending sorts, after other values for descending sorts.

## 9.2 Schema Object Names

[9.2.1 Identifier Length Limits](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifier-length)

[9.2.2 Identifier Qualifiers](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifier-qualifiers)

[9.2.3 Identifier Case Sensitivity](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifier-case-sensitivity)

[9.2.4 Mapping of Identifiers to File Names](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifier-mapping)

[9.2.5 Function Name Parsing and Resolution](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#function-resolution)

Certain objects within MySQL, including database, table, index, column, alias, view, stored procedure, partition, tablespace, resource group and other object names are known as identifiers. This section describes the permissible syntax for identifiers in MySQL. [Section 9.2.1, “Identifier Length Limits”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifier-length), indicates the maximum length of each type of identifier. [Section 9.2.3, “Identifier Case Sensitivity”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifier-case-sensitivity), describes which types of identifiers are case-sensitive and under what conditions.

An identifier may be quoted or unquoted. If an identifier contains special characters or is a reserved word, you must quote it whenever you refer to it. (Exception: A reserved word that follows a period in a qualified name must be an identifier, so it need not be quoted.) Reserved words are listed at [Section 9.3, “Keywords and Reserved Words”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords).

Internally, identifiers are converted to and are stored as Unicode (UTF-8). The permissible Unicode characters in identifiers are those in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted. Identifiers thus may contain these characters:

Permitted characters in unquoted identifiers:

ASCII: [0-9,a-z,A-Z$\_] (basic Latin letters, digits 0-9, dollar, underscore)

Extended: U+0080 .. U+FFFF

Permitted characters in quoted identifiers include the full Unicode Basic Multilingual Plane (BMP), except U+0000:

ASCII: U+0001 .. U+007F

Extended: U+0080 .. U+FFFF

ASCII NUL (U+0000) and supplementary characters (U+10000 and higher) are not permitted in quoted or unquoted identifiers.

Identifiers may begin with a digit but unless quoted may not consist solely of digits.

Database, table, and column names cannot end with space characters.

The identifier quote character is the backtick (**`**):

mysql> **SELECT \* FROM `select` WHERE `select`.id > 100;**

If the [**ANSI\_QUOTES**](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_ansi_quotes) SQL mode is enabled, it is also permissible to quote identifiers within double quotation marks:

mysql> **CREATE TABLE "test" (col INT);**

ERROR 1064: You have an error in your SQL syntax...

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

mysql> **CREATE TABLE "test" (col INT);**

Query OK, 0 rows affected (0.00 sec)

The [**ANSI\_QUOTES**](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_ansi_quotes) mode causes the server to interpret double-quoted strings as identifiers. Consequently, when this mode is enabled, string literals must be enclosed within single quotation marks. They cannot be enclosed within double quotation marks. The server SQL mode is controlled as described in [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).

Identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named **a`b** that contains a column named **c"d**:

mysql> **CREATE TABLE `a``b` (`c"d` INT);**

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

mysql> **SELECT 1 AS `one`, 2 AS 'two';**

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

| one | two |

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

| 1 | 2 |

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

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal.

It is recommended that you do not use names that begin with ***M*e** or ***M*e*N***, where ***M*** and ***N*** are integers. For example, avoid using **1e** as an identifier, because an expression such as **1e+3** is ambiguous. Depending on context, it might be interpreted as the expression **1e + 3** or as the number **1e+3**.

Be careful when using [**MD5()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_md5) to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See [Section 9.4, “User-Defined Variables”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#user-variables), for more information and examples of workarounds.

Special characters in database and table names are encoded in the corresponding file system names as described in [Section 9.2.4, “Mapping of Identifiers to File Names”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifier-mapping).

### 9.2.1 Identifier Length Limits

The following table describes the maximum length for each type of identifier.

| **Identifier Type** | **Maximum Length (characters)** |
| --- | --- |
| Database | 64 (includes NDB Cluster 8.0.18 and later) |
| Table | 64 (includes NDB Cluster 8.0.18 and later) |
| Column | 64 |
| Index | 64 |
| Constraint | 64 |
| Stored Program | 64 |
| View | 64 |
| Tablespace | 64 |
| Server | 64 |
| Log File Group | 64 |
| Alias | 256 (see exception following table) |
| Compound Statement Label | 16 |
| User-Defined Variable | 64 |
| Resource Group | 64 |

Aliases for column names in [**CREATE VIEW**](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-view) statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

For constraint definitions that include no constraint name, the server internally generates a name derived from the associated table name. For example, internally generated foreign key and **CHECK** constraint names consist of the table name plus **\_ibfk\_** or **\_chk\_** and a number. If the table name is close to the length limit for constraint names, the additional characters required for the constraint name may cause that name to exceed the limit, resulting in an error.

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions and to identifiers stored in the grant tables in the **mysql** database. The sizes of the identifier string columns in the grant tables are measured in characters. You can use multibyte characters without reducing the number of characters permitted for values stored in these columns.

Prior to NDB 8.0.18, NDB Cluster imposed a maximum length of 63 characters for names of databases and tables. As of NDB 8.0.18, this limitation is removed. See [Section 23.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”](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-limitations-resolved).

Values such as user name and host names in MySQL account names are strings rather than identifiers. For information about the maximum length of such values as stored in grant tables, see [Grant Table Scope Column Properties](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#grant-tables-scope-column-properties).

### 9.2.2 Identifier Qualifiers

Object names may be unqualified or qualified. An unqualified name is permitted in contexts where interpretation of the name is unambiguous. A qualified name includes at least one qualifier to clarify the interpretive context by overriding a default context or providing missing context.

For example, this statement creates a table using the unqualified name **t1**:

CREATE TABLE t1 (i INT);

Because **t1** includes no qualifier to specify a database, the statement creates the table in the default database. If there is no default database, an error occurs.

This statement creates a table using the qualified name **db1.t1**:

CREATE TABLE db1.t1 (i INT);

Because **db1.t1** includes a database qualifier **db1**, the statement creates **t1** in the database named **db1**, regardless of the default database. The qualifier must be specified if there is no default database. The qualifier may be specified if there is a default database, to specify a database different from the default, or to make the database explicit if the default is the same as the one specified.

Qualifiers have these characteristics:

An unqualified name consists of a single identifier. A qualified name consists of multiple identifiers.

The components of a multiple-part name must be separated by period (**.**) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which to interpret the final identifier.

The qualifier character is a separate token and need not be contiguous with the associated identifiers. For example, ***tbl\_name.col\_name*** and ***tbl\_name . col\_name*** are equivalent.

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write **`my-table`.`my-column`**, not **`my-table.my-column`**.

A reserved word that follows a period in a qualified name must be an identifier, so in that context it need not be quoted.

The permitted qualifiers for object names depend on the object type:

A database name is fully qualified and takes no qualifier:

CREATE DATABASE db1;

A table, view, or stored program name may be given a database-name qualifier. Examples of unqualified and qualified names in **CREATE** statements:

CREATE TABLE mytable ...;

CREATE VIEW myview ...;

CREATE PROCEDURE myproc ...;

CREATE FUNCTION myfunc ...;

CREATE EVENT myevent ...;

CREATE TABLE mydb.mytable ...;

CREATE VIEW mydb.myview ...;

CREATE PROCEDURE mydb.myproc ...;

CREATE FUNCTION mydb.myfunc ...;

CREATE EVENT mydb.myevent ...;

A trigger is associated with a table, so any qualifier applies to the table name:

CREATE TRIGGER mytrigger ... ON mytable ...;

CREATE TRIGGER mytrigger ... ON mydb.mytable ...;

A column name may be given multiple qualifiers to indicate context in statements that reference it, as shown in the following table.

| **Column Reference** | **Meaning** |
| --- | --- |
| ***col\_name*** | Column ***col\_name*** from whichever table used in the statement contains a column of that name |
| ***tbl\_name.col\_name*** | Column ***col\_name*** from table ***tbl\_name*** of the default database |
| ***db\_name.tbl\_name.col\_name*** | Column ***col\_name*** from table ***tbl\_name*** of the database ***db\_name*** |

In other words, a column name may be given a table-name qualifier, which itself may be given a database-name qualifier. Examples of unqualified and qualified column references in **SELECT** statements:

SELECT c1 FROM mytable

WHERE c2 > 100;

SELECT mytable.c1 FROM mytable

WHERE mytable.c2 > 100;

SELECT mydb.mytable.c1 FROM mydb.mytable

WHERE mydb.mytable.c2 > 100;

You need not specify a qualifier for an object reference in a statement unless the unqualified reference is ambiguous. Suppose that column **c1** occurs only in table **t1**, **c2** only in **t2**, and **c** in both **t1** and **t2**. Any unqualified reference to **c** is ambiguous in a statement that refers to both tables and must be qualified as **t1.c** or **t2.c** to indicate which table you mean:

SELECT c1, c2, t1.c FROM t1 INNER JOIN t2

WHERE t2.c > 100;

Similarly, to retrieve from a table **t** in database **db1** and from a table **t** in database **db2** in the same statement, you must qualify the table references: For references to columns in those tables, qualifiers are required only for column names that appear in both tables. Suppose that column **c1** occurs only in table **db1.t**, **c2** only in **db2.t**, and **c** in both **db1.t** and **db2.t**. In this case, **c** is ambiguous and must be qualified but **c1** and **c2** need not be:

SELECT c1, c2, db1.t.c FROM db1.t INNER JOIN db2.t

WHERE db2.t.c > 100;

Table aliases enable qualified column references to be written more simply:

SELECT c1, c2, t1.c FROM db1.t AS t1 INNER JOIN db2.t AS t2

WHERE t2.c > 100;

### 9.2.3 Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Triggers also correspond to files. Consequently, the case sensitivity of the underlying operating system plays a part in the case sensitivity of database, table, and trigger names. This means such names are not case-sensitive in Windows, but are case-sensitive in most varieties of Unix. One notable exception is macOS, which is Unix-based but uses a default file system type (HFS+) that is not case-sensitive. However, macOS also supports UFS volumes, which are case-sensitive just as on any Unix. See [Section 1.7.1, “MySQL Extensions to Standard SQL”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\introduction.html#extensions-to-ansi). The **[lower\_case\_table\_names](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_lower_case_table_names)** system variable also affects how the server handles identifier case sensitivity, as described later in this section.

**Note**

Although database, table, and trigger names are not case-sensitive on some platforms, you should not refer to one of these using different cases within the same statement. The following statement would not work because it refers to a table both as **my\_table** and as **MY\_TABLE**:

mysql> **SELECT \* FROM my\_table WHERE MY\_TABLE.col=1;**

Partition, subpartition, column, index, stored routine, event, and resource group names are not case-sensitive on any platform, nor are column aliases.

However, names of logfile groups are case-sensitive. This differs from standard SQL.

By default, table aliases are case-sensitive on Unix, but not so on Windows or macOS. The following statement would not work on Unix, because it refers to the alias both as **a** and as **A**:

mysql> **SELECT *col\_name* FROM *tbl\_name* AS a**

**WHERE a.*col\_name* = 1 OR A.*col\_name* = 2;**

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the **[lower\_case\_table\_names](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_lower_case_table_names)** system variable. **[lower\_case\_table\_names](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_lower_case_table_names)** can take the values shown in the following table. This variable does not affect case sensitivity of trigger identifiers. On Unix, the default value of **[lower\_case\_table\_names](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_lower_case_table_names)** is 0. On Windows, the default value is 1. On macOS, the default value is 2.

[**lower\_case\_table\_names**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sysvar_lower_case_table_names) can only be configured when initializing the server. Changing the **[lower\_case\_table\_names](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_lower_case_table_names)** setting after the server is initialized is prohibited.

| **Value** | **Meaning** |
| --- | --- |
| **0** | Table and database names are stored on disk using the lettercase specified in 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) or [**CREATE DATABASE**](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-database) statement. Name comparisons are case-sensitive. You should not set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or macOS). If you force this variable to 0 with [--lower-case-table-names=0](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sysvar_lower_case_table_names) on a case-insensitive file system and access **MyISAM** tablenames using different lettercases, index corruption may result. |
| **1** | Table names are stored in lowercase on disk and name comparisons are not case-sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases. |
| **2** | Table and database names are stored on disk using the lettercase specified in 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) or [**CREATE DATABASE**](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-database) statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case-sensitive. This works only on file systems that are not case-sensitive! **InnoDB** table names and view names are stored in lowercase, as for **lower\_case\_table\_names=1**. |

If you are using MySQL on only one platform, you do not normally have to use a **[lower\_case\_table\_names](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_lower_case_table_names)** setting other than the default. However, you may encounter difficulties if you want to transfer tables between platforms that differ in file system case sensitivity. For example, on Unix, you can have two different tables named **my\_table** and **MY\_TABLE**, but on Windows these two names are considered identical. To avoid data transfer problems arising from lettercase of database or table names, you have two options:

Use **lower\_case\_table\_names=1** on all systems. The main disadvantage with this is that when you use [**SHOW TABLES**](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-tables) or [**SHOW DATABASES**](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-databases), you do not see the names in their original lettercase.

Use **lower\_case\_table\_names=0** on Unix and **lower\_case\_table\_names=2** on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

***Exception***: If you are using **InnoDB** tables and you are trying to avoid these data transfer problems, you should use **[lower\_case\_table\_names=1](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_lower_case_table_names)** on all platforms to force names to be converted to lowercase.

Object names may be considered duplicates if their uppercase forms are equal according to a binary collation. That is true for names of cursors, conditions, procedures, functions, savepoints, stored routine parameters, stored program local variables, and plugins. It is not true for names of columns, constraints, databases, partitions, statements prepared with [**PREPARE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#prepare), tables, triggers, users, and user-defined variables.

File system case sensitivity can affect searches in string columns of **INFORMATION\_SCHEMA** tables. For more information, see [Section 10.8.7, “Using Collation in INFORMATION\_SCHEMA Searches”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\charset.html#charset-collation-information-schema).

### 9.2.4 Mapping of Identifiers to File Names

There is a correspondence between database and table identifiers and names in the file system. For the basic structure, MySQL represents each database as a directory in the data directory, and depending upon the storage engine, each table may be represented by one or more files in the appropriate database directory.

For the data and index files, the exact representation on disk is storage engine specific. These files may be stored in the database directory, or the information may be stored in a separate file. **InnoDB** data is stored in the InnoDB data files. If you are using tablespaces with **InnoDB**, then the specific tablespace files you create are used instead.

Any character is legal in database or table identifiers except ASCII NUL (**X'00'**). MySQL encodes any characters that are problematic in the corresponding file system objects when it creates database directories or table files:

Basic Latin letters (**a..zA..Z**), digits (**0..9**) and underscore (**\_**) are encoded as is. Consequently, their case sensitivity directly depends on file system features.

All other national letters from alphabets that have uppercase/lowercase mapping are encoded as shown in the following table. Values in the Code Range column are UCS-2 values.

| **Code Range** | **Pattern** | **Number** | **Used** | **Unused** | **Blocks** |
| --- | --- | --- | --- | --- | --- |
| **00C0..017F** | [@][0..4][g..z] | 5\*20= 100 | 97 | 3 | Latin-1 Supplement + Latin Extended-A |
| **0370..03FF** | [@][5..9][g..z] | 5\*20= 100 | 88 | 12 | Greek and Coptic |
| **0400..052F** | [@][g..z][0..6] | 20\*7= 140 | 137 | 3 | Cyrillic + Cyrillic Supplement |
| **0530..058F** | [@][g..z][7..8] | 20\*2= 40 | 38 | 2 | Armenian |
| **2160..217F** | [@][g..z][9] | 20\*1= 20 | 16 | 4 | Number Forms |
| **0180..02AF** | [@][g..z][a..k] | 20\*11=220 | 203 | 17 | Latin Extended-B + IPA Extensions |
| **1E00..1EFF** | [@][g..z][l..r] | 20\*7= 140 | 136 | 4 | Latin Extended Additional |
| **1F00..1FFF** | [@][g..z][s..z] | 20\*8= 160 | 144 | 16 | Greek Extended |
| **.... ....** | [@][a..f][g..z] | 6\*20= 120 | 0 | 120 | RESERVED |
| **24B6..24E9** | [@][@][a..z] | 26 | 26 | 0 | Enclosed Alphanumerics |
| **FF21..FF5A** | [@][a..z][@] | 26 | 26 | 0 | Halfwidth and Fullwidth forms |

One of the bytes in the sequence encodes lettercase. For example: **LATIN CAPITAL LETTER A WITH GRAVE** is encoded as **@0G**, whereas **LATIN SMALL LETTER A WITH GRAVE** is encoded as **@0g**. Here the third byte (**G** or **g**) indicates lettercase. (On a case-insensitive file system, both letters are treated as the same.)

For some blocks, such as Cyrillic, the second byte determines lettercase. For other blocks, such as Latin1 Supplement, the third byte determines lettercase. If two bytes in the sequence are letters (as in Greek Extended), the leftmost letter character stands for lettercase. All other letter bytes must be in lowercase.

All nonletter characters except underscore (**\_**), as well as letters from alphabets that do not have uppercase/lowercase mapping (such as Hebrew) are encoded using hexadecimal representation using lowercase letters for hexadecimal digits **a..f**:

0x003F -> @003f

0xFFFF -> @ffff

The hexadecimal values correspond to character values in the **ucs2** double-byte character set.

On Windows, some names such as **nul**, **prn**, and **aux** are encoded by appending **@@@** to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms.

### 9.2.5 Function Name Parsing and Resolution

MySQL supports built-in (native) functions, user-defined functions (UDFs), and stored functions. This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines which function to use in cases when functions of different types exist with a given name.

[Built-In Function Name Parsing](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#function-name-parsing)

[Function Name Resolution](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#function-name-resolution)

#### Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the [**IGNORE\_SPACE**](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_ignore_space) SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a nonexpression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to **count** is a function call, whereas the second reference is a table name:

SELECT COUNT(\*) FROM mytable;

CREATE TABLE count (i INT);

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in nonexpression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in nonexpression context:

To use the name as a function call in an expression, there must be no whitespace between the name and the following **(** parenthesis character.

Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. **COUNT** is one such name. The sql/lex.h source file lists the names of these special functions for which following whitespace determines their interpretation: names defined by the **SYM\_FN()** macro in the **symbols[]** array.

The following list names the functions in MySQL 8.0 that are affected by the [**IGNORE\_SPACE**](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_ignore_space) setting and listed as special in the sql/lex.h source file. You may find it easiest to treat the no-whitespace requirement as applying to all function calls.

**ADDDATE**

**BIT\_AND**

**BIT\_OR**

**BIT\_XOR**

**CAST**

**COUNT**

**CURDATE**

**CURTIME**

**DATE\_ADD**

**DATE\_SUB**

**EXTRACT**

**GROUP\_CONCAT**

**MAX**

**MID**

**MIN**

**NOW**

**POSITION**

**SESSION\_USER**

**STD**

**STDDEV**

**STDDEV\_POP**

**STDDEV\_SAMP**

**SUBDATE**

**SUBSTR**

**SUBSTRING**

**SUM**

**SYSDATE**

**SYSTEM\_USER**

**TRIM**

**VARIANCE**

**VAR\_POP**

**VAR\_SAMP**

For functions not listed as special in sql/lex.h, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. **ASCII** is one such name. However, for these nonaffected function names, interpretation may vary in expression context: ***func\_name* ()** is interpreted as a built-in function if there is one with the given name; if not, ***func\_name* ()** is interpreted as a user-defined function or stored function if one exists with that name.

The [**IGNORE\_SPACE**](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_ignore_space) SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

With [**IGNORE\_SPACE**](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_ignore_space) disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in nonexpression context:

mysql> **CREATE TABLE count(i INT);**

ERROR 1064 (42000): You have an error in your SQL syntax ...

near 'count(i INT)'

To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

CREATE TABLE count (i INT);

CREATE TABLE `count`(i INT);

CREATE TABLE `count` (i INT);

With [**IGNORE\_SPACE**](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_ignore_space) enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

SELECT COUNT(\*) FROM mytable;

SELECT COUNT (\*) FROM mytable;

However, enabling [**IGNORE\_SPACE**](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_ignore_space) also has the side effect that the parser treats the affected function names as reserved words (see [Section 9.3, “Keywords and Reserved Words”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords)). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in nonexpression context unless it is quoted. For example, with [**IGNORE\_SPACE**](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_ignore_space) enabled, both of the following statements fail with a syntax error because the parser interprets **count** as a reserved word:

CREATE TABLE count(i INT);

CREATE TABLE count (i INT);

To use the function name in nonexpression context, write it as a quoted identifier:

CREATE TABLE `count`(i INT);

CREATE TABLE `count` (i INT);

To enable the [**IGNORE\_SPACE**](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_ignore_space) SQL mode, use this statement:

SET sql\_mode = 'IGNORE\_SPACE';

[**IGNORE\_SPACE**](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_ignore_space) is also enabled by certain other composite modes such as [**ANSI**](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_ansi) that include it in their value:

SET sql\_mode = 'ANSI';

Check [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), to see which composite modes enable [**IGNORE\_SPACE**](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_ignore_space).

To minimize the dependency of SQL code on the [**IGNORE\_SPACE**](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_ignore_space) setting, use these guidelines:

Avoid creating UDFs or stored functions that have the same name as a built-in function.

Avoid using function names in nonexpression context. For example, these statements use **count** (one of the affected function names affected by [**IGNORE\_SPACE**](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_ignore_space)), so they fail with or without whitespace following the name if [**IGNORE\_SPACE**](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_ignore_space) is enabled:

CREATE TABLE count(i INT);

CREATE TABLE count (i INT);

If you must use a function name in nonexpression context, write it as a quoted identifier:

CREATE TABLE `count`(i INT);

CREATE TABLE `count` (i INT);

#### Function Name Resolution

The following rules describe how the server resolves references to function names for function creation and invocation:

Built-in functions and user-defined functions

An error occurs if you try to create a UDF with the same name as a built-in function.

Built-in functions and stored functions

It is possible to create a stored function with the same name as a built-in function, but to invoke the stored function it is necessary to qualify it with a schema name. For example, if you create a stored function named **PI** in the **test** schema, invoke it as **test.PI()** because the server resolves [**PI()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_pi) without a qualifier as a reference to the built-in function. The server generates a warning if the stored function name collides with a built-in function name. The warning can be displayed with [**SHOW WARNINGS**](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-warnings).

User-defined functions and stored functions

User-defined functions and stored functions share the same namespace, so you cannot create a UDF and a stored function with the same name.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions:

If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use [**DROP FUNCTION**](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-function) to drop the UDF and [**CREATE FUNCTION**](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-function) to re-create the UDF with a different nonconflicting name. Then modify any affected code to use the new name.

If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a nonconflicting name, or change calls to the function so that they use a schema qualifier (that is, use ***schema\_name*.*func\_name*()** syntax). In either case, modify any affected code accordingly.

## 9.3 Keywords and Reserved Words

Keywords are words that have significance in SQL. Certain keywords, such as [**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), [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete), or [**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), are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Nonreserved keywords are permitted as identifiers without quoting. Reserved words are permitted as identifiers if you quote them as described in [Section 9.2, “Schema Object Names”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifiers):

mysql> **CREATE TABLE interval (begin INT, end INT);**

ERROR 1064 (42000): You have an error in your SQL syntax ...

near 'interval (begin INT, end INT)'

**BEGIN** and **END** are keywords but not reserved, so their use as identifiers does not require quoting. **INTERVAL** is a reserved keyword and must be quoted to be used as an identifier:

mysql> **CREATE TABLE `interval` (begin INT, end INT);**

Query OK, 0 rows affected (0.01 sec)

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

mysql> **CREATE TABLE mydb.interval (begin INT, end INT);**

Query OK, 0 rows affected (0.01 sec)

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, **COUNT** is acceptable as a column name. However, by default, no whitespace is permitted in function invocations between the function name and the following **(** character. This requirement enables the parser to distinguish whether the name is used in a function call or in nonfunction context. For further details on recognition of function names, see [Section 9.2.5, “Function Name Parsing and Resolution”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#function-resolution).

The **INFORMATION\_SCHEMA.KEYWORDS** table lists the words considered keywords by MySQL and indicates whether they are reserved. See [Section 26.3.17, “The INFORMATION\_SCHEMA KEYWORDS 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-keywords-table).

[MySQL 8.0 Keywords and Reserved Words](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-in-current-series)

[MySQL 8.0 New Keywords and Reserved Words](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-in-current-series)

[MySQL 8.0 Removed Keywords and Reserved Words](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-removed-in-current-series)

### MySQL 8.0 Keywords and Reserved Words

The following list shows the keywords and reserved words in MySQL 8.0, along with changes to individual words from version to version. Reserved keywords are marked with (R). In addition, **\_FILENAME** is reserved.

At some point, you might upgrade to a higher version, so it is a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the reserved words in the list are forbidden by standard SQL as column or table names (for example, **GROUP**). A few are reserved because MySQL needs them and uses a **yacc** parser.

[A](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\language-structure.html" \l "keywords-8-0-detailed-A) | [B](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-B) | [C](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-C) | [D](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-D) | [E](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-E) | [F](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-F) | [G](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-G) | [H](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-H) | [I](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-I) | [J](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-J) | [K](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-K) | [L](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-L) | [M](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-M) | [N](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-N) | [O](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-O) | [P](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-P) | [Q](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-Q) | [R](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-R) | [S](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-S) | [T](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-T) | [U](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-U) | [V](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-V) | [W](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-W) | [X](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-X) | [Y](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-Y) | [Z](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-8-0-detailed-Z)

A

**ACCESSIBLE** (R)

**ACCOUNT**

**ACTION**

**ACTIVE**; added in 8.0.14 (nonreserved)

**ADD** (R)

**ADMIN**; became nonreserved in 8.0.12

**AFTER**

**AGAINST**

**AGGREGATE**

**ALGORITHM**

**ALL** (R)

**ALTER** (R)

**ALWAYS**

**ANALYSE**; removed in 8.0.1

**ANALYZE** (R)

**AND** (R)

**ANY**

**ARRAY**; added in 8.0.17 (reserved); became nonreserved in 8.0.19

**AS** (R)

**ASC** (R)

**ASCII**

**ASENSITIVE** (R)

**AT**

**ATTRIBUTE**; added in 8.0.21 (nonreserved)

**AUTOEXTEND\_SIZE**

**AUTO\_INCREMENT**

**AVG**

**AVG\_ROW\_LENGTH**

B

**BACKUP**

**BEFORE** (R)

**BEGIN**

**BETWEEN** (R)

**BIGINT** (R)

**BINARY** (R)

**BINLOG**

**BIT**

**BLOB** (R)

**BLOCK**

**BOOL**

**BOOLEAN**

**BOTH** (R)

**BTREE**

**BUCKETS**; added in 8.0.2 (nonreserved)

**BY** (R)

**BYTE**

C

**CACHE**

**CALL** (R)

**CASCADE** (R)

**CASCADED**

**CASE** (R)

**CATALOG\_NAME**

**CHAIN**

**CHANGE** (R)

**CHANGED**

**CHANNEL**

**CHAR** (R)

**CHARACTER** (R)

**CHARSET**

**CHECK** (R)

**CHECKSUM**

**CIPHER**

**CLASS\_ORIGIN**

**CLIENT**

**CLONE**; added in 8.0.3 (nonreserved)

**CLOSE**

**COALESCE**

**CODE**

**COLLATE** (R)

**COLLATION**

**COLUMN** (R)

**COLUMNS**

**COLUMN\_FORMAT**

**COLUMN\_NAME**

**COMMENT**

**COMMIT**

**COMMITTED**

**COMPACT**

**COMPLETION**

**COMPONENT**

**COMPRESSED**

**COMPRESSION**

**CONCURRENT**

**CONDITION** (R)

**CONNECTION**

**CONSISTENT**

**CONSTRAINT** (R)

**CONSTRAINT\_CATALOG**

**CONSTRAINT\_NAME**

**CONSTRAINT\_SCHEMA**

**CONTAINS**

**CONTEXT**

**CONTINUE** (R)

**CONVERT** (R)

**CPU**

**CREATE** (R)

**CROSS** (R)

**CUBE** (R); became reserved in 8.0.1

**CUME\_DIST** (R); added in 8.0.2 (reserved)

**CURRENT**

**CURRENT\_DATE** (R)

**CURRENT\_TIME** (R)

**CURRENT\_TIMESTAMP** (R)

**CURRENT\_USER** (R)

**CURSOR** (R)

**CURSOR\_NAME**

D

**DATA**

**DATABASE** (R)

**DATABASES** (R)

**DATAFILE**

**DATE**

**DATETIME**

**DAY**

**DAY\_HOUR** (R)

**DAY\_MICROSECOND** (R)

**DAY\_MINUTE** (R)

**DAY\_SECOND** (R)

**DEALLOCATE**

**DEC** (R)

**DECIMAL** (R)

**DECLARE** (R)

**DEFAULT** (R)

**DEFAULT\_AUTH**

**DEFINER**

**DEFINITION**; added in 8.0.4 (nonreserved)

**DELAYED** (R)

**DELAY\_KEY\_WRITE**

**DELETE** (R)

**DENSE\_RANK** (R); added in 8.0.2 (reserved)

**DESC** (R)

**DESCRIBE** (R)

**DESCRIPTION**; added in 8.0.4 (nonreserved)

**DES\_KEY\_FILE**; removed in 8.0.3

**DETERMINISTIC** (R)

**DIAGNOSTICS**

**DIRECTORY**

**DISABLE**

**DISCARD**

**DISK**

**DISTINCT** (R)

**DISTINCTROW** (R)

**DIV** (R)

**DO**

**DOUBLE** (R)

**DROP** (R)

**DUAL** (R)

**DUMPFILE**

**DUPLICATE**

**DYNAMIC**

E

**EACH** (R)

**ELSE** (R)

**ELSEIF** (R)

**EMPTY** (R); added in 8.0.4 (reserved)

**ENABLE**

**ENCLOSED** (R)

**ENCRYPTION**

**END**

**ENDS**

**ENFORCED**; added in 8.0.16 (nonreserved)

**ENGINE**

**ENGINES**

**ENGINE\_ATTRIBUTE**; added in 8.0.21 (nonreserved)

**ENUM**

**ERROR**

**ERRORS**

**ESCAPE**

**ESCAPED** (R)

**EVENT**

**EVENTS**

**EVERY**

**EXCEPT** (R)

**EXCHANGE**

**EXCLUDE**; added in 8.0.2 (nonreserved)

**EXECUTE**

**EXISTS** (R)

**EXIT** (R)

**EXPANSION**

**EXPIRE**

**EXPLAIN** (R)

**EXPORT**

**EXTENDED**

**EXTENT\_SIZE**

F

**FAILED\_LOGIN\_ATTEMPTS**; added in 8.0.19 (nonreserved)

**FALSE** (R)

**FAST**

**FAULTS**

**FETCH** (R)

**FIELDS**

**FILE**

**FILE\_BLOCK\_SIZE**

**FILTER**

**FIRST**

**FIRST\_VALUE** (R); added in 8.0.2 (reserved)

**FIXED**

**FLOAT** (R)

**FLOAT4** (R)

**FLOAT8** (R)

**FLUSH**

**FOLLOWING**; added in 8.0.2 (nonreserved)

**FOLLOWS**

**FOR** (R)

**FORCE** (R)

**FOREIGN** (R)

**FORMAT**

**FOUND**

**FROM** (R)

**FULL**

**FULLTEXT** (R)

**FUNCTION** (R); became reserved in 8.0.1

G

**GENERAL**

**GENERATED** (R)

**GEOMCOLLECTION**; added in 8.0.11 (nonreserved)

**GEOMETRY**

**GEOMETRYCOLLECTION**

**GET** (R)

**GET\_FORMAT**

**GET\_MASTER\_PUBLIC\_KEY**; added in 8.0.4 (reserved); became nonreserved in 8.0.11

**GET\_SOURCE\_PUBLIC\_KEY**; added in 8.0.23 (nonreserved)

**GLOBAL**

**GRANT** (R)

**GRANTS**

**GROUP** (R)

**GROUPING** (R); added in 8.0.1 (reserved)

**GROUPS** (R); added in 8.0.2 (reserved)

**GROUP\_REPLICATION**

H

**HANDLER**

**HASH**

**HAVING** (R)

**HELP**

**HIGH\_PRIORITY** (R)

**HISTOGRAM**; added in 8.0.2 (nonreserved)

**HISTORY**; added in 8.0.3 (nonreserved)

**HOST**

**HOSTS**

**HOUR**

**HOUR\_MICROSECOND** (R)

**HOUR\_MINUTE** (R)

**HOUR\_SECOND** (R)

I

**IDENTIFIED**

**IF** (R)

**IGNORE** (R)

**IGNORE\_SERVER\_IDS**

**IMPORT**

**IN** (R)

**INACTIVE**; added in 8.0.14 (nonreserved)

**INDEX** (R)

**INDEXES**

**INFILE** (R)

**INITIAL\_SIZE**

**INNER** (R)

**INOUT** (R)

**INSENSITIVE** (R)

**INSERT** (R)

**INSERT\_METHOD**

**INSTALL**

**INSTANCE**

**INT** (R)

**INT1** (R)

**INT2** (R)

**INT3** (R)

**INT4** (R)

**INT8** (R)

**INTEGER** (R)

**INTERVAL** (R)

**INTO** (R)

**INVISIBLE**

**INVOKER**

**IO**

**IO\_AFTER\_GTIDS** (R)

**IO\_BEFORE\_GTIDS** (R)

**IO\_THREAD**

**IPC**

**IS** (R)

**ISOLATION**

**ISSUER**

**ITERATE** (R)

J

**JOIN** (R)

**JSON**

**JSON\_TABLE** (R); added in 8.0.4 (reserved)

**JSON\_VALUE**; added in 8.0.21 (nonreserved)

K

**KEY** (R)

**KEYRING**; added in 8.0.24 (nonreserved)

**KEYS** (R)

**KEY\_BLOCK\_SIZE**

**KILL** (R)

L

**LAG** (R); added in 8.0.2 (reserved)

**LANGUAGE**

**LAST**

**LAST\_VALUE** (R); added in 8.0.2 (reserved)

**LATERAL** (R); added in 8.0.14 (reserved)

**LEAD** (R); added in 8.0.2 (reserved)

**LEADING** (R)

**LEAVE** (R)

**LEAVES**

**LEFT** (R)

**LESS**

**LEVEL**

**LIKE** (R)

**LIMIT** (R)

**LINEAR** (R)

**LINES** (R)

**LINESTRING**

**LIST**

**LOAD** (R)

**LOCAL**

**LOCALTIME** (R)

**LOCALTIMESTAMP** (R)

**LOCK** (R)

**LOCKED**; added in 8.0.1 (nonreserved)

**LOCKS**

**LOGFILE**

**LOGS**

**LONG** (R)

**LONGBLOB** (R)

**LONGTEXT** (R)

**LOOP** (R)

**LOW\_PRIORITY** (R)

M

**MASTER**

**MASTER\_AUTO\_POSITION**

**MASTER\_BIND** (R)

**MASTER\_COMPRESSION\_ALGORITHMS**; added in 8.0.18 (nonreserved)

**MASTER\_CONNECT\_RETRY**

**MASTER\_DELAY**

**MASTER\_HEARTBEAT\_PERIOD**

**MASTER\_HOST**

**MASTER\_LOG\_FILE**

**MASTER\_LOG\_POS**

**MASTER\_PASSWORD**

**MASTER\_PORT**

**MASTER\_PUBLIC\_KEY\_PATH**; added in 8.0.4 (nonreserved)

**MASTER\_RETRY\_COUNT**

**MASTER\_SERVER\_ID**; removed in 8.0.23

**MASTER\_SSL**

**MASTER\_SSL\_CA**

**MASTER\_SSL\_CAPATH**

**MASTER\_SSL\_CERT**

**MASTER\_SSL\_CIPHER**

**MASTER\_SSL\_CRL**

**MASTER\_SSL\_CRLPATH**

**MASTER\_SSL\_KEY**

**MASTER\_SSL\_VERIFY\_SERVER\_CERT** (R)

**MASTER\_TLS\_CIPHERSUITES**; added in 8.0.19 (nonreserved)

**MASTER\_TLS\_VERSION**

**MASTER\_USER**

**MASTER\_ZSTD\_COMPRESSION\_LEVEL**; added in 8.0.18 (nonreserved)

**MATCH** (R)

**MAXVALUE** (R)

**MAX\_CONNECTIONS\_PER\_HOUR**

**MAX\_QUERIES\_PER\_HOUR**

**MAX\_ROWS**

**MAX\_SIZE**

**MAX\_UPDATES\_PER\_HOUR**

**MAX\_USER\_CONNECTIONS**

**MEDIUM**

**MEDIUMBLOB** (R)

**MEDIUMINT** (R)

**MEDIUMTEXT** (R)

**MEMBER**; added in 8.0.17 (reserved); became nonreserved in 8.0.19

**MEMORY**

**MERGE**

**MESSAGE\_TEXT**

**MICROSECOND**

**MIDDLEINT** (R)

**MIGRATE**

**MINUTE**

**MINUTE\_MICROSECOND** (R)

**MINUTE\_SECOND** (R)

**MIN\_ROWS**

**MOD** (R)

**MODE**

**MODIFIES** (R)

**MODIFY**

**MONTH**

**MULTILINESTRING**

**MULTIPOINT**

**MULTIPOLYGON**

**MUTEX**

**MYSQL\_ERRNO**

N

**NAME**

**NAMES**

**NATIONAL**

**NATURAL** (R)

**NCHAR**

**NDB**

**NDBCLUSTER**

**NESTED**; added in 8.0.4 (nonreserved)

**NETWORK\_NAMESPACE**; added in 8.0.16 (nonreserved)

**NEVER**

**NEW**

**NEXT**

**NO**

**NODEGROUP**

**NONE**

**NOT** (R)

**NOWAIT**; added in 8.0.1 (nonreserved)

**NO\_WAIT**

**NO\_WRITE\_TO\_BINLOG** (R)

**NTH\_VALUE** (R); added in 8.0.2 (reserved)

**NTILE** (R); added in 8.0.2 (reserved)

**NULL** (R)

**NULLS**; added in 8.0.2 (nonreserved)

**NUMBER**

**NUMERIC** (R)

**NVARCHAR**

O

**OF** (R); added in 8.0.1 (reserved)

**OFF**; added in 8.0.20 (nonreserved)

**OFFSET**

**OJ**; added in 8.0.16 (nonreserved)

**OLD**; added in 8.0.14 (nonreserved)

**ON** (R)

**ONE**

**ONLY**

**OPEN**

**OPTIMIZE** (R)

**OPTIMIZER\_COSTS** (R)

**OPTION** (R)

**OPTIONAL**; added in 8.0.13 (nonreserved)

**OPTIONALLY** (R)

**OPTIONS**

**OR** (R)

**ORDER** (R)

**ORDINALITY**; added in 8.0.4 (nonreserved)

**ORGANIZATION**; added in 8.0.4 (nonreserved)

**OTHERS**; added in 8.0.2 (nonreserved)

**OUT** (R)

**OUTER** (R)

**OUTFILE** (R)

**OVER** (R); added in 8.0.2 (reserved)

**OWNER**

P

**PACK\_KEYS**

**PAGE**

**PARSER**

**PARTIAL**

**PARTITION** (R)

**PARTITIONING**

**PARTITIONS**

**PASSWORD**

**PASSWORD\_LOCK\_TIME**; added in 8.0.19 (nonreserved)

**PATH**; added in 8.0.4 (nonreserved)

**PERCENT\_RANK** (R); added in 8.0.2 (reserved)

**PERSIST**; became nonreserved in 8.0.16

**PERSIST\_ONLY**; added in 8.0.2 (reserved); became nonreserved in 8.0.16

**PHASE**

**PLUGIN**

**PLUGINS**

**PLUGIN\_DIR**

**POINT**

**POLYGON**

**PORT**

**PRECEDES**

**PRECEDING**; added in 8.0.2 (nonreserved)

**PRECISION** (R)

**PREPARE**

**PRESERVE**

**PREV**

**PRIMARY** (R)

**PRIVILEGES**

**PRIVILEGE\_CHECKS\_USER**; added in 8.0.18 (nonreserved)

**PROCEDURE** (R)

**PROCESS**; added in 8.0.11 (nonreserved)

**PROCESSLIST**

**PROFILE**

**PROFILES**

**PROXY**

**PURGE** (R)

Q

**QUARTER**

**QUERY**

**QUICK**

R

**RANDOM**; added in 8.0.18 (nonreserved)

**RANGE** (R)

**RANK** (R); added in 8.0.2 (reserved)

**READ** (R)

**READS** (R)

**READ\_ONLY**

**READ\_WRITE** (R)

**REAL** (R)

**REBUILD**

**RECOVER**

**RECURSIVE** (R); added in 8.0.1 (reserved)

**REDOFILE**; removed in 8.0.3

**REDO\_BUFFER\_SIZE**

**REDUNDANT**

**REFERENCE**; added in 8.0.4 (nonreserved)

**REFERENCES** (R)

**REGEXP** (R)

**RELAY**

**RELAYLOG**

**RELAY\_LOG\_FILE**

**RELAY\_LOG\_POS**

**RELAY\_THREAD**

**RELEASE** (R)

**RELOAD**

**REMOTE**; added in 8.0.3 (nonreserved); removed in 8.0.14

**REMOVE**

**RENAME** (R)

**REORGANIZE**

**REPAIR**

**REPEAT** (R)

**REPEATABLE**

**REPLACE** (R)

**REPLICA**; added in 8.0.22 (nonreserved)

**REPLICAS**; added in 8.0.22 (nonreserved)

**REPLICATE\_DO\_DB**

**REPLICATE\_DO\_TABLE**

**REPLICATE\_IGNORE\_DB**

**REPLICATE\_IGNORE\_TABLE**

**REPLICATE\_REWRITE\_DB**

**REPLICATE\_WILD\_DO\_TABLE**

**REPLICATE\_WILD\_IGNORE\_TABLE**

**REPLICATION**

**REQUIRE** (R)

**REQUIRE\_ROW\_FORMAT**; added in 8.0.19 (nonreserved)

**RESET**

**RESIGNAL** (R)

**RESOURCE**; added in 8.0.3 (nonreserved)

**RESPECT**; added in 8.0.2 (nonreserved)

**RESTART**; added in 8.0.4 (nonreserved)

**RESTORE**

**RESTRICT** (R)

**RESUME**

**RETAIN**; added in 8.0.14 (nonreserved)

**RETURN** (R)

**RETURNED\_SQLSTATE**

**RETURNING**; added in 8.0.21 (nonreserved)

**RETURNS**

**REUSE**; added in 8.0.3 (nonreserved)

**REVERSE**

**REVOKE** (R)

**RIGHT** (R)

**RLIKE** (R)

**ROLE**; became nonreserved in 8.0.1

**ROLLBACK**

**ROLLUP**

**ROTATE**

**ROUTINE**

**ROW** (R); became reserved in 8.0.2

**ROWS** (R); became reserved in 8.0.2

**ROW\_COUNT**

**ROW\_FORMAT**

**ROW\_NUMBER** (R); added in 8.0.2 (reserved)

**RTREE**

S

**SAVEPOINT**

**SCHEDULE**

**SCHEMA** (R)

**SCHEMAS** (R)

**SCHEMA\_NAME**

**SECOND**

**SECONDARY**; added in 8.0.16 (nonreserved)

**SECONDARY\_ENGINE**; added in 8.0.13 (nonreserved)

**SECONDARY\_ENGINE\_ATTRIBUTE**; added in 8.0.21 (nonreserved)

**SECONDARY\_LOAD**; added in 8.0.13 (nonreserved)

**SECONDARY\_UNLOAD**; added in 8.0.13 (nonreserved)

**SECOND\_MICROSECOND** (R)

**SECURITY**

**SELECT** (R)

**SENSITIVE** (R)

**SEPARATOR** (R)

**SERIAL**

**SERIALIZABLE**

**SERVER**

**SESSION**

**SET** (R)

**SHARE**

**SHOW** (R)

**SHUTDOWN**

**SIGNAL** (R)

**SIGNED**

**SIMPLE**

**SKIP**; added in 8.0.1 (nonreserved)

**SLAVE**

**SLOW**

**SMALLINT** (R)

**SNAPSHOT**

**SOCKET**

**SOME**

**SONAME**

**SOUNDS**

**SOURCE**

**SOURCE\_AUTO\_POSITION**; added in 8.0.23 (nonreserved)

**SOURCE\_BIND**; added in 8.0.23 (nonreserved)

**SOURCE\_COMPRESSION\_ALGORITHMS**; added in 8.0.23 (nonreserved)

**SOURCE\_CONNECT\_RETRY**; added in 8.0.23 (nonreserved)

**SOURCE\_DELAY**; added in 8.0.23 (nonreserved)

**SOURCE\_HEARTBEAT\_PERIOD**; added in 8.0.23 (nonreserved)

**SOURCE\_HOST**; added in 8.0.23 (nonreserved)

**SOURCE\_LOG\_FILE**; added in 8.0.23 (nonreserved)

**SOURCE\_LOG\_POS**; added in 8.0.23 (nonreserved)

**SOURCE\_PASSWORD**; added in 8.0.23 (nonreserved)

**SOURCE\_PORT**; added in 8.0.23 (nonreserved)

**SOURCE\_PUBLIC\_KEY\_PATH**; added in 8.0.23 (nonreserved)

**SOURCE\_RETRY\_COUNT**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_CA**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_CAPATH**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_CERT**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_CIPHER**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_CRL**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_CRLPATH**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_KEY**; added in 8.0.23 (nonreserved)

**SOURCE\_SSL\_VERIFY\_SERVER\_CERT**; added in 8.0.23 (nonreserved)

**SOURCE\_TLS\_CIPHERSUITES**; added in 8.0.23 (nonreserved)

**SOURCE\_TLS\_VERSION**; added in 8.0.23 (nonreserved)

**SOURCE\_USER**; added in 8.0.23 (nonreserved)

**SOURCE\_ZSTD\_COMPRESSION\_LEVEL**; added in 8.0.23 (nonreserved)

**SPATIAL** (R)

**SPECIFIC** (R)

**SQL** (R)

**SQLEXCEPTION** (R)

**SQLSTATE** (R)

**SQLWARNING** (R)

**SQL\_AFTER\_GTIDS**

**SQL\_AFTER\_MTS\_GAPS**

**SQL\_BEFORE\_GTIDS**

**SQL\_BIG\_RESULT** (R)

**SQL\_BUFFER\_RESULT**

**SQL\_CACHE**; removed in 8.0.3

**SQL\_CALC\_FOUND\_ROWS** (R)

**SQL\_NO\_CACHE**

**SQL\_SMALL\_RESULT** (R)

**SQL\_THREAD**

**SQL\_TSI\_DAY**

**SQL\_TSI\_HOUR**

**SQL\_TSI\_MINUTE**

**SQL\_TSI\_MONTH**

**SQL\_TSI\_QUARTER**

**SQL\_TSI\_SECOND**

**SQL\_TSI\_WEEK**

**SQL\_TSI\_YEAR**

**SRID**; added in 8.0.3 (nonreserved)

**SSL** (R)

**STACKED**

**START**

**STARTING** (R)

**STARTS**

**STATS\_AUTO\_RECALC**

**STATS\_PERSISTENT**

**STATS\_SAMPLE\_PAGES**

**STATUS**

**STOP**

**STORAGE**

**STORED** (R)

**STRAIGHT\_JOIN** (R)

**STREAM**; added in 8.0.20 (nonreserved)

**STRING**

**SUBCLASS\_ORIGIN**

**SUBJECT**

**SUBPARTITION**

**SUBPARTITIONS**

**SUPER**

**SUSPEND**

**SWAPS**

**SWITCHES**

**SYSTEM** (R); added in 8.0.3 (reserved)

T

**TABLE** (R)

**TABLES**

**TABLESPACE**

**TABLE\_CHECKSUM**

**TABLE\_NAME**

**TEMPORARY**

**TEMPTABLE**

**TERMINATED** (R)

**TEXT**

**THAN**

**THEN** (R)

**THREAD\_PRIORITY**; added in 8.0.3 (nonreserved)

**TIES**; added in 8.0.2 (nonreserved)

**TIME**

**TIMESTAMP**

**TIMESTAMPADD**

**TIMESTAMPDIFF**

**TINYBLOB** (R)

**TINYINT** (R)

**TINYTEXT** (R)

**TLS**; added in 8.0.21 (nonreserved)

**TO** (R)

**TRAILING** (R)

**TRANSACTION**

**TRIGGER** (R)

**TRIGGERS**

**TRUE** (R)

**TRUNCATE**

**TYPE**

**TYPES**

U

**UNBOUNDED**; added in 8.0.2 (nonreserved)

**UNCOMMITTED**

**UNDEFINED**

**UNDO** (R)

**UNDOFILE**

**UNDO\_BUFFER\_SIZE**

**UNICODE**

**UNINSTALL**

**UNION** (R)

**UNIQUE** (R)

**UNKNOWN**

**UNLOCK** (R)

**UNSIGNED** (R)

**UNTIL**

**UPDATE** (R)

**UPGRADE**

**USAGE** (R)

**USE** (R)

**USER**

**USER\_RESOURCES**

**USE\_FRM**

**USING** (R)

**UTC\_DATE** (R)

**UTC\_TIME** (R)

**UTC\_TIMESTAMP** (R)

V

**VALIDATION**

**VALUE**

**VALUES** (R)

**VARBINARY** (R)

**VARCHAR** (R)

**VARCHARACTER** (R)

**VARIABLES**

**VARYING** (R)

**VCPU**; added in 8.0.3 (nonreserved)

**VIEW**

**VIRTUAL** (R)

**VISIBLE**

W

**WAIT**

**WARNINGS**

**WEEK**

**WEIGHT\_STRING**

**WHEN** (R)

**WHERE** (R)

**WHILE** (R)

**WINDOW** (R); added in 8.0.2 (reserved)

**WITH** (R)

**WITHOUT**

**WORK**

**WRAPPER**

**WRITE** (R)

X

**X509**

**XA**

**XID**

**XML**

**XOR** (R)

Y

**YEAR**

**YEAR\_MONTH** (R)

Z

**ZEROFILL** (R)

**ZONE**; added in 8.0.22 (nonreserved)

### MySQL 8.0 New Keywords and Reserved Words

The following list shows the keywords and reserved words that are added in MySQL 8.0, compared to MySQL 5.7. Reserved keywords are marked with (R).

[A](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\language-structure.html" \l "keywords-new-8-0-A) | [B](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-B) | [C](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-C) | [D](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-D) | [E](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-E) | [F](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-F) | [G](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-G) | [H](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-H) | [I](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-I) | [J](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-J) | [K](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-K) | [L](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-L) | [M](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-M) | [N](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-N) | [O](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-O) | [P](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-P) | [R](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-R) | [S](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-S) | [T](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-T) | [U](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-U) | [V](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-V) | [W](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-W) | [Z](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#keywords-new-8-0-Z)

A

**ACTIVE**

**ADMIN**

**ARRAY**

**ATTRIBUTE**

B

**BUCKETS**

C

**CLONE**

**COMPONENT**

**CUME\_DIST** (R)

D

**DEFINITION**

**DENSE\_RANK** (R)

**DESCRIPTION**

E

**EMPTY** (R)

**ENFORCED**

**ENGINE\_ATTRIBUTE**

**EXCEPT** (R)

**EXCLUDE**

F

**FAILED\_LOGIN\_ATTEMPTS**

**FIRST\_VALUE** (R)

**FOLLOWING**

G

**GEOMCOLLECTION**

**GET\_MASTER\_PUBLIC\_KEY**

**GET\_SOURCE\_PUBLIC\_KEY**

**GROUPING** (R)

**GROUPS** (R)

H

**HISTOGRAM**

**HISTORY**

I

**INACTIVE**

**INVISIBLE**

J

**JSON\_TABLE** (R)

**JSON\_VALUE**

K

**KEYRING**

L

**LAG** (R)

**LAST\_VALUE** (R)

**LATERAL** (R)

**LEAD** (R)

**LOCKED**

M

**MASTER\_COMPRESSION\_ALGORITHMS**

**MASTER\_PUBLIC\_KEY\_PATH**

**MASTER\_TLS\_CIPHERSUITES**

**MASTER\_ZSTD\_COMPRESSION\_LEVEL**

**MEMBER**

N

**NESTED**

**NETWORK\_NAMESPACE**

**NOWAIT**

**NTH\_VALUE** (R)

**NTILE** (R)

**NULLS**

O

**OF** (R)

**OFF**

**OJ**

**OLD**

**OPTIONAL**

**ORDINALITY**

**ORGANIZATION**

**OTHERS**

**OVER** (R)

P

**PASSWORD\_LOCK\_TIME**

**PATH**

**PERCENT\_RANK** (R)

**PERSIST**

**PERSIST\_ONLY**

**PRECEDING**

**PRIVILEGE\_CHECKS\_USER**

**PROCESS**

R

**RANDOM**

**RANK** (R)

**RECURSIVE** (R)

**REFERENCE**

**REPLICA**

**REPLICAS**

**REQUIRE\_ROW\_FORMAT**

**RESOURCE**

**RESPECT**

**RESTART**

**RETAIN**

**RETURNING**

**REUSE**

**ROLE**

**ROW\_NUMBER** (R)

S

**SECONDARY**

**SECONDARY\_ENGINE**

**SECONDARY\_ENGINE\_ATTRIBUTE**

**SECONDARY\_LOAD**

**SECONDARY\_UNLOAD**

**SKIP**

**SOURCE\_AUTO\_POSITION**

**SOURCE\_BIND**

**SOURCE\_COMPRESSION\_ALGORITHMS**

**SOURCE\_CONNECT\_RETRY**

**SOURCE\_DELAY**

**SOURCE\_HEARTBEAT\_PERIOD**

**SOURCE\_HOST**

**SOURCE\_LOG\_FILE**

**SOURCE\_LOG\_POS**

**SOURCE\_PASSWORD**

**SOURCE\_PORT**

**SOURCE\_PUBLIC\_KEY\_PATH**

**SOURCE\_RETRY\_COUNT**

**SOURCE\_SSL**

**SOURCE\_SSL\_CA**

**SOURCE\_SSL\_CAPATH**

**SOURCE\_SSL\_CERT**

**SOURCE\_SSL\_CIPHER**

**SOURCE\_SSL\_CRL**

**SOURCE\_SSL\_CRLPATH**

**SOURCE\_SSL\_KEY**

**SOURCE\_SSL\_VERIFY\_SERVER\_CERT**

**SOURCE\_TLS\_CIPHERSUITES**

**SOURCE\_TLS\_VERSION**

**SOURCE\_USER**

**SOURCE\_ZSTD\_COMPRESSION\_LEVEL**

**SRID**

**STREAM**

**SYSTEM** (R)

T

**THREAD\_PRIORITY**

**TIES**

**TLS**

U

**UNBOUNDED**

V

**VCPU**

**VISIBLE**

W

**WINDOW** (R)

Z

**ZONE**

### MySQL 8.0 Removed Keywords and Reserved Words

The following list shows the keywords and reserved words that are removed in MySQL 8.0, compared to MySQL 5.7. Reserved keywords are marked with (R).

**ANALYSE**

**DES\_KEY\_FILE**

**MASTER\_SERVER\_ID**

**PARSE\_GCOL\_EXPR**

**REDOFILE**

**SQL\_CACHE**

## 9.4 User-Defined Variables

You can store a value in a user-defined variable in one statement and refer to it later in another statement. This enables you to pass values from one statement to another.

User variables are written as **@*var\_name***, where the variable name ***var\_name*** consists of alphanumeric characters, **.**, **\_**, and **$**. A user variable name can contain other characters if you quote it as a string or identifier (for example, **@'my-var'**, **@"my-var"**, or **@`my-var`**).

User-defined variables are session specific. A user variable defined by one client cannot be seen or used by other clients. (Exception: A user with access to the Performance Schema **[user\_variables\_by\_thread](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\performance-schema.html" \l "performance-schema-user-variable-tables" \o "27.12.10 Performance Schema User-Defined Variable Tables)** table can see all user variables for all sessions.) All variables for a given client session are automatically freed when that client exits.

User variable names are not case-sensitive. Names have a maximum length of 64 characters.

One way to set a user-defined variable is by issuing a [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable) statement:

SET @***var\_name*** = ***expr*** [, @***var\_name*** = ***expr***] ...

For [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable), either [**=**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_assign-equal) or [**:=**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_assign-value) can be used as the assignment operator.

User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or nonbinary string, or **NULL** value. Assignment of decimal and real values does not preserve the precision or scale of the value. A value of a type other than one of the permissible types is converted to a permissible type. For example, a value having a temporal or spatial data type is converted to a binary string. A value having the [**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 type is converted to a string with a character set of **utf8mb4** and a collation of **utf8mb4\_bin**.

If a user variable is assigned a nonbinary (character) string value, it has the same character set and collation as the string. The coercibility of user variables is implicit. (This is the same coercibility as for table column values.)

Hexadecimal or bit values assigned to user variables are treated as binary strings. To assign a hexadecimal or bit value as a number to a user variable, use it in numeric context. For example, add 0 or use [**CAST(... AS UNSIGNED)**](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):

mysql> **SET @v1 = X'41';**

mysql> **SET @v2 = X'41'+0;**

mysql> **SET @v3 = CAST(X'41' AS UNSIGNED);**

mysql> **SELECT @v1, @v2, @v3;**

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

| @v1 | @v2 | @v3 |

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

| A | 65 | 65 |

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

mysql> **SET @v1 = b'1000001';**

mysql> **SET @v2 = b'1000001'+0;**

mysql> **SET @v3 = CAST(b'1000001' AS UNSIGNED);**

mysql> **SELECT @v1, @v2, @v3;**

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

| @v1 | @v2 | @v3 |

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

| A | 65 | 65 |

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

If the value of a user variable is selected in a result set, it is returned to the client as a string.

If you refer to a variable that has not been initialized, it has a value of **NULL** and a type of string.

Beginning with MySQL 8.0.22, a reference to a user variable in a prepared statement has its type determined when the statement is first prepared, and retains this type each time the statement is executed thereafter. Similarly, the type of a user variable employed in a statement within a stored procedure is determined the first time the stored procedure is invoked, and retains this type with each subsequent invocation.

User variables may be used in most contexts where expressions are permitted. This does not currently include contexts that explicitly require a literal value, such as in the **LIMIT** clause of a [**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, or the **IGNORE *N* LINES** clause of a [**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) statement.

Previous releases of MySQL made it possible to assign a value to a user variable in statements other than [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable). This functionality is supported in MySQL 8.0 for backward compatibility but is subject to removal in a future release of MySQL.

When making an assignment in this way, you must use [**:=**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_assign-value) as the assignment operator; **=** is treated as the comparison operator in statements other than [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable).

The order of evaluation for expressions involving user variables is undefined. For example, there is no guarantee that **SELECT @a, @a:=@a+1** evaluates **@a** first and then performs the assignment.

In addition, the default result type of a variable is based on its type at the beginning of the statement. This may have unintended effects if a variable holds a value of one type at the beginning of a statement in which it is also assigned a new value of a different type.

To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to **0**, **0.0**, or **''** to define its type before you use it.

**HAVING**, **GROUP BY**, and **ORDER BY**, when referring to a variable that is assigned a value in the select expression list do not work as expected because the expression is evaluated on the client and thus can use stale column values from a previous row.

User variables are intended to provide data values. They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected, or as a reserved word such as [**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). This is true even if the variable is quoted, as shown in the following example:

mysql> **SELECT c1 FROM t;**

+----+

| c1 |

+----+

| 0 |

+----+

| 1 |

+----+

2 rows in set (0.00 sec)

mysql> **SET @col = "c1";**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT @col FROM t;**

+------+

| @col |

+------+

| c1 |

+------+

1 row in set (0.00 sec)

mysql> **SELECT `@col` FROM t;**

ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT @col FROM t;**

+------+

| @col |

+------+

| `c1` |

+------+

1 row in set (0.00 sec)

An exception to this principle that user variables cannot be used to provide identifiers, is when you are constructing a string for use as a prepared statement to execute later. In this case, user variables can be used to provide any part of the statement. The following example illustrates how this can be done:

mysql> **SET @c = "c1";**

Query OK, 0 rows affected (0.00 sec)

mysql> **SET @s = CONCAT("SELECT ", @c, " FROM t");**

Query OK, 0 rows affected (0.00 sec)

mysql> **PREPARE stmt FROM @s;**

Query OK, 0 rows affected (0.04 sec)

Statement prepared

mysql> **EXECUTE stmt;**

+----+

| c1 |

+----+

| 0 |

+----+

| 1 |

+----+

2 rows in set (0.00 sec)

mysql> **DEALLOCATE PREPARE stmt;**

Query OK, 0 rows affected (0.00 sec)

See [Section 13.5, “Prepared Statements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#sql-prepared-statements), for more information.

A similar technique can be used in application programs to construct SQL statements using program variables, as shown here using PHP 5:

<?php

$mysqli = new mysqli("localhost", "user", "pass", "test");

if( mysqli\_connect\_errno() )

die("Connection failed: %s\n", mysqli\_connect\_error());

$col = "c1";

$query = "SELECT $col FROM t";

$result = $mysqli->query($query);

while($row = $result->fetch\_assoc())

{

echo "<p>" . $row["$col"] . "</p>\n";

}

$result->close();

$mysqli->close();

?>

Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”.

## 9.5 Expressions

This section lists the grammar rules that expressions must follow in MySQL and provides additional information about the types of terms that may appear in expressions.

[Expression Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#expression-syntax)

[Expression Term Notes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#expression-term-notes)

[Temporal Intervals](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#temporal-intervals)

### Expression Syntax

The following grammar rules define expression syntax in MySQL. The grammar shown here is based on that given in the sql/sql\_yacc.yy file of MySQL source distributions. For additional information about some of the expression terms, see [Expression Term Notes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#expression-term-notes).

***expr***:

***expr*** OR ***expr***

| ***expr*** || ***expr***

| ***expr*** XOR ***expr***

| ***expr*** AND ***expr***

| ***expr*** && ***expr***

| NOT ***expr***

| ! ***expr***

| ***boolean\_primary*** IS [NOT] {TRUE | FALSE | UNKNOWN}

| ***boolean\_primary***

***boolean\_primary***:

***boolean\_primary*** IS [NOT] NULL

| ***boolean\_primary*** <=> ***predicate***

| ***boolean\_primary*** ***comparison\_operator*** ***predicate***

| ***boolean\_primary*** ***comparison\_operator*** {ALL | ANY} (***subquery***)

| ***predicate***

***comparison\_operator***: = | >= | > | <= | < | <> | !=

***predicate***:

***bit\_expr*** [NOT] IN (***subquery***)

| ***bit\_expr*** [NOT] IN (***expr*** [, ***expr***] ...)

| ***bit\_expr*** [NOT] BETWEEN ***bit\_expr*** AND ***predicate***

| ***bit\_expr*** SOUNDS LIKE ***bit\_expr***

| ***bit\_expr*** [NOT] LIKE ***simple\_expr*** [ESCAPE ***simple\_expr***]

| ***bit\_expr*** [NOT] REGEXP ***bit\_expr***

| ***bit\_expr***

***bit\_expr***:

***bit\_expr*** | ***bit\_expr***

| ***bit\_expr*** & ***bit\_expr***

| ***bit\_expr*** << ***bit\_expr***

| ***bit\_expr*** >> ***bit\_expr***

| ***bit\_expr*** + ***bit\_expr***

| ***bit\_expr*** - ***bit\_expr***

| ***bit\_expr*** \* ***bit\_expr***

| ***bit\_expr*** / ***bit\_expr***

| ***bit\_expr*** DIV ***bit\_expr***

| ***bit\_expr*** MOD ***bit\_expr***

| ***bit\_expr*** % ***bit\_expr***

| ***bit\_expr*** ^ ***bit\_expr***

| ***bit\_expr*** + ***interval\_expr***

| ***bit\_expr*** - ***interval\_expr***

| ***simple\_expr***

***simple\_expr***:

***literal***

| ***identifier***

| ***function\_call***

| ***simple\_expr*** COLLATE ***collation\_name***

| ***param\_marker***

| ***variable***

| ***simple\_expr*** || ***simple\_expr***

| + ***simple\_expr***

| - ***simple\_expr***

| ~ ***simple\_expr***

| ! ***simple\_expr***

| BINARY ***simple\_expr***

| (***expr*** [, ***expr***] ...)

| ROW (***expr***, ***expr*** [, ***expr***] ...)

| (***subquery***)

| EXISTS (***subquery***)

| {***identifier*** ***expr***}

| ***match\_expr***

| ***case\_expr***

| ***interval\_expr***

For operator precedence, see [Section 12.4.1, “Operator Precedence”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator-precedence). The precedence and meaning of some operators depends on the SQL mode:

By default, [**||**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_or) is a logical [**OR**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_or) operator. With [**PIPES\_AS\_CONCAT**](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_pipes_as_concat) enabled, [**||**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_or) is string concatenation, with a precedence 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_bitwise-xor) and the unary operators.

By default, [**!**](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) has a higher precedence than **NOT**. With [**HIGH\_NOT\_PRECEDENCE**](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_high_not_precedence) enabled, [**!**](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) and **NOT** have the same precedence.

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).

### Expression Term Notes

For literal value syntax, see [Section 9.1, “Literal Values”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#literals).

For identifier syntax, see [Section 9.2, “Schema Object Names”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#identifiers).

Variables can be user variables, system variables, or stored program local variables or parameters:

User variables: [Section 9.4, “User-Defined Variables”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#user-variables)

System variables: [Section 5.1.9, “Using 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#using-system-variables)

Stored program local variables: [Section 13.6.4.1, “Local Variable DECLARE 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#declare-local-variable)

Stored program parameters: [Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Statements”](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-procedure)

***param\_marker*** is **?** as used in prepared statements for placeholders. See [Section 13.5.1, “PREPARE 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#prepare).

**(*subquery*)** indicates a subquery that returns a single value; that is, a scalar subquery. See [Section 13.2.11.1, “The Subquery as Scalar Operand”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#scalar-subqueries).

**{*identifier* *expr*}** is ODBC escape syntax and is accepted for ODBC compatibility. The value is ***expr***. The **{** and **}** curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

***match\_expr*** indicates a [**MATCH**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_match) expression. See [Section 12.10, “Full-Text Search Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#fulltext-search).

***case\_expr*** indicates a [**CASE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_case) expression. See [Section 12.5, “Flow Control Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#flow-control-functions).

***interval\_expr*** represents a temporal interval. See [Temporal Intervals](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#temporal-intervals).

### Temporal Intervals

***interval\_expr*** in expressions represents a temporal interval. Intervals have this syntax:

INTERVAL ***expr*** ***unit***

***expr*** represents a quantity. ***unit*** represents the unit for interpreting the quantity; it is a specifier such as **HOUR**, **DAY**, or **WEEK**. The **INTERVAL** keyword and the ***unit*** specifier are not case-sensitive.

The following table shows the expected form of the ***expr*** argument for each ***unit*** value.

**Table 9.2 Temporal Interval Expression and Unit Arguments**

| ***unit* Value** | **Expected *expr* Format** |
| --- | --- |
| **MICROSECOND** | **MICROSECONDS** |
| **SECOND** | **SECONDS** |
| **MINUTE** | **MINUTES** |
| **HOUR** | **HOURS** |
| **DAY** | **DAYS** |
| **WEEK** | **WEEKS** |
| **MONTH** | **MONTHS** |
| **QUARTER** | **QUARTERS** |
| **YEAR** | **YEARS** |
| **SECOND\_MICROSECOND** | **'SECONDS.MICROSECONDS'** |
| **MINUTE\_MICROSECOND** | **'MINUTES:SECONDS.MICROSECONDS'** |
| **MINUTE\_SECOND** | **'MINUTES:SECONDS'** |
| **HOUR\_MICROSECOND** | **'HOURS:MINUTES:SECONDS.MICROSECONDS'** |
| **HOUR\_SECOND** | **'HOURS:MINUTES:SECONDS'** |
| **HOUR\_MINUTE** | **'HOURS:MINUTES'** |
| **DAY\_MICROSECOND** | **'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'** |
| **DAY\_SECOND** | **'DAYS HOURS:MINUTES:SECONDS'** |
| **DAY\_MINUTE** | **'DAYS HOURS:MINUTES'** |
| **DAY\_HOUR** | **'DAYS HOURS'** |
| **YEAR\_MONTH** | **'YEARS-MONTHS'** |

MySQL permits any punctuation delimiter in the ***expr*** format. Those shown in the table are the suggested delimiters.

Temporal intervals are used for certain functions, such as [**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) and [**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):

mysql> **SELECT DATE\_ADD('2018-05-01',INTERVAL 1 DAY);**

-> '2018-05-02'

mysql> **SELECT DATE\_SUB('2018-05-01',INTERVAL 1 YEAR);**

-> '2017-05-01'

mysql> **SELECT DATE\_ADD('2020-12-31 23:59:59',**

-> **INTERVAL 1 SECOND);**

-> '2021-01-01 00:00:00'

mysql> **SELECT DATE\_ADD('2018-12-31 23:59:59',**

-> **INTERVAL 1 DAY);**

-> '2019-01-01 23:59:59'

mysql> **SELECT DATE\_ADD('2100-12-31 23:59:59',**

-> **INTERVAL '1:1' MINUTE\_SECOND);**

-> '2101-01-01 00:01:00'

mysql> **SELECT DATE\_SUB('2025-01-01 00:00:00',**

-> **INTERVAL '1 1:1:1' DAY\_SECOND);**

-> '2024-12-30 22:58:59'

mysql> **SELECT DATE\_ADD('1900-01-01 00:00:00',**

-> **INTERVAL '-1 10' DAY\_HOUR);**

-> '1899-12-30 14:00:00'

mysql> **SELECT DATE\_SUB('1998-01-02', INTERVAL 31 DAY);**

-> '1997-12-02'

mysql> **SELECT DATE\_ADD('1992-12-31 23:59:59.000002',**

-> **INTERVAL '1.999999' SECOND\_MICROSECOND);**

-> '1993-01-01 00:00:01.000001'

Temporal arithmetic also can be performed in expressions using **INTERVAL** together with 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_plus) or [**-**](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) operator:

**date** + INTERVAL ***expr*** ***unit***

**date** - INTERVAL ***expr*** ***unit***

**INTERVAL *expr* *unit*** is permitted on either side of 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_plus) operator if the expression on the other side is a date or datetime value. For 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) operator, **INTERVAL *expr* *unit*** is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

mysql> **SELECT '2018-12-31 23:59:59' + INTERVAL 1 SECOND;**

-> '2019-01-01 00:00:00'

mysql> **SELECT INTERVAL 1 DAY + '2018-12-31';**

-> '2019-01-01'

mysql> **SELECT '2025-01-01' - INTERVAL 1 SECOND;**

-> '2024-12-31 23:59:59'

The [**EXTRACT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_extract) function uses the same kinds of ***unit*** specifiers as [**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) or [**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), but extracts parts from the date rather than performing date arithmetic:

mysql> **SELECT EXTRACT(YEAR FROM '2019-07-02');**

-> 2019

mysql> **SELECT EXTRACT(YEAR\_MONTH FROM '2019-07-02 01:02:03');**

-> 201907

Temporal intervals can be used in [**CREATE EVENT**](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-event) statements:

CREATE EVENT myevent

ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 1 HOUR

DO

UPDATE myschema.mytable SET mycol = mycol + 1;

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the ***unit*** keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a ***unit*** of **DAY\_SECOND**, the value of ***expr*** is expected to have days, hours, minutes, and seconds parts. If you specify a value like **'1:10'**, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, **'1:10' DAY\_SECOND** is interpreted in such a way that it is equivalent to **'1:10' MINUTE\_SECOND**. This is analogous to the way that MySQL interprets [**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 as representing elapsed time rather than as a time of day.

***expr*** is treated as a string, so be careful if you specify a nonstring value with **INTERVAL**. For example, with an interval specifier of **HOUR\_MINUTE**, '6/4' is treated as 6 hours, four minutes, whereas **6/4** evaluates to **1.5000** and is treated as 1 hour, 5000 minutes:

mysql> **SELECT '6/4', 6/4;**

-> 1.5000

mysql> **SELECT DATE\_ADD('2019-01-01', INTERVAL '6/4' HOUR\_MINUTE);**

-> '2019-01-01 06:04:00'

mysql> **SELECT DATE\_ADD('2019-01-01', INTERVAL 6/4 HOUR\_MINUTE);**

-> '2019-01-04 12:20:00'

To ensure interpretation of the interval value as you expect, a [**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) operation may be used. To treat **6/4** as 1 hour, 5 minutes, cast it to a [**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) value with a single fractional digit:

mysql> **SELECT CAST(6/4 AS DECIMAL(3,1));**

-> 1.5

mysql> **SELECT DATE\_ADD('1970-01-01 12:00:00',**

-> **INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR\_MINUTE);**

-> '1970-01-01 13:05:00'

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

mysql> **SELECT DATE\_ADD('2023-01-01', INTERVAL 1 DAY);**

-> '2023-01-02'

mysql> **SELECT DATE\_ADD('2023-01-01', INTERVAL 1 HOUR);**

-> '2023-01-01 01:00:00'

If you add **MONTH**, **YEAR\_MONTH**, or **YEAR** and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

mysql> **SELECT DATE\_ADD('2019-01-30', INTERVAL 1 MONTH);**

-> '2019-02-28'

Date arithmetic operations require complete dates and do not work with incomplete dates such as **'2016-07-00'** or badly malformed dates:

mysql> **SELECT DATE\_ADD('2016-07-00', INTERVAL 1 DAY);**

-> NULL

mysql> **SELECT '2005-03-32' + INTERVAL 1 MONTH;**

-> NULL

## 9.6 Query Attributes

The most visible part of an SQL statement is the text of the statement. As of MySQL 8.0.23, clients can also define query attributes that apply to the next statement sent to the server for execution:

Attributes are defined prior to sending the statement.

Attributes exist until statement execution ends, at which point the attribute set is cleared.

While attributes exist, they can be accessed on the server side.

Examples of the ways query attributes may be used:

A web application produces pages that generate database queries, and for each query must track the URL of the page that generated it.

An application passes extra processing information with each query, for use by a plugin such as an audit plugin or query rewrite plugin.

MySQL supports these capabilities without the use of workarounds such as specially formatted comments included in query strings. The remainder of this section describes how to use query attribute support, including the prerequisites that must be satisfied.

[Defining and Accessing Query Attributes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#using-query-attributes)

[Prerequisites for Using Query Attributes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#query-attributes-prerequisites)

### Defining and Accessing Query Attributes

Applications that use the MySQL C API define query attributes by calling the **[mysql\_bind\_param()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-bind-param.html" \t "_top)** function. See [mysql\_bind\_param()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-bind-param.html" \t "_top). Other MySQL connectors may also provide query-attribute support. See the documentation for individual connectors.

The **[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)** client has a **query\_attributes** command that enables defining up to 32 pairs of attribute names and values. See [Section 4.5.1.2, “mysql Client Commands”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\programs.html#mysql-commands).

Query attribute names are transmitted using the character set indicated by the **[character\_set\_client](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_character_set_client)** system variable.

To access query attributes within SQL statements for which attributes have been defined, install the **query\_attributes** component as described in [Prerequisites for Using Query Attributes](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\language-structure.html#query-attributes-prerequisites). The component implements a **[mysql\_query\_attribute\_string()](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 "udf_mysql-query-attribute-string)** user-defined function (UDF) that takes an attribute name argument and returns the attribute value as a string, or **NULL** if the attribute does not exist. See [Section 5.7.3, “Query Attribute User-Defined Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#query-attribute-udfs).

The following examples use the **[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)** client **query\_attributes** command to define attribute name/value pairs, and the **[mysql\_query\_attribute\_string()](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 "udf_mysql-query-attribute-string)** UDF to access attribute values by name.

This example defines two attributes named **n1** and **n2**. The first **SELECT** shows how to retrieve those attributes, and also demonstrates that retrieving a nonexistent attribute (**n3**) returns **NULL**. The second **SELECT** shows that attributes do not persist across statements.

mysql> **query\_attributes n1 v1 n2 v2;**

mysql> **SELECT**

**mysql\_query\_attribute\_string('n1') AS 'attr 1',**

**mysql\_query\_attribute\_string('n2') AS 'attr 2',**

**mysql\_query\_attribute\_string('n3') AS 'attr 3';**

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

| attr 1 | attr 2 | attr 3 |

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

| v1 | v2 | NULL |

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

mysql> **SELECT**

**mysql\_query\_attribute\_string('n1') AS 'attr 1',**

**mysql\_query\_attribute\_string('n2') AS 'attr 2';**

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

| attr 1 | attr 2 |

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

| NULL | NULL |

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

As shown by the second **SELECT** statement, attributes defined prior to a given statement are available only to that statement and are cleared after the statement executes. To use an attribute value across multiple statements, assign it to a variable. The following example shows how to do this, and illustrates that attribute values are available in subsequent statements by means of the variables, but not by calling **[mysql\_query\_attribute\_string()](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 "udf_mysql-query-attribute-string)**:

mysql> **query\_attributes n1 v1 n2 v2;**

mysql> **SET**

**@attr1 = mysql\_query\_attribute\_string('n1'),**

**@attr2 = mysql\_query\_attribute\_string('n2');**

mysql> **SELECT**

**@attr1, mysql\_query\_attribute\_string('n1') AS 'attr 1',**

**@attr2, mysql\_query\_attribute\_string('n2') AS 'attr 2';**

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

| @attr1 | attr 1 | @attr2 | attr 2 |

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

| v1 | NULL | v2 | NULL |

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

Attributes can also be saved for later use by storing them in a table:

mysql> **CREATE TABLE t1 (c1 CHAR(20), c2 CHAR(20));**

mysql> **query\_attributes n1 v1 n2 v2;**

mysql> **INSERT INTO t1 (c1, c2) VALUES(**

**mysql\_query\_attribute\_string('n1'),**

**mysql\_query\_attribute\_string('n2')**

**);**

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

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

| c1 | c2 |

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

| v1 | v2 |

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

Query attributes are subject to these limitations and restrictions:

If multiple attribute-definition operations occur prior to sending a statement to the server for execution, the most recent definition operation applies and replaces attributes defined in earlier operations.

If multiple attributes are defined with the same name, attempts to retrieve the attribute value have an undefined result.

An attribute defined with an empty name cannot be retrieved by name.

Attributes are not available to statements prepared with [**PREPARE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#prepare).

The **[mysql\_query\_attribute\_string()](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 "udf_mysql-query-attribute-string)** UDF cannot be used in DDL statements.

Attributes are not replicated. Statements that invoke the **[mysql\_query\_attribute\_string()](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 "udf_mysql-query-attribute-string)** UDF will not get the same value on all servers.

### Prerequisites for Using Query Attributes

To access query attributes within SQL statements for which attributes have been defined, the **query\_attributes** component must be installed. Do so using this statement:

INSTALL COMPONENT "file://component\_query\_attributes";

Component installation is a one-time operation that need not be done per server startup. [**INSTALL COMPONENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#install-component) loads the component, and also registers it in the **mysql.component** system table to cause it to be loaded during subsequent server startups.

The **query\_attributes** component accesses query attributes to implement a **[mysql\_query\_attribute\_string()](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 "udf_mysql-query-attribute-string)** UDF. See [Section 5.5.4, “Query Attribute Components”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#query-attribute-components).

To uninstall the **query\_attributes** component, use this statement:

UNINSTALL COMPONENT "file://component\_query\_attributes";

[**UNINSTALL COMPONENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#uninstall-component) unloads the component, and unregisters it from the **mysql.component** system table to cause it not to be loaded during subsequent server startups.

Because installing and uninstalling the **query\_attributes** component installs and uninstalls the **[mysql\_query\_attribute\_string()](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 "udf_mysql-query-attribute-string)** function that the component implements, it is not necessary to use **CREATE FUNCTION** or **DROP FUNCTION** to do so.

## 9.7 Comments

MySQL Server supports three comment styles:

From a **#** character to the end of the line.

From a **--** sequence to the end of the line. In MySQL, the **--** (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in [Section 1.7.2.4, “'--' as the Start of a Comment”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\introduction.html#ansi-diff-comments).

From a **/\*** sequence to the following **\*/** sequence, as in the C programming language. This syntax enables a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

mysql> **SELECT 1+1; # This comment continues to the end of line**

mysql> **SELECT 1+1; -- This comment continues to the end of line**

mysql> **SELECT 1 /\* this is an in-line comment \*/ + 1;**

mysql> **SELECT 1+**

**/\***

**this is a**

**multiple-line comment**

**\*/**

**1;**

Nested comments are not supported, and are deprecated; expect them to be removed in a future MySQL release. (Under some conditions, nested comments might be permitted, but usually are not, and users should avoid them.)

MySQL Server supports certain variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

/\*! ***MySQL-specific code*** \*/

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers should ignore the extensions. For example, MySQL Server recognizes the **STRAIGHT\_JOIN** keyword in the following statement, but other servers should not:

SELECT /\*! STRAIGHT\_JOIN \*/ col1 FROM table1,table2 WHERE ...

If you add a version number after the **!** character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The **KEY\_BLOCK\_SIZE** keyword in the following comment is executed only by servers from MySQL 5.1.10 or higher:

CREATE TABLE t1(a INT, KEY (a)) /\*!50110 KEY\_BLOCK\_SIZE=1024 \*/;

The comment syntax just described applies to how the **[mysqld](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysqld" \o "4.3.1 mysqld — The MySQL Server)** server parses SQL statements. The **[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)** client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.) For information about differences between the server and **[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)** client parsers, see [Section 4.5.1.6, “mysql Client Tips”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\programs.html#mysql-tips).

Comments in **/\*!12345 ... \*/** format are not stored on the server. If this format is used to comment stored programs, the comments are not retained in the program body.

Another variant of C-style comment syntax is used to specify optimizer hints. Hint comments include a **+** character following the **/\*** comment opening sequence. Example:

SELECT /\*+ BKA(t1) \*/ FROM ... ;

For more information, see [Section 8.9.3, “Optimizer Hints”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#optimizer-hints).

The use of short-form **[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)** commands such as **\C** within multiple-line **/\* ... \*/** comments is not supported. Short-form commands do work within single-line **/\*! ... \*/** version comments, as do **/\*+ ... \*/** optimizer-hint comments, which are stored in object definitions. If there is a concern that optimizer-hint comments may be stored in object definitions so that dump files when reloaded with **mysql** would result in execution of such commands, either invoke **[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)** with the [--binary-mode](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\programs.html#option_mysql_binary-mode) option or use a reload client other than **[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)**.