NULL clause, and it is terminated with END CASE instead of END.

 for use inside stored programs. The CASE statement cannot have an ELSE

The return type of a CASE expression result is the aggregated type of all result values:

• If all types are numeric, the aggregated type is also numeric:

• If at least one argument is double precision, the result is double precision.

• Otherwise, if at least one argument is DECIMAL, the result is DECIMAL.

• Otherwise, the result is an integer type (with one exception):

• If all integer types are all signed or all unsigned, the result is the same sign and the precision is the highest of all specified integer types (that is, TINYINT, SMALLINT, MEDIUMINT, INT, or BIGINT).

• If there is a combination of signed and unsigned integer types, the result is signed and the precision may be higher. For example, if the types are signed INT and unsigned INT, the result is signed BIGINT.

• The exception is unsigned BIGINT combined with any signed integer type. The result is DECIMAL with sufficient precision and scale 0.

• If all types are BIT, the result is BIT. Otherwise, BIT arguments are treated similar to BIGINT.

• If all types are YEAR, the result is YEAR. Otherwise, YEAR arguments are treated similar to INT.

• If all types are character string (CHAR or VARCHAR), the result is VARCHAR with maximum length determined by the longest character length of the operands.

• If all types are character or binary string, the result is VARBINARY.

• SET and ENUM are treated similar to VARCHAR; the result is VARCHAR.

• If all types are JSON, the result is JSON.

• If all types are temporal, the result is temporal:

• If all temporal types are DATE, TIME, or TIMESTAMP, the result is DATE, TIME, or TIMESTAMP, respectively.

• Otherwise, for a mix of temporal types, the result is DATETIME.

• If all types are GEOMETRY, the result is GEOMETRY.

• If any type is BLOB, the result is BLOB.

• For all other type combinations, the result is VARCHAR.

• Literal NULL operands are ignored for type aggregation.

mysql> **SELECT** **CASE** **1** **WHEN** **1** **THEN** **'one'**

-> **WHEN** **2** **THEN** **'two'** **ELSE** **'more'** **END;**

-> 'one'

mysql> **SELECT** **CASE** **WHEN** **1>0** **THEN** **'true'** **ELSE** **'false'** **END;**

-> 'true'

mysql> **SELECT** **CASE** **BINARY** **'B'**

-> **WHEN** **'a'** **THEN** **1** **WHEN** **'b'** **THEN** **2** **END;**

-> NULL

• [IF(*expr1*,*expr2*,*expr3*)](#_bookmark1)

If *expr1* is TRUE (*expr1* <> 0 and *expr1* IS NOT NULL), [IF()](#_bookmark1) returns *expr2*. Otherwise, it returns *expr3*.

**Note**

There is also an IF *statement*, which differs from the [IF()](#_bookmark1) *function* described here. See Section 13.6.5.2, “IF Statement” .

If only one of *expr2* or *expr3* is explicitly NULL, the result type of the [IF()](#_bookmark1) function is the type of the non-NULL expression.

The default return type of [IF()](#_bookmark1) (which may matter when it is stored into a temporary table) is calculated as follows:

• If *expr2* or *expr3* produce a string, the result is a string.

If *expr2* and *expr3* are both strings, the result is case-sensitive if either string is case-sensitive.

• If *expr2* or *expr3* produce a floating-point value, the result is a floating-point value.

• If *expr2* or *expr3* produce an integer, the result is an integer.

mysql> **SELECT** **IF(1>2,2,3);**

-> 3

mysql> **SELECT** **IF(1<2,'yes','no');**

-> 'yes'

mysql> **SELECT** **IF(STRCMP('test','test1'),'no','yes');**

-> 'no'

• [IFNULL(*expr1*,*expr2*)](#_bookmark2)

If *expr1* is not NULL, [IFNULL()](#_bookmark2) returns *expr1*; otherwise it returns *expr2*.

mysql> **SELECT** **IFNULL(1,0);**

-> 1

mysql> **SELECT** **IFNULL(NULL,10);**

-> 10

mysql> **SELECT** **IFNULL(1/0,10);**

-> 10

mysql> **SELECT** **IFNULL(1/0,'yes');**

-> 'yes'

The default return type of [IFNULL(*expr1*,*expr2*)](#_bookmark2) is the more “general” of the two expressions, in the order STRING, REAL, or INTEGER. Consider the case of a table based on expressions or where MySQL must internally store a value returned by [IFNULL()](#_bookmark2) in a temporary table:

mysql> **CREATE** **TABLE** **tmp** **SELECT** **IFNULL(1,'test')** **AS** **test;**

mysql> **DESCRIBE** **tmp;**

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

| Field | Type | Null | Key | Default | Extra |

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

| test | varbinary(4) | NO | | | |

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

In this example, the type of the test column is VARBINARY(4) (a string type).



• [NULLIF(*expr1*,*expr2*)](#_bookmark3)

Returns NULL if *expr1* = *expr2* is true, otherwise returns *expr1*. This is the same as CASE WHEN *expr1* = *expr2* THEN NULL ELSE *expr1* END.

The return value has the same type as the first argument.

mysql> **SELECT** **NULLIF(1,1);**

-> NULL

mysql> **SELECT** **NULLIF(1,2);**

-> 1

**Note**

MySQL evaluates *expr1* twice if the arguments are not equal.

The handling of system variable values by these functions changed in MySQL 8.0.22. For each of these functions, if the first argument contains only characters present in the character set and collation used by the second argument (and it is constant), the latter character set and collation is used to make the comparison. In MySQL 8.0.22 and later, system variable values are handled as column values of the same character and collation. Some queries using these functions with system variables that were previously successful may subsequently be rejected with Illegal mix of collations. In such cases, you should cast the system variable to the correct character set and collation.

**12.6** **Numeric** **Functions** **and** **Operators**

**Table** **12.8** **Numeric** **Functions** **and** **Operators**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [%, MOD](#_bookmark4) | Modulo operator |
| [\*](#_bookmark5) | Multiplication operator |
| [+](#_bookmark6) | Addition operator |
| [-](#_bookmark7) | Minus operator |
| [-](#_bookmark8) | Change the sign of the argument |
| [/](#_bookmark9) | Division operator |
| [ABS()](#_bookmark10) | Return the absolute value |
| [ACOS()](#_bookmark11) | Return the arc cosine |
| [ASIN()](#_bookmark12) | Return the arc sine |
| [ATAN()](#_bookmark13) | Return the arc tangent |
| [ATAN2(), ATAN()](#_bookmark14) | Return the arc tangent of the two arguments |
| [CEIL()](#_bookmark15) | Return the smallest integer value not less than the argument |
| [CEILING()](#_bookmark16) | Return the smallest integer value not less than the argument |
| [CONV()](#_bookmark17) | Convert numbers between different number bases |
| [COS()](#_bookmark18) | Return the cosine |
| [COT()](#_bookmark19) | Return the cotangent |
| [CRC32()](#_bookmark20) | Compute a cyclic redundancy check value |
| [DEGREES()](#_bookmark21) | Convert radians to degrees |
| [DIV](#_bookmark22) | Integer division |
| [EXP()](#_bookmark23) | Raise to the power of |

|  |  |
| --- | --- |
| **Name** | **Description** |
| [FLOOR()](#_bookmark24) | Return the largest integer value not greater than the argument |
| [LN()](#_bookmark25) | Return the natural logarithm of the argument |
| [LOG()](#_bookmark26) | Return the natural logarithm of the first argument |
| [LOG10()](#_bookmark27) | Return the base-10 logarithm of the argument |
| [LOG2()](#_bookmark28) | Return the base-2 logarithm of the argument |
| [MOD()](#_bookmark29) | Return the remainder |
| [PI()](#_bookmark30) | Return the value of pi |
| [POW()](#_bookmark31) | Return the argument raised to the specified power |
| [POWER()](#_bookmark32) | Return the argument raised to the specified power |
| [RADIANS()](#_bookmark33) | Return argument converted to radians |
| [RAND()](#_bookmark34) | Return a random floating-point value |
| [ROUND()](#_bookmark35) | Round the argument |
| [SIGN()](#_bookmark36) | Return the sign of the argument |
| [SIN()](#_bookmark37) | Return the sine of the argument |
| [SQRT()](#_bookmark38) | Return the square root of the argument |
| [TAN()](#_bookmark39) | Return the tangent of the argument |
| [TRUNCATE()](#_bookmark40) | Truncate to specified number of decimal places |

**12.6.1** **Arithmetic** **Operators**

**Table** **12.9** **Arithmetic** **Operators**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [%, MOD](#_bookmark4) | Modulo operator |
| [\*](#_bookmark5) | Multiplication operator |
| [+](#_bookmark6) | Addition operator |
| [-](#_bookmark7) | Minus operator |
| [-](#_bookmark8) | Change the sign of the argument |
| [/](#_bookmark9) | Division operator |
| [DIV](#_bookmark22) | Integer division |

The usual arithmetic operators are available. The result is determined according to the following rules:

• In the case of [-](#_bookmark7), [+](#_bookmark6), and [\*](#_bookmark5), the result is calculated with BIGINT (64-bit) precision if both operands are integers.

• If both operands are integers and any of them are unsigned, the result is an unsigned integer. For subtraction, if the NO\_UNSIGNED\_SUBTRACTION SQL mode is enabled, the result is signed even if any operand is unsigned.

• If any of the operands of a [+](#_bookmark6), [-](#_bookmark7), [/](#_bookmark9), [\*](#_bookmark5), [%](#_bookmark4) is a real or string value, the precision of the result is the precision of the operand with the maximum precision.

• In division performed with [/](#_bookmark9), the scale of the result when using two exact-value operands is the scale of the first operand plus the value of the div\_precision\_increment system variable (which is

4 by default). For example, the result of the expression 5.05 / 0.014 has a scale of six decimal places (360.714286).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, (14620 / 9432456) / (24250 / 9432456), resolves first to (0.0014) / (0.0026), with the final result having 8 decimal places ( 0.60288653).

Because of these rules and the way they are applied, care should be taken to ensure that components and subcomponents of a calculation use the appropriate level of precision. See [Section 12.11, “Cast](#_bookmark42) [Functions and Operators”](#_bookmark42) .

For information about handling of overflow in numeric expression evaluation, see Section 11.1.7, “Out- of-Range and Overflow Handling” .

Arithmetic operators apply to numbers. For other types of values, alternative operations may be available. For example, to add date values, use [DATE\_ADD()](#_bookmark43); see [Section 12.7, “Date and Time](#_bookmark44) [Functions”](#_bookmark44) .

• [+](#_bookmark6)

Addition:

mysql> **SELECT** **3+5;**

-> 8

• [-](#_bookmark7)

Subtraction:

mysql> **SELECT** **3-5;**

-> -2

• [-](#_bookmark8)

Unary minus. This operator changes the sign of the operand.

mysql> **SELECT** **-** **2;**

-> -2



• [\*](#_bookmark5)



**Note**

If this operator is used with a BIGINT, the return value is also a BIGINT. This means that you should avoid using - on integers that may have the value of −263 .

Multiplication:

mysql> **SELECT** **3\*5;**

-> 15

mysql> **SELECT** **18014398509481984\*18014398509481984** **.0;**

-> 324518553658426726783156020576256.0

mysql> **SELECT** **18014398509481984\*18014398509481984;**

-> out-of-range error

The last expression produces an error because the result of the integer multiplication exceeds the 64-bit range of BIGINT calculations. (See Section 11.1, “Numeric Data Types” .)

• [/](#_bookmark9)

Division:

mysql> **SELECT** **3/5;**

-> 0.60

Division by zero produces a NULL result:

mysql> **SELECT** **102/(1-1);**

-> NULL

A division is calculated with BIGINT arithmetic only if performed in a context where its result is converted to an integer.

• [DIV](#_bookmark22)

Integer division. Discards from the division result any fractional part to the right of the decimal point.

If either operand has a noninteger type, the operands are converted to DECIMAL and divided using DECIMAL arithmetic before converting the result to BIGINT. If the result exceeds BIGINT range, an error occurs.

mysql> **SELECT** **5** **DIV** **2,** **-5** **DIV** **2,** **5** **DIV** **-2,** **-5** **DIV** **-2;**

-> 2, -2, -2, 2

• [*N* % *M*](#_bookmark4), [*N* MOD *M*](#_bookmark4)

Modulo operation. Returns the remainder of *N*divided by *M*. For more information, see the description for the [MOD()](#_bookmark29) function in [Section 12.6.2, “Mathematical Functions”](#_bookmark45) .

**12.6.2** **Mathematical** **Functions**

**Table** **12.10** **Mathematical** **Functions**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [ABS()](#_bookmark10) | Return the absolute value |
| [ACOS()](#_bookmark11) | Return the arc cosine |
| [ASIN()](#_bookmark12) | Return the arc sine |
| [ATAN()](#_bookmark13) | Return the arc tangent |
| [ATAN2(), ATAN()](#_bookmark14) | Return the arc tangent of the two arguments |
| [CEIL()](#_bookmark15) | Return the smallest integer value not less than the argument |
| [CEILING()](#_bookmark16) | Return the smallest integer value not less than the argument |
| [CONV()](#_bookmark17) | Convert numbers between different number bases |
| [COS()](#_bookmark18) | Return the cosine |
| [COT()](#_bookmark19) | Return the cotangent |
| [CRC32()](#_bookmark20) | Compute a cyclic redundancy check value |
| [DEGREES()](#_bookmark21) | Convert radians to degrees |
| [EXP()](#_bookmark23) | Raise to the power of |
| [FLOOR()](#_bookmark24) | Return the largest integer value not greater than the argument |
| [LN()](#_bookmark25) | Return the natural logarithm of the argument |
| [LOG()](#_bookmark26) | Return the natural logarithm of the first argument |
| [LOG10()](#_bookmark27) | Return the base-10 logarithm of the argument |
| [LOG2()](#_bookmark28) | Return the base-2 logarithm of the argument |
| [MOD()](#_bookmark29) | Return the remainder |
| [PI()](#_bookmark30) | Return the value of pi |
| [POW()](#_bookmark31) | Return the argument raised to the specified power |
| [POWER()](#_bookmark32) | Return the argument raised to the specified power |

|  |  |
| --- | --- |
| **Name** | **Description** |
| [RADIANS()](#_bookmark33) | Return argument converted to radians |
| [RAND()](#_bookmark34) | Return a random floating-point value |
| [ROUND()](#_bookmark35) | Round the argument |
| [SIGN()](#_bookmark36) | Return the sign of the argument |
| [SIN()](#_bookmark37) | Return the sine of the argument |
| [SQRT()](#_bookmark38) | Return the square root of the argument |
| [TAN()](#_bookmark39) | Return the tangent of the argument |
| [TRUNCATE()](#_bookmark40) | Truncate to specified number of decimal places |

All mathematical functions return NULL in the event of an error.

• [ABS(*X*)](#_bookmark10)

Returns the absolute value of *X*, or NULL if *X* is NULL.

The result type is derived from the argument type. An implication of this is that [ABS(-9223372036854775808)](#_bookmark10) produces an error because the result cannot be stored in a signed BIGINT value.

mysql> **SELECT** **ABS(2);**

-> 2

mysql> **SELECT** **ABS(-32);**

-> 32

This function is safe to use with BIGINT values.

• [ACOS(*X*)](#_bookmark11)

Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns NULL if *X* is not in the range -1 to 1, or if *X* is NULL.

mysql> **SELECT** **ACOS(1);**

-> 0

mysql> **SELECT** **ACOS(1** **.0001);**

-> NULL

mysql> **SELECT** **ACOS(0);**

-> 1.5707963267949

• [ASIN(*X*)](#_bookmark12)

Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns NULL if *X* is not in the range -1 to 1, or if *X* is NULL.

mysql> **SELECT** **ASIN(0** **.2);**

-> 0.20135792079033

mysql> **SELECT** **ASIN('foo');**

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

| ASIN('foo') |

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

| 0 |

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

1 row in set, 1 warning (0.00 sec)

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |

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

• [ATAN(*X*)](#_bookmark13)

Returns the arc tangent of *X*, that is, the value whose tangent is *X*. Returns *NULL* if *X* is NULL

mysql> **SELECT** **ATAN(2);**

-> 1.1071487177941

mysql> **SELECT** **ATAN(-2);**

-> -1.1071487177941

• [ATAN(*Y*,*X*)](#_bookmark14), [ATAN2(*Y*,*X*)](#_bookmark14)

Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of *Y* / *X*, except that the signs of both arguments are used to determine the quadrant of the result. Returns NULL if *X* or *Y* is NULL.

mysql> **SELECT** **ATAN(-2,2);**

-> -0 .78539816339745

mysql> **SELECT** **ATAN2(PI(),0);**

-> 1.5707963267949

• [CEIL(*X*)](#_bookmark15)

[CEIL()](#_bookmark15) is a synonym for [CEILING()](#_bookmark16).

• [CEILING(*X*](#_bookmark16))

Returns the smallest integer value not less than *X*. Returns NULL if *X* is NULL.

mysql> **SELECT** **CEILING(1** **.23);**

-> 2

mysql> **SELECT** **CEILING(-1** **.23);**

-> -1

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

• [CONV(*N*,*from\_base*,*to\_base*)](#_bookmark17)

Converts numbers between different number bases. Returns a string representation of the number *N*, converted from base *from\_base* to base *to\_base*. Returns NULL if any argument is NULL. The argument *N* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If *from\_base* is a negative number, *N* is regarded as a signed number. Otherwise, *N* is treated as unsigned. [CONV()](#_bookmark17) works with 64-bit precision.

CONV() returns NULL if any of its arguments are NULL.

mysql> **SELECT** **CONV('a',16,2);**

-> '1010'

mysql> **SELECT** **CONV('6E',18,8);**

-> '172'

mysql> **SELECT** **CONV(-17,10,-18);**

-> '-H'

mysql> **SELECT** **CONV(10+'10'+'10'+X'0a',10,10);**

-> '40'

• [COS(*X*)](#_bookmark18)

Returns the cosine of *X*, where *X* is given in radians. Returns NULL if *X* is NULL.

mysql> **SELECT** **COS(PI());**

-> -1

• [COT(*X*)](#_bookmark19)

Returns the cotangent of *X*. Returns NULL if *X* is NULL.

mysql> **SELECT** **COT(12);**

-> -1.5726734063977

mysql> **SELECT** **COT(0);**

-> out-of-range error

• [CRC32(*expr*](#_bookmark20))

Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is NULL if the argument is NULL. The argument is expected to be a string and (if possible) is treated as one if it is not.

mysql> **SELECT** **CRC32('MySQL');**

-> 3259397556

mysql> **SELECT** **CRC32('mysql');**

-> 2501908538

• [DEGREES(*X*](#_bookmark21))

Returns the argument *X*, converted from radians to degrees. Returns NULL if *X* is NULL.

mysql> **SELECT** **DEGREES(PI());**

-> 180

mysql> **SELECT** **DEGREES(PI()** **/** **2);**

-> 90

• [EXP(*X*)](#_bookmark23)

Returns the value of *e* (the base of natural logarithms) raised to the power of *X*. The inverse of this function is [LOG()](#_bookmark26) (using a single argument only) or [LN()](#_bookmark25).

If *X* is NULL, this function returns NULL.

mysql> **SELECT** **EXP(2);**

-> 7.3890560989307

mysql> **SELECT** **EXP(-2);**

-> 0.13533528323661

mysql> **SELECT** **EXP(0);**

-> 1

• [FLOOR(*X*)](#_bookmark24)

Returns the largest integer value not greater than *X*. Returns NULL if *X* is NULL.

mysql> **SELECT** **FLOOR(1** **.23),** **FLOOR(-1** **.23);**

-> 1, -2

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

• [FORMAT(*X*,*D*)](#_bookmark46)

Formats the number *X*to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. For details, see [Section 12.8, “String Functions and Operators”](#_bookmark47) .

• [HEX(N\_or\_S)](#_bookmark48)

This function can be used to obtain a hexadecimal representation of a decimal number or a string; the manner in which it does so varies according to the argument's type. See this function's description in [Section 12.8, “String Functions and Operators”](#_bookmark47) , for details.

• [LN(*X*)](#_bookmark25)

Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns NULL and a warning “Invalid argument for logarithm” is reported. Returns NULL if *X* is NULL.

mysql> **SELECT** **LN(2);**

-> 0.69314718055995

mysql> **SELECT** **LN(-2);**

-> NULL

This function is synonymous with [LOG(*X*](#_bookmark26)). The inverse of this function is the [EXP()](#_bookmark23) function.

• [LOG(*X*)](#_bookmark26), [LOG(*B*,*X*)](#_bookmark26)

If called with one parameter, this function returns the natural logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns NULL and a warning “Invalid argument for logarithm” is reported. Returns NULL if *X* or *B* is NULL.

The inverse of this function (when called with a single argument) is the [EXP()](#_bookmark23) function.

mysql> **SELECT** **LOG(2);**

-> 0.69314718055995

mysql> **SELECT** **LOG(-2);**

-> NULL

If called with two parameters, this function returns the logarithm of *X*to the base *B*. If *X* is less than or equal to 0, or if *B* is less than or equal to 1, then NULL is returned.

mysql> **SELECT** **LOG(2,65536);**

-> 16

mysql> **SELECT** **LOG(10,100);**

-> 2

mysql> **SELECT** **LOG(1,100);**

-> NULL

[LOG(*B*,*X*)](#_bookmark26) is equivalent to [LOG(*X*) / LOG(*B*)](#_bookmark26).

• [LOG2(*X*)](#_bookmark28)

Returns the base-2 logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns NULL and a warning “Invalid argument for logarithm” is reported. Returns NULL if *X* is NULL.

mysql> **SELECT** **LOG2(65536);**

-> 16

mysql> **SELECT** **LOG2(-100);**

-> NULL

[LOG2()](#_bookmark28) is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression [LOG(*X*) / LOG(2)](#_bookmark26).

• [LOG10(*X*)](#_bookmark27)

Returns the base-10 logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns NULL and a warning “Invalid argument for logarithm” is reported. Returns NULL if *X* is NULL.

mysql> **SELECT** **LOG10(2);**

-> 0.30102999566398

mysql> **SELECT** **LOG10(100);**

-> 2

mysql> **SELECT** **LOG10(-100);**

-> NULL

[LOG10(*X*)](#_bookmark27) is equivalent to [LOG(10,*X*)](#_bookmark26).

• [MOD(*N*,*M*)](#_bookmark29), [*N* % *M*](#_bookmark4), [*N* MOD *M*](#_bookmark4)

Modulo operation. Returns the remainder of *N*divided by *M*. Returns NULL if *M*or *N* is NULL.

|  |  |
| --- | --- |
| mysql> **SELECT**  -> 4  mysql> **SELECT**  -> 1  mysql> **SELECT**  -> 2  mysql> **SELECT** | **MOD(234,** **10);**  **253** **%** **7;**  **MOD(29,9);**  **29** **MOD** **9;** |

-> 2

This function is safe to use with BIGINT values.

[MOD()](#_bookmark29) also works on values that have a fractional part and returns the exact remainder after division:

mysql> **SELECT** **MOD(34** **.5,3);**

-> 1.5

[MOD(*N*,0)](#_bookmark29) returns NULL.

• [PI()](#_bookmark30)

Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

mysql> **SELECT** **PI();**

-> 3.141593

mysql> **SELECT** **PI()+0** **.000000000000000000;**

-> 3.141592653589793000

• [POW(*X*,*Y*)](#_bookmark31)

Returns the value of *X* raised to the power of *Y*. Returns NULL if *X* or *Y* is NULL.

mysql> **SELECT** **POW(2,2);**

-> 4

mysql> **SELECT** **POW(2,-2);**

-> 0.25

• [POWER(*X*,*Y*)](#_bookmark32)

This is a synonym for [POW()](#_bookmark31).

• [RADIANS(*X*](#_bookmark33))

Returns the argument *X*, converted from degrees to radians. (Note that π radians equals 180 degrees.) Returns NULL if *X* is NULL.

mysql> **SELECT** **RADIANS(90);**

-> 1.5707963267949

• [RAND([*N*])](#_bookmark34)

Returns a random floating-point value *v* in the range 0 <= *v* < 1.0. To obtain a random integer *R* in the range *i* <= *R* < *j*, use the expression [FLOOR(*i* + RAND() \* (*j*](#_bookmark24) − *i*)). For example, to obtain a random integer in the range the range 7 <= *R* < 12, use the following statement:

SELECT FLOOR(7 + (RAND() \* 5));

If an integer argument *N* is specified, it is used as the seed value:

• With a constant initializer argument, the seed is initialized once when the statement is prepared, prior to execution.

• With a nonconstant initializer argument (such as a column name), the seed is initialized with the value for each invocation of [RAND()](#_bookmark34).

One implication of this behavior is that for equal argument values, [RAND(*N*)](#_bookmark34) returns the same value each time, and thus produces a repeatable sequence of column values. In the following example, the sequence of values produced by RAND(3) is the same both places it occurs.

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

Query OK, 0 rows affected (0.42 sec)

mysql> **INSERT** **INTO** **t** **VALUES(1),(2),(3);**

Query OK, 3 rows affected (0.00 sec)

Records: 3 Duplicates: 0 Warnings: [0](#_bookmark49)

mysql> **SELECT** **i,** **RAND()** **FROM** **t;**

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

|  |  |  |
| --- | --- | --- |
| | i | | RAND() | | |

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

|

|

|

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

3 rows in set (0.00 sec)

mysql> **SELECT** **i,** **RAND(3)** **FROM** **t;**

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

| i | RAND(3) |

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

|

|

|

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

3 rows in set (0.00 sec)

mysql> **SELECT** **i,** **RAND()** **FROM** **t;**

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

| i | RAND() |

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

|

|

|

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

3 rows in set (0.00 sec)

mysql> **SELECT** **i,** **RAND(3)** **FROM** **t;**

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

| i | RAND(3) |

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

|

|

|

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

3 rows in set (0.01 sec)

[RAND()](#_bookmark34) in a WHERE clause is evaluated for every row (when selecting from one table) or combination of rows (when selecting from a multiple-table join). Thus, for optimizer purposes, [RAND()](#_bookmark34) is

not a constant value and cannot be used for index optimizations. For more information, see Section 8.2. 1.20, “Function Call Optimization” .

0.61914388706828

0.93845168309142

0.83482678498591

0.90576975597606

0.37307905813035

0.14808605345719

0.35877890638893

0.28941420772058

0.37073435016976

0.90576975597606

0.37307905813035

0.14808605345719

1

2

3

1

2

3

1

2

3

1

2

3

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

Use of a column with [RAND()](#_bookmark34) values in an ORDER BY or GROUP BY clause may yield unexpected results because for either clause a [RAND()](#_bookmark34) expression can be evaluated multiple times for the same row, each time returning a different result. If the goal is to retrieve rows in random order, you can use a statement like this:

SELECT \* FROM *tbl\_name* ORDER BY RAND();

To select a random sample from a set of rows, combine ORDER BY RAND() with LIMIT:

SELECT \* FROM table1, table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000;

[RAND()](#_bookmark34) is not meant to be a perfect random generator. It is a fast way to generate random numbers on demand that is portable between platforms for the same MySQL version.

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

• [ROUND(*X*)](#_bookmark35), [ROUND(*X*,*D*)](#_bookmark35)

Rounds the argument *X*to *D* decimal places. The rounding algorithm depends on the data type of *X*. *D* defaults to 0 if not specified. *D* can be negative to cause *D* digits left of the decimal point of the value *X*to become zero. The maximum absolute value for *D* is 30; any digits in excess of 30 (or -30) are truncated. If *X* or *D* is NULL, the function returns NULL.

mysql> **SELECT** **ROUND(-1** **.23);**

-> -1

mysql> **SELECT** **ROUND(-1** **.58);**

-> -2

mysql> **SELECT** **ROUND(1** **.58);**

-> 2

mysql> **SELECT** **ROUND(1** **.298,** **1);**

-> 1.3

mysql> **SELECT** **ROUND(1** **.298,** **0);**

-> 1

mysql> **SELECT** **ROUND(23** **.298,** **-1);**

-> 20

mysql> **SELECT** **ROUND(** **.12345678901234567890123456789012345,** **35);**

-> 0.123456789012345678901234567890

The return value has the same type as the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

mysql> **SELECT** **ROUND(150** **.000,2),** **ROUND(150,2);**

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

| ROUND(150 .000,2) | ROUND(150,2) |

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

|  |  |  |  |
| --- | --- | --- | --- |
| | | 150.00 | | | 150 | |

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

[ROUND()](#_bookmark35) uses the following rules depending on the type of the first argument:

• For exact-value numbers, [ROUND()](#_bookmark35) uses the “round half away from zero” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.

• For approximate-value numbers, the result depends on the C library. On many systems, this means that [ROUND()](#_bookmark35) uses the “round to nearest even” rule: A value with a fractional part exactly halfway between two integers is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

mysql> **SELECT** **ROUND(2** **.5),** **ROUND(25E-1);**

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

| ROUND(2 .5) | ROUND(25E-1) |

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

| 3 | 2 |

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

For more information, see Section 12.25, “Precision Math” .

In MySQL 8.0.21 and later, the data type returned by ROUND() (and [TRUNCATE()](#_bookmark40)) is determined according to the rules listed here:

• When the first argument is of any integer type, the return type is always BIGINT.

• When the first argument is of any floating-point type or of any non-numeric type, the return type is always DOUBLE.

• When the first argument is a DECIMAL value, the return type is also DECIMAL.

• The type attributes for the return value are also copied from the first argument, except in the case of DECIMAL, when the second argument is a constant value.

When the desired number of decimal places is less than the scale of the argument, the scale and the precision of the result are adjusted accordingly.

In addition, for ROUND() (but not for the [TRUNCATE()](#_bookmark40) function), the precision is extended by one place to accommodate rounding that increases the number of significant digits. If the second argument is negative, the return type is adjusted such that its scale is 0, with a corresponding precision. For example, ROUND(99.999, 2) returns 100.00—the first argument is DECIMAL(5, 3), and the return type is DECIMAL(5, 2).

If the second argument is negative, the return type has scale 0 and a corresponding precision; ROUND(99.999, -1) returns 100, which is DECIMAL(3, 0).

• [SIGN(*X*)](#_bookmark36)

Returns the sign of the argument as -1, 0, or 1, depending on whether *X* is negative, zero, or positive. Returns NULL if *X* is NULL.

mysql> **SELECT**

**SIGN(-32);** **SIGN(0);**

**SIGN(234);**

-> -1

mysql> **SELECT**

-> 0

mysql> **SELECT**

-> 1

• [SIN(*X*)](#_bookmark37)

Returns the sine of *X*, where *X* is given in radians. Returns NULL if *X* is NULL.

mysql> **SELECT** **SIN(PI());**

-> 1 .2246063538224e-16

mysql> **SELECT** **ROUND(SIN(PI()));**

-> 0

• [SQRT(*X*)](#_bookmark38)

Returns the square root of a nonnegative number *X*. If *X* is NULL, the function returns NULL.

mysql> **SELECT** **SQRT(4);**

-> 2

mysql> **SELECT** **SQRT(20);**

-> 4.4721359549996

mysql> **SELECT** **SQRT(-16);**

-> NULL

• [TAN(*X*)](#_bookmark39)

Returns the tangent of *X*, where *X* is given in radians. Returns NULL if *X* is NULL.

mysql> **SELECT** **TAN(PI());**

-> -1 .2246063538224e-16

mysql> **SELECT** **TAN(PI()+1);**

-> 1.5574077246549

• [TRUNCATE(*X*,*D*)](#_bookmark40)

Returns the number *X*, truncated to *D* decimal places. If *D* is 0, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X*to become zero. If *X* or *D* is NULL, the function returns NULL.

mysql> **SELECT** **TRUNCATE(1** **.223,1);**

-> 1.2

mysql> **SELECT** **TRUNCATE(1** **.999,1);**

-> 1.9

mysql> **SELECT** **TRUNCATE(1** **.999,0);**

-> 1

mysql> **SELECT** **TRUNCATE(-1** **.999,1);**

-> -1 .9

mysql> **SELECT** **TRUNCATE(122,-2);**

-> 100

mysql> **SELECT** **TRUNCATE(10** **.28\*100,0);**

-> 1028

All numbers are rounded toward zero.

In MySQL 8.0.21 and later, the data type returned by TRUNCATE() follows the same rules that determine the return type of the ROUND() function; for details, see the description for [ROUND()](#_bookmark35).

**12.7** **Date** **and** **Time** **Functions**

This section describes the functions that can be used to manipulate temporal values. See Section 11.2, “Date and Time Data Types” , for a description of the range of values each date and time type has and the valid formats in which values may be specified.

**Table** **12.11** **Date** **and** **Time** **Functions**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [ADDDATE()](#_bookmark50) | Add time values (intervals) to a date value |
| [ADDTIME()](#_bookmark51) | Add time |
| [CONVERT\_TZ()](#_bookmark52) | Convert from one time zone to another |
| [CURDATE()](#_bookmark53) | Return the current date |
| [CURRENT\_DATE(), CURRENT\_DATE](#_bookmark54) | Synonyms for CURDATE() |
| [CURRENT\_TIME(), CURRENT\_TIME](#_bookmark55) | Synonyms for CURTIME() |
| [CURRENT\_TIMESTAMP(), CURRENT\_TIMESTAMP](#_bookmark56) | Synonyms for NOW() |
| [CURTIME()](#_bookmark57) | Return the current time |
| [DATE()](#_bookmark58) | Extract the date part of a date or datetime expression |
| [DATE\_ADD()](#_bookmark43) | Add time values (intervals) to a date value |
| [DATE\_FORMAT()](#_bookmark59) | Format date as specified |
| [DATE\_SUB()](#_bookmark60) | Subtract a time value (interval) from a date |
| [DATEDIFF()](#_bookmark61) | Subtract two dates |
| [DAY()](#_bookmark62) | Synonym for DAYOFMONTH() |
| [DAYNAME()](#_bookmark63) | Return the name of the weekday |
| [DAYOFMONTH()](#_bookmark64) | Return the day of the month (0-31) |
| [DAYOFWEEK()](#_bookmark65) | Return the weekday index of the argument |
| [DAYOFYEAR()](#_bookmark66) | Return the day of the year (1-366) |
| [EXTRACT()](#_bookmark67) | Extract part of a date |
| [FROM\_DAYS()](#_bookmark68) | Convert a day number to a date |
| [FROM\_UNIXTIME()](#_bookmark69) | Format Unix timestamp as a date |
| [GET\_FORMAT()](#_bookmark70) | Return a date format string |
| [HOUR()](#_bookmark71) | Extract the hour |
| [LAST\_DAY](#_bookmark72) | Return the last day of the month for the argument |
| [LOCALTIME(), LOCALTIME](#_bookmark73) | Synonym for NOW() |
| [LOCALTIMESTAMP, LOCALTIMESTAMP()](#_bookmark74) | Synonym for NOW() |

|  |  |
| --- | --- |
| **Name** | **Description** |
| [MAKEDATE()](#_bookmark75) | Create a date from the year and day of year |
| [MAKETIME()](#_bookmark76) | Create time from hour, minute, second |
| [MICROSECOND()](#_bookmark77) | Return the microseconds from argument |
| [MINUTE()](#_bookmark78) | Return the minute from the argument |
| [MONTH()](#_bookmark79) | Return the month from the date passed |
| [MONTHNAME()](#_bookmark80) | Return the name of the month |
| [NOW()](#_bookmark81) | Return the current date and time |
| [PERIOD\_ADD()](#_bookmark82) | Add a period to a year-month |
| [PERIOD\_DIFF()](#_bookmark83) | Return the number of months between periods |
| [QUARTER()](#_bookmark84) | Return the quarter from a date argument |
| [SEC\_TO\_TIME()](#_bookmark85) | Converts seconds to 'hh:mm:ss' format |
| [SECOND()](#_bookmark86) | Return the second (0-59) |
| [STR\_TO\_DATE()](#_bookmark87) | Convert a string to a date |
| [SUBDATE()](#_bookmark88) | Synonym for DATE\_SUB() when invoked with three arguments |
| [SUBTIME()](#_bookmark89) | Subtract times |
| [SYSDATE()](#_bookmark90) | Return the time at which the function executes |
| [TIME()](#_bookmark91) | Extract the time portion of the expression passed |
| [TIME\_FORMAT()](#_bookmark92) | Format as time |
| [TIME\_TO\_SEC()](#_bookmark93) | Return the argument converted to seconds |
| [TIMEDIFF()](#_bookmark94) | Subtract time |
| [TIMESTAMP()](#_bookmark95) | With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments |
| [TIMESTAMPADD()](#_bookmark96) | Add an interval to a datetime expression |
| [TIMESTAMPDIFF()](#_bookmark97) | Return the difference of two datetime expressions, using the units specified |
| [TO\_DAYS()](#_bookmark98) | Return the date argument converted to days |
| [TO\_SECONDS()](#_bookmark99) | Return the date or datetime argument converted to seconds since Year 0 |
| [UNIX\_TIMESTAMP()](#_bookmark100) | Return a Unix timestamp |
| [UTC\_DATE()](#_bookmark101) | Return the current UTC date |
| [UTC\_TIME()](#_bookmark102) | Return the current UTC time |
| [UTC\_TIMESTAMP()](#_bookmark103) | Return the current UTC date and time |
| [WEEK()](#_bookmark104) | Return the week number |
| [WEEKDAY()](#_bookmark105) | Return the weekday index |
| [WEEKOFYEAR()](#_bookmark106) | Return the calendar week of the date (1-53) |
| [YEAR()](#_bookmark107) | Return the year |
| [YEARWEEK()](#_bookmark108) | Return the year and week |

Here is an example that uses date functions. The following query selects all rows with a *date\_col* value from within the last 30 days:

mysql> **SELECT** ***something*** **FROM** ***tbl\_name***

-> **WHERE** **DATE\_SUB(CURDATE(),INTERVAL** **30** **DAY)** **<=** ***date\_col*;**

The query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as [NOW()](#_bookmark81) within a single query always produce the same result. (For our purposes, a single query also includes a call to a stored program (stored routine, trigger, or event) and all subprograms called by that program.) This principle also applies to [CURDATE()](#_bookmark53), [CURTIME()](#_bookmark57), [UTC DATE()](#_bookmark101)\_, [UTC TIME()](#_bookmark102)\_, [UTC TIMESTAMP()](#_bookmark103)\_, and to any of their synonyms.

The [CURRENT\_TIMESTAMP()](#_bookmark56), [CURRENT\_TIME()](#_bookmark55), [CURRENT\_DATE()](#_bookmark54), and [FROM\_UNIXTIME()](#_bookmark69) functions return values in the current session time zone, which is available as the session value of the time\_zone system variable. In addition, [UNIX\_TIMESTAMP()](#_bookmark100) assumes that its argument is a datetime value in the session time zone. See Section 5.1.15, “MySQL Server Time Zone Support” .

Some date functions can be used with “zero” dates or incomplete dates such as '2001-11-00', whereas others cannot. Functions that extract parts of dates typically work with incomplete dates and thus can return 0 when you might otherwise expect a nonzero value. For example:

mysql> **SELECT** **DAYOFMONTH('2001-11-00'),** **MONTH('2005-00-00');**

-> 0, 0

Other functions expect complete dates and return NULL for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

mysql> **SELECT** **DATE\_ADD('2006-05-00',INTERVAL** **1** **DAY);**

-> NULL

mysql> **SELECT** **DAYNAME('2006-05-00');**

-> NULL

Several functions are strict when passed a [DATE()](#_bookmark58) function value as their argument and reject incomplete dates with a day part of zero: [CONVERT\_TZ()](#_bookmark52), [DATE\_ADD()](#_bookmark43), [DATE\_SUB()](#_bookmark60), [DAYOFYEAR()](#_bookmark66), [TIMESTAMPDIFF()](#_bookmark97), [TO DAYS()](#_bookmark98)\_, [TO SECONDS()](#_bookmark99)\_, [WEEK()](#_bookmark104), [WEEKDAY()](#_bookmark105), [WEEKOFYEAR()](#_bookmark106), [YEARWEEK()](#_bookmark108).

Fractional seconds for TIME, DATETIME, and TIMESTAMP values are supported, with up to microsecond precision. Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate.

• [ADDDATE(*date*,INTERVAL *expr* *unit*)](#_bookmark50), [ADDDATE(*date*,*days*)](#_bookmark50)

When invoked with the INTERVAL form of the second argument, [ADDDATE()](#_bookmark50) is a synonym for [DATE ADD()](#_bookmark43)\_. The related function [SUBDATE()](#_bookmark88) is a synonym for [DATE SUB()](#_bookmark60)\_. For information on the INTERVAL *unit* argument, see Temporal Intervals.

mysql> **SELECT** **DATE\_ADD('2008-01-02',** **INTERVAL** **31** **DAY);**

-> '2008-02-02'

mysql> **SELECT** **ADDDATE('2008-01-02',** **INTERVAL** **31** **DAY);**

-> '2008-02-02'

When invoked with the *days* form of the second argument, MySQL treats it as an integer number of days to be added to *expr*.

mysql> **SELECT** **ADDDATE('2008-01-02',** **31);**

-> '2008-02-02'

This function returns NULL if *date* or *days* is NULL.

• [ADDTIME(*expr1*,*expr2*)](#_bookmark51)

[ADDTIME()](#_bookmark51) adds *expr2* to *expr1* and returns the result. *expr1* is a time or datetime expression, and *expr2* is a time expression. Returns NULL if *expr1*or *expr2* is NULL.

Beginning with MySQL 8.0.28, the return type of this function and of the [SUBTIME()](#_bookmark89) function is determined as follows:

• If the first argument is a dynamic parameter (such as in a prepared statement), the return type is TIME.

• Otherwise, the resolved type of the function is derived from the resolved type of the first argument.

mysql> **SELECT** **ADDTIME('2007-12-31** **23:59:59** **.999999',** **'1** **1:1:1** **.000002');**

-> '2008-01-02 01:01:01 .000001'

mysql> **SELECT** **ADDTIME('01:00:00** **.999999',** **'02:00:00** **.999998');**

-> '03:00:01.999997'

• [CONVERT\_TZ(*dt*,*from\_tz*,*to\_tz*)](#_bookmark52)

[CONVERT\_TZ()](#_bookmark52) converts a datetime value *dt* from the time zone given by *from\_tz* to the time zone given by *to\_tz* and returns the resulting value. Time zones are specified as described in Section 5.1.15, “MySQL Server Time Zone Support” . This function returns NULL if any of the arguments are invalid, or if any of them are NULL.

On 32-bit platforms, the supported range of values for this function is the same as for the TIMESTAMP type (see Section 11.2.1, “Date and Time Data Type Syntax” , for range information). On 64-bit platforms, beginning with MySQL 8.0.28, the maximum supported value is '3001-01-18

23:59:59.999999' UTC.

Regardless of platform or MySQL version, if the value falls out of the supported range when converted from *from\_tz* to UTC, no conversion occurs.

|  |  |
| --- | --- |
| mysql> **SELECT** **CONVERT\_TZ('2004-01-01**  -> '2004-01-01 13:00:00'  mysql> **SELECT** **CONVERT\_TZ('2004-01-01**  -> '2004-01-01 22:00:00' | **12:00:00','GMT','MET');**  **12:00:00','+00:00','+10:00');** |

**Note**

To use named time zones such as 'MET' or 'Europe/Amsterdam' the time zone tables must be properly set up. For instructions, see Section 5.1.15, “MySQL Server Time Zone Support” .

,

• [CURDATE()](#_bookmark53)

Returns the current date as a value in '*YYYY-MM-DD*' or *YYYYMMDD* format, depending on whether the function is used in string or numeric context.

mysql> **SELECT** **CURDATE();**

-> '2008-06-13'

mysql> **SELECT** **CURDATE()** **+** **0;**

-> 20080613

• [CURRENT\_DATE](#_bookmark54), [CURRENT\_DATE()](#_bookmark54)

[CURRENT DATE](#_bookmark54)\_ and [CURRENT DATE()](#_bookmark54)\_ are synonyms for [CURDATE()](#_bookmark53).

• [CURRENT\_TIME](#_bookmark55), [CURRENT\_TIME([*fsp*])](#_bookmark55)

[CURRENT TIME](#_bookmark55)\_ and [CURRENT TIME()](#_bookmark55)\_ are synonyms for [CURTIME()](#_bookmark57).

• [CURRENT\_TIMESTAMP](#_bookmark56), [CURRENT\_TIMESTAMP([*fsp*])](#_bookmark56) [CURRENT TIMESTAMP](#_bookmark56)\_ and [CURRENT TIMESTAMP()](#_bookmark56)\_ are synonyms for [NOW()](#_bookmark81).

• [CURTIME([*fsp*])](#_bookmark57)

Returns the current time as a value in *'hh:mm:ss'* or *hhmmss* format, depending on whether the function is used in string or numeric context. The value is expressed in the session time zone.

If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

mysql> **SELECT** **CURTIME();**

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

| CURTIME() |

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

| 19:25:37 |

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

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

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

| CURTIME() + 0 |

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

| 192537 |

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

mysql> **SELECT** **CURTIME(3);**

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

| CURTIME(3) |

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

| 19:25:37 .840 |

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

• [DATE(*expr*](#_bookmark58))

Extracts the date part of the date or datetime expression *expr*. Returns NULL if *expr* is NULL.

mysql> **SELECT** **DATE('2003-12-31** **01:02:03');**

-> '2003-12-31'

• [DATEDIFF(*expr1*,*expr2*)](#_bookmark61)

[DATEDIFF()](#_bookmark61) returns *expr1* − *expr2* expressed as a value in days from one date to the other. *expr1* and *expr2* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

mysql> **SELECT** **DATEDIFF('2007-12-31** **23:59:59','2007-12-30');**

-> 1

mysql> **SELECT** **DATEDIFF('2010-11-30** **23:59:59','2010-12-31');**

-> -31

This function returns NULL if *expr1* or *expr2* is NULL.

• [DATE\_ADD(*date*,INTERVAL *expr* *unit*)](#_bookmark43), [DATE\_SUB(*date*,INTERVAL *expr* *unit*)](#_bookmark60)

These functions perform date arithmetic. The *date* argument specifies the starting date or datetime value. *expr* is an expression specifying the interval value to be added or subtracted from the starting

date. *expr* is evaluated as a string; it may start with a - for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted.

For more information about temporal interval syntax, including a full list of *unit* specifiers, the expected form of the *expr* argument for each *unit* value, and rules for operand interpretation in temporal arithmetic, see Temporal Intervals.

The return value depends on the arguments:

• If *date* is NULL, the function returns NULL.

• DATE if the *date* argument is a DATE value and your calculations involve only YEAR, MONTH, and DAY parts (that is, no time parts).

• (*MySQL* *8.0.28* *and* *later*:) TIME if the *date* argument is a TIME value and the calculations involve only HOURS, MINUTES, and SECONDS parts (that is, no date parts).

• DATETIME if the first argument is a DATETIME (or TIMESTAMP) value, or if the first argument is a DATE and the *unit* value uses HOURS, MINUTES, or SECONDS, or if the first argument is of type TIME and the *unit* value uses YEAR, MONTH, or DAY.

• (*MySQL* *8.0.28* *and* *later*:) If the first argument is a dynamic parameter (for example, of a prepared statement), its resolved type is DATE if the second argument is an interval that contains some combination of YEAR, MONTH, or DAY values only; otherwise, its type is DATETIME.

• String otherwise (type VARCHAR).

**Note**

In MySQL 8.0.22 through 8.0.27, when used in prepared statements, these functions returned DATETIME values regardless of argument types. (Bug #103781)

To ensure that the result is DATETIME, you can use [CAST()](#_bookmark109) to convert the first argument to

DATETIME.

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'

When adding a MONTH interval to a DATE or DATETIME value, and the resulting date includes a day that does not exist in the given month, the day is adjusted to the last day of the month, as shown here:

mysql> **SELECT** **DATE\_ADD('2024-03-30',** **INTERVAL** **1** **MONTH)** **AS** **d1,**

> **DATE\_ADD('2024-03-31',** **INTERVAL** **1** **MONTH)** **AS** **d2;**

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

| d1 | d2 |

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

| 2024-04-30 | 2024-04-30 |

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

1 row in set (0.00 sec)

• [DATE\_FORMAT(*date*,*format*)](#_bookmark59)

Formats the *date* value according to the *format* string. If either argument is NULL, the function returns NULL.

The specifiers shown in the following table may be used in the *format* string. The % character is required before format specifier characters. The specifiers apply to other functions as well: [STR\_TO\_DATE()](#_bookmark87), [TIME\_FORMAT()](#_bookmark92), [UNIX\_TIMESTAMP()](#_bookmark100).

|  |  |
| --- | --- |
| **Specifier** | **Description** |
| %a | Abbreviated weekday name (Sun..Sat) |
| %b | Abbreviated month name (Jan..Dec) |
| %c | Month, numeric ( 0..12) |
| %D | Day of the month with English suffix ( 0th, 1st, 2nd, 3rd, …) |
| %d | Day of the month, numeric ( 00..31) |
| %e | Day of the month, numeric ( 0..31) |
| %f | Microseconds ( 000000.. 999999) |
| %H | Hour ( 00..23) |
| %h | Hour ( 01..12) |
| %I | Hour ( 01..12) |
| %i | Minutes, numeric ( 00..59) |
| %j | Day of year ( 001..366) |
| %k | Hour ( 0..23) |
| %l | Hour (1..12) |
| %M | Month name (January..December) |
| %m | Month, numeric ( 00..12) |
| %p | AM or PM |
| %r | Time, 12-hour (*hh:mm:ss* followed by AM or PM) |
| %S | Seconds ( 00..59) |
| %s | Seconds ( 00..59) |
| %T | Time, 24-hour (*hh:mm:ss*) |
| %U | Week ( 00..53), where Sunday is the first day of the week; [WEEK()](#_bookmark104) mode 0 |
| %u | Week ( 00..53), where Monday is the first day of the week; [WEEK()](#_bookmark104) mode 1 |

Date and Time Functions

|  |  |
| --- | --- |
| **Specifier** | **Description** |
| %V | Week ( 01..53), where Sunday is the first day of the week; [WEEK()](#_bookmark104) mode 2; used with %X |
| %v | Week ( 01..53), where Monday is the first day of the week; [WEEK()](#_bookmark104) mode 3; used with %x |
| %W | Weekday name (Sunday..Saturday) |
| %w | Day of the week ( 0=Sunday.. 6=Saturday) |
| %X | Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V |
| %x | Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v |
| %Y | Year, numeric, four digits |
| %y | Year, numeric (two digits) |
| %% | A literal % character |
| %*x* | *x*, for any “*x*” not listed above |

Ranges for the month and day specifiers begin with zero due to the fact that MySQL permits the storing of incomplete dates such as '2014-00-00'.

The language used for day and month names and abbreviations is controlled by the value of the lc\_time\_names system variable (Section 10.16, “MySQL Server Locale Support”).

For the %U, %u, %V, and %v specifiers, see the description of the [WEEK()](#_bookmark104) function for information about the mode values. The mode affects how week numbering occurs.

[DATE\_FORMAT()](#_bookmark59) returns a string with a character set and collation given by character\_set\_connection and collation\_connection so that it can return month and weekday names containing non-ASCII characters.

mysql> **SELECT** **DATE\_FORMAT('2009-10-04** **22:23:00',** **'%W** **%M** **%Y');**

-> 'Sunday October 2009'

mysql> **SELECT** **DATE\_FORMAT('2007-10-04** **22:23:00',** **'%H:%i:%s');**

-> '22:23:00'

mysql> **SELECT** **DATE\_FORMAT('1900-10-04** **22:23:00',**

-> **'%D** **%y** **%a** **%d** **%m** **%b** **%j');**

-> '4th 00 Thu 04 10 Oct 277'

mysql> **SELECT** **DATE\_FORMAT('1997-10-04** **22:23:00',**

-> **'%H** **%k** **%I** **%r** **%T** **%S** **%w');**

-> '22 22 10 10:23:00 PM 22:23:00 00 6'

mysql> **SELECT** **DATE\_FORMAT('1999-01-01',** **'%X** **%V');**

-> '1998 52'

mysql> **SELECT** **DATE\_FORMAT('2006-06-00',** **'%d');**

-> '00'

• [DATE\_SUB(*date*,INTERVAL *expr* *unit*)](#_bookmark60) See the description for [DATE\_ADD()](#_bookmark43).

• [DAY(*date*)](#_bookmark62)

[DAY()](#_bookmark62) is a synonym for [DAYOFMONTH()](#_bookmark64).

• [DAYNAME(*date*)](#_bookmark63)

Returns the name of the weekday for *date*. The language used for the name is controlled by the value of the lc\_time\_names system variable (see Section 10.16, “MySQL Server Locale Support”). Returns NULL if *date* is NULL.

mysql> **SELECT** **DAYNAME('2007-02-03');**

-> 'Saturday'

• [DAYOFMONTH(*date*)](#_bookmark64)

Returns the day of the month for *date*, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero day part. Returns NULL if *date* is NULL.

mysql> **SELECT** **DAYOFMONTH('2007-02-03');**

-> 3

• [DAYOFWEEK(*date*)](#_bookmark65)

Returns the weekday index for *date* (1 = Sunday, 2 = Monday, …, 7 = Saturday). These index values correspond to the ODBC standard. Returns NULL if *date* is NULL.

mysql> **SELECT** **DAYOFWEEK('2007-02-03');**

-> 7

• [DAYOFYEAR(*date*)](#_bookmark66)

Returns the day of the year for *date*, in the range 1 to 366. Returns NULL if *date* is NULL.

mysql> **SELECT** **DAYOFYEAR('2007-02-03');**

-> 34

• [EXTRACT(*unit* FROM *date*)](#_bookmark67)

The [EXTRACT()](#_bookmark67) function uses the same kinds of *unit* specifiers as [DATE ADD()](#_bookmark43)\_ or [DATE SUB()](#_bookmark60)\_, but extracts parts from the date rather than performing date arithmetic. For information on the *unit* argument, see Temporal Intervals. Returns NULL if *date* is NULL.

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

-> 2019

mysql> **SELECT** **EXTRACT(YEAR\_MONTH** **FROM** **'2019-07-02**

**01:02:03');**

**01:02:03');**

-> 201907

mysql> **SELECT** **EXTRACT(DAY\_MINUTE** **FROM** **'2019-07-02**

-> 20102

mysql> **SELECT** **EXTRACT(MICROSECOND**

->

**FROM** **'2003-01-02** **10:30:00.000123');**

-> 123

• [FROM\_DAYS(*N*)](#_bookmark68)

Given a day number *N*, returns a DATE value. Returns NULL if *N* is NULL.

mysql> **SELECT** **FROM\_DAYS(730669);**

-> '2000-07-03'

Use [FROM\_DAYS()](#_bookmark68) with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See [Section 12.9, “What Calendar Is Used By MySQL?”](#_bookmark110) .

• [FROM\_UNIXTIME(*unix\_timestamp*[,*format*])](#_bookmark69)

Returns a representation of *unix\_timestamp* as a datetime or character string value. The value returned is expressed using the session time zone. (Clients can set the session time zone as described in Section 5.1.15, “MySQL Server Time Zone Support” .) *unix\_timestamp* is an internal timestamp value representing seconds since '1970-01-01 00:00:00' UTC, such as produced by the [UNIX\_TIMESTAMP()](#_bookmark100) function.

If *format* is omitted, this function returns a DATETIME value.

If *unix\_timestamp* or *format* is NULL, this function returns NULL.

If *unix\_timestamp* is an integer, the fractional seconds precision of the DATETIME is zero. When

as the precision of the decimal value, up to a maximum of 6. When *unix\_timestamp* is a floating point number, the fractional seconds precision of the datetime is 6.

On 32-bit platforms, the maximum useful value for *unix\_timestamp* is 2147483647.999999, which returns '2038-01-19 03:14:07.999999' UTC. On 64-bit platforms running MySQL 8.0.28 or later, the effective maximum is 32536771199.999999, which returns '3001-01-18 23:59:59.999999' UTC. Regardless of platform or version, a greater value for *unix\_timestamp* than the effective maximum returns 0.

*format* is used to format the result in the same way as the format string used for the [DATE\_FORMAT()](#_bookmark59) function. If *format* is supplied, the value returned is a VARCHAR.

mysql> **SELECT** **FROM\_UNIXTIME(1447430881);**

-> '2015-11-13 10:08:01'

mysql> **SELECT** **FROM\_UNIXTIME(1447430881)** **+** **0;**

-> 20151113100801

mysql> **SELECT** **FROM\_UNIXTIME(1447430881,**

-> **'%Y** **%D** **%M** **%h:%i:%s** **%x');**

-> '2015 13th November 10:08:01 2015'

**Note**

If you use [UNIX\_TIMESTAMP()](#_bookmark100) and [FROM\_UNIXTIME()](#_bookmark69) to convert between values in a non-UTC time zone and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the [UNIX\_TIMESTAMP()](#_bookmark100) function.

• [GET\_FORMAT({DATE|TIME|DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})](#_bookmark70)

Returns a format string. This function is useful in combination with the [DATE\_FORMAT()](#_bookmark59) and the [STR\_TO\_DATE()](#_bookmark87) functions.

If *format* is NULL, this function returns NULL.

The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the [DATE\_FORMAT()](#_bookmark59) function description). ISO format refers to ISO 9075, not ISO 8601.

|  |  |
| --- | --- |
| **Function** **Call** | **Result** |
| [GET\_FORMAT(DATE,'USA')](#_bookmark70) | '%m.%d.%Y' |
| [GET\_FORMAT(DATE,'JIS')](#_bookmark70) | '%Y-%m-%d' |
| [GET\_FORMAT(DATE,'ISO')](#_bookmark70) | '%Y-%m-%d' |
| [GET\_FORMAT(DATE,'EUR')](#_bookmark70) | '%d.%m.%Y' |
| [GET\_FORMAT(DATE,'INTERNAL')](#_bookmark70) | '%Y%m%d' |
| [GET\_FORMAT(DATETIME,'USA')](#_bookmark70) | '%Y-%m-%d %H.%i.%s' |
| [GET\_FORMAT(DATETIME,'JIS')](#_bookmark70) | '%Y-%m-%d %H:%i:%s' |
| [GET\_FORMAT(DATETIME,'ISO')](#_bookmark70) | '%Y-%m-%d %H:%i:%s' |
| [GET\_FORMAT(DATETIME,'EUR')](#_bookmark70) | '%Y-%m-%d %H.%i.%s' |
| [GET\_FORMAT(DATETIME,'INTERNAL')](#_bookmark70) | '%Y%m%d%H%i%s' |
| [GET\_FORMAT(TIME,'USA')](#_bookmark70) | '%h:%i:%s %p' |
| [GET\_FORMAT(TIME,'JIS')](#_bookmark70) | '%H:%i:%s' |
| [GET\_FORMAT(TIME,'ISO')](#_bookmark70) | '%H:%i:%s' |
| [GET\_FORMAT(TIME,'EUR')](#_bookmark70) | '%H.%i.%s' |



|  |  |
| --- | --- |
| **Function** **Call** | **Result** |
| [GET\_FORMAT(TIME,'INTERNAL')](#_bookmark70) | '%H%i%s' |

TIMESTAMP can also be used as the first argument to [GET\_FORMAT()](#_bookmark70), in which case the function returns the same values as for DATETIME.

mysql> **SELECT** **DATE\_FORMAT('2003-10-03',GET\_FORMAT(DATE,'EUR'));**

-> '03 .10 .2003'

mysql> **SELECT** **STR\_TO\_DATE('10** **.31** **.2003',GET\_FORMAT(DATE,'USA'));**

-> '2003-10-31'

• [HOUR(*time*)](#_bookmark71)

Returns the hour for *time*. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23. Returns NULL if *time* is NULL.

mysql> **SELECT** **HOUR('10:05:03');**

-> 10

mysql> **SELECT** **HOUR('272:59:59');**

-> 272

• [LAST\_DAY(*date*)](#_bookmark72)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid or NULL.

mysql> **SELECT** **LAST\_DAY('2003-02-05');**

-> '2003-02-28'

mysql> **SELECT** **LAST\_DAY('2004-02-05');**

-> '2004-02-29'

mysql> **SELECT** **LAST\_DAY('2004-01-01** **01:01:01');**

-> '2004-01-31'

mysql> **SELECT** **LAST\_DAY('2003-03-32');**

-> NULL

• [LOCALTIME](#_bookmark73), [LOCALTIME([](#_bookmark73)*fsp*])

[LOCALTIME](#_bookmark73) and [LOCALTIME()](#_bookmark73) are synonyms for [NOW()](#_bookmark81).

• [LOCALTIMESTAMP](#_bookmark74), [LOCALTIMESTAMP([](#_bookmark74)*fsp*])

[LOCALTIMESTAMP](#_bookmark74) and [LOCALTIMESTAMP()](#_bookmark74) are synonyms for [NOW()](#_bookmark81).

• [MAKEDATE(*year*,*dayofyear*)](#_bookmark75)

Returns a date, given year and day-of-year values. *dayofyear* must be greater than 0 or the result is NULL. The result is also NULL if either argument is NULL.

mysql> **SELECT** **MAKEDATE(2011,31),** **MAKEDATE(2011,32);**

-> '2011-01-31', '2011-02-01'

mysql> **SELECT** **MAKEDATE(2011,365),** **MAKEDATE(2014,365);**

-> '2011-12-31', '2014-12-31'

mysql> **SELECT** **MAKEDATE(2011,0);**

-> NULL

• [MAKETIME(*hour*,*minute*,*second*)](#_bookmark76)

Returns a time value calculated from the *hour*, *minute*, and *second* arguments. Returns NULL if any of its arguments are NULL.

The *second* argument can have a fractional part.

mysql> **SELECT** **MAKETIME(12,15,30);**

-> '12:15:30'

• [MICROSECOND(*expr*](#_bookmark77))

Returns the microseconds from the time or datetime expression *expr* as a number in the range from 0 to 999999. Returns NULL if *expr* is NULL.

mysql> **SELECT** **MICROSECOND('12:00:00** **.123456');**

-> 123456

mysql> **SELECT** **MICROSECOND('2019-12-31** **23:59:59** **.000010');**

-> 10

• [MINUTE(*time*)](#_bookmark78)

Returns the minute for *time*, in the range 0 to 59, or NULL if *time* is NULL.

mysql> **SELECT** **MINUTE('2008-02-03** **10:05:03');**

-> 5

• [MONTH(*date*)](#_bookmark79)

Returns the month for *date*, in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part. Returns NULL if *date* is NULL.

mysql> **SELECT** **MONTH('2008-02-03');**

-> 2

• [MONTHNAME(*date*)](#_bookmark80)

Returns the full name of the month for *date*. The language used for the name is controlled by the value of the lc\_time\_names system variable (Section 10.16, “MySQL Server Locale Support”). Returns NULL if *date* is NULL.

mysql> **SELECT** **MONTHNAME('2008-02-03');**

-> 'February'

• [NOW([*fsp*])](#_bookmark81)

Returns the current date and time as a value in '*YYYY-MM-DD* *hh:mm:ss*' or *YYYYMMDDhhmmss* format, depending on whether the function is used in string or numeric context. The value is expressed in the session time zone.

If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

mysql> **SELECT** **NOW();**

-> '2007-12-15 23:50:26'

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

-> 20071215235026.000000

[NOW()](#_bookmark81) returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, [NOW()](#_bookmark81) returns the time at which the function or triggering statement began to execute.) This differs from the behavior for [SYSDATE()](#_bookmark90), which returns the exact time at which it executes.

mysql> **SELECT** **NOW(),** **SLEEP(2),** **NOW();**

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

| NOW() | SLEEP(2) | NOW() |

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

| 2006-04-12 13:47:36 | 0 | 2006-04-12 13:47:36 |

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

mysql> **SELECT** **SYSDATE(),** **SLEEP(2),** **SYSDATE();**

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

| SYSDATE() | SLEEP(2) | SYSDATE() |

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

| 2006-04-12 13:47:44 | 0 | 2006-04-12 13:47:46 |



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

In addition, the SET TIMESTAMP statement affects the value returned by [NOW()](#_bookmark81) but not by [SYSDATE()](#_bookmark90). This means that timestamp settings in the binary log have no effect on invocations of [SYSDATE()](#_bookmark90). Setting the timestamp to a nonzero value causes each subsequent invocation of [NOW()](#_bookmark81) to return that value. Setting the timestamp to zero cancels this effect so that [NOW()](#_bookmark81) once again returns the current date and time.

See the description for [SYSDATE()](#_bookmark90) for additional information about the differences between the two functions.

• [PERIOD\_ADD(*P*,*N*)](#_bookmark82)

Adds *N* months to period *P* (in the format *YYMM*or *YYYYMM*). Returns a value in the format *YYYYMM*.

**Note**

The period argument *P* is *not* a date value.

This function returns NULL if *P* or *N* is NULL.

mysql> **SELECT** **PERIOD\_ADD(200801,2);**

-> 200803

• [PERIOD\_DIFF(*P1*,*P2*)](#_bookmark83)

Returns the number of months between periods *P1* and *P2*. *P1* and *P2* should be in the format *YYMM* or *YYYYMM*. Note that the period arguments *P1* and *P2* are *not* date values.

This function returns NULL if *P1* or *P2* is NULL.

mysql> **SELECT** **PERIOD\_DIFF(200802,200703);**

-> 11

• [QUARTER(*date*)](#_bookmark84)

Returns the quarter of the year for *date*, in the range 1 to 4, or NULL if *date* is NULL.

mysql> **SELECT** **QUARTER('2008-04-01');**

-> 2

• [SECOND(*time*)](#_bookmark86)

Returns the second for *time*, in the range 0 to 59, or NULL if *time* is NULL.

mysql> **SELECT** **SECOND('10:05:03');**

-> 3

• [SEC\_TO\_TIME(*seconds*)](#_bookmark85)

Returns the *seconds* argument, converted to hours, minutes, and seconds, as a TIME value. The range of the result is constrained to that of the TIME data type. A warning occurs if the argument corresponds to a value outside that range.

The function returns NULL if *seconds* is NULL.

mysql> **SELECT** **SEC\_TO\_TIME(2378);**

-> '00:39:38'

mysql> **SELECT** **SEC\_TO\_TIME(2378)** **+** **0;**

-> 3938

• [STR\_TO\_DATE(*str*,*format*)](#_bookmark87)

This is the inverse of the [DATE\_FORMAT()](#_bookmark59) function. It takes a string *str* and a format string *format*. STR\_TO\_DATE() returns a DATETIME value if the format string contains both date and

time parts, or a DATE or TIME value if the string contains only date or time parts. If *str* or *format* is NULL, the function returns NULL. If the date, time, or datetime value extracted from *str* cannot be parsed according to the rules followed by the server, STR\_TO\_DATE() returns NULL and produces a warning.

The server scans *str* attempting to match *format* to it. The format string can contain literal characters and format specifiers beginning with %. Literal characters in *format* must match literally in *str*. Format specifiers in *format* must match a date or time part in *str*. For the specifiers that can be used in *format*, see the [DATE\_FORMAT()](#_bookmark59) function description.

mysql> **SELECT** **STR\_TO\_DATE('01,5,2013','%d,%m,%Y');**

-> '2013-05-01'

mysql> **SELECT** **STR\_TO\_DATE('May** **1,** **2013','%M** **%d,%Y');**

-> '2013-05-01'

Scanning starts at the beginning of *str* and fails if *format* is found not to match. Extra characters at the end of *str* are ignored.

mysql> **SELECT** **STR\_TO\_DATE('a09:30:17','a%h:%i:%s');**

-> '09:30:17'

mysql> **SELECT** **STR\_TO\_DATE('a09:30:17','%h:%i:%s');**

-> NULL

mysql> **SELECT** **STR\_TO\_DATE('09:30:17a','%h:%i:%s');**

-> '09:30:17'

Unspecified date or time parts have a value of 0, so incompletely specified values in *str* produce a result with some or all parts set to 0:

mysql> **SELECT** **STR\_TO\_DATE('abc','abc');**

-> '0000-00-00'

mysql> **SELECT** **STR\_TO\_DATE('9','%m');**

-> '0000-09-00'

mysql> **SELECT** **STR\_TO\_DATE('9','%s');**

-> '00:00:09'

Range checking on the parts of date values is as described in Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types” . This means, for example, that “zero” dates or dates with part values of 0 are permitted unless the SQL mode is set to disallow such values.

mysql> **SELECT** **STR\_TO\_DATE('00/00/0000',** **'%m/%d/%Y');**

-> '0000-00-00'

mysql> **SELECT** **STR\_TO\_DATE('04/31/2004',** **'%m/%d/%Y');**

-> '2004-04-31'

If the NO\_ZERO\_DATE SQL mode is enabled, zero dates are disallowed. In that case, [STR\_TO\_DATE()](#_bookmark87) returns NULL and generates a warning:

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

mysql> **SELECT** **STR\_TO\_DATE('00/00/0000',** **'%m/%d/%Y');**

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

| STR\_TO\_DATE('00/00/0000', '%m/%d/%Y') |

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

| 0000-00-00 |

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

mysql> **SET** **sql\_mode** **=** **'NO\_ZERO\_DATE';**

mysql> **SELECT** **STR\_TO\_DATE('00/00/0000',** **'%m/%d/%Y');**

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

| STR\_TO\_DATE('00/00/0000', '%m/%d/%Y') |

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

| NULL |

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

mysql> **SHOW** **WARNINGS\G**

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

Level: Warning

Code: 1411



Message: Incorrect datetime value: '00/00/0000' for function str\_to\_date

**Note**

You cannot use format "%X%V" to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:

mysql> **SELECT** **STR\_TO\_DATE('200442** **Monday',** **'%X%V** **%W');**

-> '2004-10-18'

You should also be aware that, for dates and the date portions of datetime values, STR\_TO\_DATE() checks (only) the individual year, month, and day of month values for validity. More precisely, this means that the year is checked to be sure that it is in the range 0-9999 inclusive, the month is checked to ensure that it is in the range 1-12 inclusive, and the day of month is checked to make sure that it is in the range 1-31 inclusive, but the server does not check the values in combination. For example, SELECT STR\_TO\_DATE('23-2-31', '%Y-%m-%d') returns 2023-02-31. Enabling or disabling the ALLOW\_INVALID\_DATES server SQL mode has no effect on this behavior. See Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types” , for more information.

• [SUBDATE(*date*,INTERVAL *expr* *unit*)](#_bookmark88), [SUBDATE(*expr*,*days*)](#_bookmark88)

When invoked with the INTERVAL form of the second argument, [SUBDATE()](#_bookmark88) is a synonym for [DATE\_SUB()](#_bookmark60). For information on the INTERVAL *unit* argument, see the discussion for [DATE\_ADD()](#_bookmark43).

mysql> **SELECT** **DATE\_SUB('2008-01-02',** **INTERVAL** **31** **DAY);**

-> '2007-12-02'

mysql> **SELECT** **SUBDATE('2008-01-02',** **INTERVAL** **31** **DAY);**

-> '2007-12-02'

The second form enables the use of an integer value for *days*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *expr*.

mysql> **SELECT** **SUBDATE('2008-01-02** **12:00:00',** **31);**

-> '2007-12-02 12:00:00'

This function returns NULL if any of its arguments are NULL.

• [SUBTIME(*expr1*,*expr2*)](#_bookmark89)

[SUBTIME()](#_bookmark89) returns *expr1* − *expr2* expressed as a value in the same format as *expr1*. *expr1* is a time or datetime expression, and *expr2* is a time expression.

Resolution of this function's return type is performed as it is for the [ADDTIME()](#_bookmark51) function; see the description of that function for more information.

mysql> **SELECT** **SUBTIME('2007-12-31** **23:59:59** **.999999','1** **1:1:1** **.000002');**

-> '2007-12-30 22:58:58 .999997'

mysql> **SELECT** **SUBTIME('01:00:00** **.999999',** **'02:00:00** **.999998');**

-> '-00:59:59.999999'

This function returns NULL if *expr1* or *expr2* is NULL.

• [SYSDATE([*fsp*])](#_bookmark90)

Returns the current date and time as a value in '*YYYY-MM-DD* *hh:mm:ss*' or *YYYYMMDDhhmmss* format, depending on whether the function is used in string or numeric context.

If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

[SYSDATE()](#_bookmark90) returns the time at which it executes. This differs from the behavior for [NOW()](#_bookmark81), which returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, [NOW()](#_bookmark81) returns the time at which the function or triggering statement began to execute.)

mysql> **SELECT** **NOW(),** **SLEEP(2),** **NOW();**

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

| NOW() | SLEEP(2) | NOW() |

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

| 2006-04-12 13:47:36 | 0 | 2006-04-12 13:47:36 |

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

mysql> **SELECT** **SYSDATE(),** **SLEEP(2),** **SYSDATE();**

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

| SYSDATE() | SLEEP(2) | SYSDATE() |

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

| 2006-04-12 13:47:44 | 0 | 2006-04-12 13:47:46 |

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

In addition, the SET TIMESTAMP statement affects the value returned by [NOW()](#_bookmark81) but not by [SYSDATE()](#_bookmark90). This means that timestamp settings in the binary log have no effect on invocations of [SYSDATE()](#_bookmark90).

Because [SYSDATE()](#_bookmark90) can return different values even within the same statement, and is not affected by SET TIMESTAMP, it is nondeterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging.

Alternatively, you can use the --sysdate-is-now option to cause [SYSDATE()](#_bookmark90) to be an alias for [NOW()](#_bookmark81). This works if the option is used on both the replication source server and the replica.

The nondeterministic nature of [SYSDATE()](#_bookmark90) also means that indexes cannot be used for evaluating expressions that refer to it.

• [TIME(*expr*](#_bookmark91))

Extracts the time part of the time or datetime expression *expr* and returns it as a string. Returns NULL if *expr* is NULL.

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

|  |  |
| --- | --- |
| mysql> **SELECT** **TIME('2003-12-31**  -> '01:02:03'  mysql> **SELECT** **TIME('2003-12-31**  -> '01:02:03.000123' | **01:02:03');**  **01:02:03** **.000123');** |

> **TIMESTAMPADD(MONTH,** **1,** **DATE** **'2024-03-31')** **AS** **t2;**

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

| t1 | t2 |

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

| 2024-04-30 | 2024-04-30 |

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

1 row in set (0.00 sec)

• [TIMEDIFF(*expr1*,*expr2*)](#_bookmark94)

[TIMEDIFF()](#_bookmark94) returns *expr1* − *expr2* expressed as a time value. *expr1* and *expr2* are strings which are converted to TIME or DATETIME expressions; these must be of the same type following conversion. Returns NULL if *expr1* or *expr2* is NULL.

The result returned by TIMEDIFF() is limited to the range allowed for TIME values. Alternatively, you can use either of the functions [TIMESTAMPDIFF()](#_bookmark97) and [UNIX TIMESTAMP()](#_bookmark100)\_, both of which return integers.

mysql> **SELECT** **TIMEDIFF('2000-01-01** **00:00:00',**

-> **'2000-01-01** **00:00:00** **.000001');**

-> '-00:00:00 .000001'

mysql> **SELECT** **TIMEDIFF('2008-12-31** **23:59:59** **.000001',**

-> **'2008-12-30** **01:01:01** **.000002');**

-> '46:58:57.999999'

• [TIMESTAMP(*expr*](#_bookmark95)), [TIMESTAMP(*expr1*,](#_bookmark95)*expr2*)

With a single argument, this function returns the date or datetime expression *expr* as a datetime value. With two arguments, it adds the time expression *expr2* to the date or datetime expression *expr1* and returns the result as a datetime value. Returns NULL if *expr*, *expr1*, or *expr2* is NULL.

mysql> **SELECT** **TIMESTAMP('2003-12-31');**

-> '2003-12-31 00:00:00'

mysql> **SELECT** **TIMESTAMP('2003-12-31** **12:00:00','12:00:00');**

-> '2004-01-01 00:00:00'

• [TIMESTAMPADD(*unit*,*interval*,*datetime\_expr*)](#_bookmark96)

Adds the integer expression *interval* to the date or datetime expression *datetime\_expr*. The unit for *interval* is given by the *unit* argument, which should be one of the following values: MICROSECOND (microseconds), SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

The *unit* value may be specified using one of keywords as shown, or with a prefix of SQL\_TSI\_. For example, DAY and SQL\_TSI\_DAY both are legal.

This function returns NULL if *interval* or *datetime\_expr* is NULL.

mysql> **SELECT** **TIMESTAMPADD(MINUTE,** **1,** **'2003-01-02');**

-> '2003-01-02 00:01:00'

mysql> **SELECT** **TIMESTAMPADD(WEEK,1,'2003-01-02');**

-> '2003-01-09'

When adding a MONTH interval to a DATE or DATETIME value, and the resulting date includes a day that does not exist in the given month, the day is adjusted to the last day of the month, as shown here:

mysql> **SELECT** **TIMESTAMPADD(MONTH,** **1,** **DATE** **'2024-03-30')** **AS** **t1,**

• [TIMESTAMPDIFF(*unit*,*datetime\_expr1*,*datetime\_expr2*)](#_bookmark97)

Returns *datetime\_expr2* − *datetime\_expr1*, where *datetime\_expr1* and *datetime\_expr2* are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part '00:00:00' where necessary. The unit for the

result (an integer) is given by the *unit* argument. The legal values for *unit* are the same as those listed in the description of the [TIMESTAMPADD()](#_bookmark96) function.

This function returns NULL if *datetime\_expr1* or *datetime\_expr2* is NULL.

mysql> **SELECT** **TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');**

-> 3

mysql> **SELECT** **TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');**

-> -1

mysql> **SELECT** **TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01** **12:05:55');**

-> 128885

**Note**

The order of the date or datetime arguments for this function is the opposite of that used with the [TIMESTAMP()](#_bookmark95) function when invoked with 2 arguments.

• [TIME\_FORMAT(*time*,*format*)](#_bookmark92)

This is used like the [DATE\_FORMAT()](#_bookmark59) function, but the *format* string may contain format specifiers only for hours, minutes, seconds, and microseconds. Other specifiers produce a NULL or 0. TIME\_FORMAT() returns NULL if *time* or *format* is NULL.

If the *time* value contains an hour part that is greater than 23, the %H and %k hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

mysql> **SELECT** **TIME\_FORMAT('100:00:00',** **'%H** **%k** **%h** **%I** **%l');**

-> '100 100 04 04 4'

• [TIME\_TO\_SEC(*time*)](#_bookmark93)

Returns the *time* argument, converted to seconds. Returns NULL if *time* is NULL.

mysql> **SELECT** **TIME\_TO\_SEC('22:23:00');**

-> 80580

mysql> **SELECT** **TIME\_TO\_SEC('00:39:38');**

-> 2378

• [TO\_DAYS(*date*)](#_bookmark98)

Given a date *date*, returns a day number (the number of days since year 0). Returns NULL if *date* is NULL.

mysql> **SELECT** **TO\_DAYS(950501);**

-> 728779

mysql> **SELECT** **TO\_DAYS('2007-10-07');**

-> 733321

[TO\_DAYS()](#_bookmark98) is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See [Section 12.9, “What Calendar Is Used By MySQL?”](#_bookmark110) , for details.

Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in Section 11.2, “Date and Time Data Types” . For example, '2008-10-07' and '08-10-07' are seen as identical dates:

mysql> **SELECT** **TO\_DAYS('2008-10-07'),** **TO\_DAYS('08-10-07');**



-> 733687, 733687

In MySQL, the zero date is defined as '0000-00-00', even though this date is itself considered invalid. This means that, for '0000-00-00' and '0000-01-01', [TO\_DAYS()](#_bookmark98) returns the values shown here:

mysql> **SELECT** **TO\_DAYS('0000-00-00');**

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

| to\_days('0000-00-00') |

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

| NULL |

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

1 row in set, 1 warning (0.00 sec)

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1292 | Incorrect datetime value: '0000-00-00' |

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

1 row in set (0.00 sec)

mysql> **SELECT** **TO\_DAYS('0000-01-01');**

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

| to\_days('0000-01-01') |

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

| 1 |

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

1 row in set (0.00 sec)

This is true whether or not the ALLOW\_INVALID\_DATES SQL server mode is enabled.

• [TO\_SECONDS(*expr*)](#_bookmark99)

Given a date or datetime *expr*, returns the number of seconds since the year 0. If *expr* is not a valid date or datetime value (including NULL), it returns NULL.

mysql> **SELECT** **TO\_SECONDS(950501);**

-> 62966505600

mysql> **SELECT** **TO\_SECONDS('2009-11-29');**

-> 63426672000

mysql> **SELECT** **TO\_SECONDS('2009-11-29** **13:43:32');**

-> 63426721412

mysql> **SELECT** **TO\_SECONDS(** **NOW()** **);**

-> 63426721458

Like [TO\_DAYS()](#_bookmark98), TO\_SECONDS() is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See [Section 12.9, “What Calendar Is Used By MySQL?”](#_bookmark110) , for details.

Like [TO\_DAYS()](#_bookmark98), TO\_SECONDS(), converts two-digit year values in dates to four-digit form using the rules in Section 11.2, “Date and Time Data Types” .

In MySQL, the zero date is defined as '0000-00-00', even though this date is itself considered invalid. This means that, for '0000-00-00' and '0000-01-01', [TO\_SECONDS()](#_bookmark99) returns the values shown here:

mysql> **SELECT** **TO\_SECONDS('0000-00-00');**

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

| TO\_SECONDS('0000-00-00') |

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

| NULL |

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

1 row in set, 1 warning (0.00 sec)

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1292 | Incorrect datetime value: '0000-00-00' |

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

1 row in set (0.00 sec)

mysql> **SELECT** **TO\_SECONDS('0000-01-01');**

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

| TO\_SECONDS('0000-01-01') |

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

| 86400 |

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

1 row in set (0.00 sec)

This is true whether or not the ALLOW\_INVALID\_DATES SQL server mode is enabled.

• [UNIX\_TIMESTAMP([*date*])](#_bookmark100)

If [UNIX\_TIMESTAMP()](#_bookmark100) is called with no *date* argument, it returns a Unix timestamp representing seconds since '1970-01-01 00:00:00' UTC.

If [UNIX\_TIMESTAMP()](#_bookmark100) is called with a *date* argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. The server interprets *date* as a value in the session time zone and converts it to an internal Unix timestamp value in UTC. (Clients can set the session time zone as described in Section 5.1.15, “MySQL Server Time Zone Support” .) The *date* argument may be a DATE, DATETIME, or TIMESTAMP string, or a number in *YYMMDD*, *YYMMDDhhmmss*, *YYYYMMDD*, or *YYYYMMDDhhmmss* format. If the argument includes a time part, it may optionally include a fractional seconds part.

The return value is an integer if no argument is given or the argument does not include a fractional seconds part, or DECIMAL if an argument is given that includes a fractional seconds part.

When the *date* argument is a TIMESTAMP column, [UNIX\_TIMESTAMP()](#_bookmark100) returns the internal timestamp value directly, with no implicit “string-to-Unix-timestamp” conversion.

Prior to MySQL 8.0.28, the valid range of argument values is the same as for the TIMESTAMP data type: '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.999999' UTC. This is also the case in MySQL 8.0.28 and later for 32-bit platforms. For MySQL 8.0.28 and later running on 64-bit platforms, the valid range of argument values for UNIX\_TIMESTAMP() is '1970-01-01 00:00:01.000000' UTC to '3001-01-19 03:14:07.999999' UTC (corresponding to 32536771199.999999 seconds).

Regardless of MySQL version or platform architecture, if you pass an out-of-range date to [UNIX\_TIMESTAMP()](#_bookmark100), it returns 0. If *date* is NULL, it returns NULL.

|  |  |
| --- | --- |
| mysql> **SELECT** **UNIX\_TIMESTAMP();**  -> 1447431666  mysql> **SELECT** **UNIX\_TIMESTAMP('2015-11-13**  -> 1447431619  mysql> **SELECT** **UNIX\_TIMESTAMP('2015-11-13**  -> 1447431619.012 | **10:20:19');**  **10:20:19** **.012');** |

If you use [UNIX\_TIMESTAMP()](#_bookmark100) and [FROM\_UNIXTIME()](#_bookmark69) to convert between values in a non-UTC time zone and Unix timestamp values, the conversion is lossy because the mapping is not one-to- one in both directions. For example, due to conventions for local time zone changes such as Daylight Saving Time (DST), it is possible for [UNIX\_TIMESTAMP()](#_bookmark100) to map two values that are distinct in a non-UTC time zone to the same Unix timestamp value. [FROM\_UNIXTIME()](#_bookmark69) maps that value back to only one of the original values. Here is an example, using values that are distinct in the MET time zone:

mysql> **SET** **time\_zone** **=** **'MET';**



mysql> **SELECT** **UNIX\_TIMESTAMP('2005-03-27** **03:00:00');**

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

| UNIX\_TIMESTAMP('2005-03-27 03:00:00') |

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

| 1111885200 |

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

mysql> **SELECT** **UNIX\_TIMESTAMP('2005-03-27** **02:00:00');**

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

| UNIX\_TIMESTAMP('2005-03-27 02:00:00') |

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

| 1111885200 |

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

mysql> **SELECT** **FROM\_UNIXTIME(1111885200);**

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

| FROM\_UNIXTIME(1111885200) |

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

| 2005-03-27 03:00:00 |

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

**Note**

To use named time zones such as 'MET' or 'Europe/Amsterdam' the time zone tables must be properly set up. For instructions, see Section 5.1.15, “MySQL Server Time Zone Support” .

,

If you want to subtract [UNIX\_TIMESTAMP()](#_bookmark100) columns, you might want to cast them to signed integers. See [Section 12.11, “Cast Functions and Operators”](#_bookmark42) .

• [UTC\_DATE](#_bookmark101), [UTC\_DATE()](#_bookmark101)

Returns the current UTC date as a value in '*YYYY-MM-DD*' or *YYYYMMDD* format, depending on whether the function is used in string or numeric context.

mysql> **SELECT** **UTC\_DATE(),** **UTC\_DATE()** **+** **0;**

-> '2003-08-14', 20030814

• [UTC\_TIME](#_bookmark102), [UTC\_TIME([*fsp*])](#_bookmark102)

Returns the current UTC time as a value in *'hh:mm:ss'* or *hhmmss* format, depending on whether the function is used in string or numeric context.

If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

mysql> **SELECT** **UTC\_TIME(),** **UTC\_TIME()** **+** **0;**

-> '18:07:53', 180753.000000

• [UTC\_TIMESTAMP](#_bookmark103), [UTC\_TIMESTAMP([*fsp*])](#_bookmark103)

Returns the current UTC date and time as a value in '*YYYY-MM-DD* *hh:mm:ss*' or *YYYYMMDDhhmmss* format, depending on whether the function is used in string or numeric context.

If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

mysql> **SELECT** **UTC\_TIMESTAMP(),** **UTC\_TIMESTAMP()** **+** **0;**

-> '2003-08-14 18:08:04', 20030814180804.000000

• [WEEK(*date*[,*mode*])](#_bookmark104)

This function returns the week number for *date*. The two-argument form of [WEEK()](#_bookmark104) enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the *mode* argument is omitted, the value of the

default\_week\_format system variable is used. See Section 5. 1.8, “Server System Variables” .

For a NULL date value, the function returns NULL.

The following table describes how the *mode* argument works.

|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **First** **day** **of** **week** | **Range** | **Week** **1** **is** **the** **first**  **week** **…** |
| 0 | Sunday | 0-53 | with a Sunday in this year |
| 1 | Monday | 0-53 | with 4 or more days this year |
| 2 | Sunday | 1-53 | with a Sunday in this year |
| 3 | Monday | 1-53 | with 4 or more days this year |
| 4 | Sunday | 0-53 | with 4 or more days this year |
| 5 | Monday | 0-53 | with a Monday in this year |
| 6 | Sunday | 1-53 | with 4 or more days this year |
| 7 | Monday | 1-53 | with a Monday in this year |

For *mode* values with a meaning of “with 4 or more days this year,” weeks are numbered according to ISO 8601:1988:

• If the week containing January 1 has 4 or more days in the new year, it is week 1.

• Otherwise, it is the last week of the previous year, and the next week is week 1.

|  |  |
| --- | --- |
| mysql> **SELECT**  -> 7  mysql> **SELECT**  -> 7  mysql> **SELECT**  -> 8  mysql> **SELECT**  -> 53 | **WEEK('2008-02-20');** **WEEK('2008-02-20',0);** **WEEK('2008-02-20',1);**  **WEEK('2008-12-31',1);** |

If a date falls in the last week of the previous year, MySQL returns 0 if you do not use 2, 3, 6, or 7 as the optional *mode* argument:

mysql> **SELECT** **YEAR('2000-01-01'),** **WEEK('2000-01-01',0);**

-> 2000, 0

One might argue that [WEEK()](#_bookmark104) should return 52 because the given date actually occurs in the 52nd week of 1999. [WEEK()](#_bookmark104) returns 0 instead so that the return value is “the week number in the given year.” This makes use of the [WEEK()](#_bookmark104) function reliable when combined with other functions that extract a date part from a date.

If you prefer a result evaluated with respect to the year that contains the first day of the week for the given date, use 0, 2, 5, or 7 as the optional *mode* argument.

mysql> **SELECT** **WEEK('2000-01-01',2);**

-> 52

Alternatively, use the [YEARWEEK()](#_bookmark108) function:

mysql> **SELECT** **YEARWEEK('2000-01-01');**

-> 199952

mysql> **SELECT** **MID(YEARWEEK('2000-01-01'),5,2);**

-> '52'

• [WEEKDAY(*date*)](#_bookmark105)

Returns the weekday index for *date* ( 0 = Monday, 1 = Tuesday, … 6 = Sunday). Returns NULL if *date* is NULL.

mysql> **SELECT** **WEEKDAY('2008-02-03** **22:23:00');**

-> 6

mysql> **SELECT** **WEEKDAY('2007-11-06');**

-> 1

• [WEEKOFYEAR(*date*)](#_bookmark106)

Returns the calendar week of the date as a number in the range from 1 to 53. Returns NULL if *date* is NULL.

[WEEKOFYEAR()](#_bookmark106) is a compatibility function that is equivalent to [WEEK( ,3)](#_bookmark104)*date*.

mysql> **SELECT** **WEEKOFYEAR('2008-02-20');**

-> 8

• [YEAR(*date*)](#_bookmark107)

Returns the year for *date*, in the range 1000 to 9999, or 0 for the “zero” date. Returns NULL if *date* is NULL.

mysql> **SELECT** **YEAR('1987-01-01');**

-> 1987

• [YEARWEEK(*date*)](#_bookmark108), [YEARWEEK(*date*,*mode*)](#_bookmark108)

Returns year and week for a date. The year in the result may be different from the year in the date argument for the first and the last week of the year. Returns NULL if *date* is NULL.

The *mode* argument works exactly like the *mode* argument to [WEEK()](#_bookmark104). For the single-argument syntax, a *mode* value of 0 is used. Unlike [WEEK()](#_bookmark104), the value of default\_week\_format does not influence [YEARWEEK()](#_bookmark108).

mysql> **SELECT** **YEARWEEK('1987-01-01');**

-> 198652

The week number is different from what the [WEEK()](#_bookmark104) function would return ( 0) for optional arguments 0 or 1, as [WEEK()](#_bookmark104) then returns the week in the context of the given year.

**12.8** **String** **Functions** **and** **Operators**

**Table** **12.12** **String** **Functions** **and** **Operators**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [ASCII()](#_bookmark111) | Return numeric value of left-most character |
| [BIN()](#_bookmark112) | Return a string containing binary representation of a number |
| [BIT\_LENGTH()](#_bookmark113) | Return length of argument in bits |
| [CHAR()](#_bookmark114) | Return the character for each integer passed |
| [CHAR\_LENGTH()](#_bookmark115) | Return number of characters in argument |

|  |  |
| --- | --- |
| **Name** | **Description** |
| [CHARACTER\_LENGTH()](#_bookmark116) | Synonym for CHAR\_LENGTH() |
| [CONCAT()](#_bookmark117) | Return concatenated string |
| [CONCAT\_WS()](#_bookmark118) | Return concatenate with separator |
| [ELT()](#_bookmark119) | Return string at index number |
| [EXPORT\_SET()](#_bookmark120) | Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string |
| [FIELD()](#_bookmark121) | Index (position) of first argument in subsequent arguments |
| [FIND\_IN\_SET()](#_bookmark122) | Index (position) of first argument within second argument |
| [FORMAT()](#_bookmark46) | Return a number formatted to specified number of decimal places |
| [FROM\_BASE64()](#_bookmark123) | Decode base64 encoded string and return result |
| [HEX()](#_bookmark48) | Hexadecimal representation of decimal or string value |
| [INSERT()](#_bookmark124) | Insert substring at specified position up to specified number of characters |
| [INSTR()](#_bookmark125) | Return the index of the first occurrence of substring |
| [LCASE()](#_bookmark126) | Synonym for LOWER() |
| [LEFT()](#_bookmark127) | Return the leftmost number of characters as specified |
| [LENGTH()](#_bookmark128) | Return the length of a string in bytes |
| [LIKE](#_bookmark129) | Simple pattern matching |
| [LOAD\_FILE()](#_bookmark130) | Load the named file |
| [LOCATE()](#_bookmark131) | Return the position of the first occurrence of substring |
| [LOWER()](#_bookmark132) | Return the argument in lowercase |
| [LPAD()](#_bookmark133) | Return the string argument, left-padded with the specified string |
| [LTRIM()](#_bookmark134) | Remove leading spaces |
| [MAKE\_SET()](#_bookmark135) | Return a set of comma-separated strings that have the corresponding bit in bits set |
| [MATCH()](#_bookmark136) | Perform full-text search |
| [MID()](#_bookmark137) | Return a substring starting from the specified position |
| [NOT LIKE](#_bookmark138) | Negation of simple pattern matching |
| [NOT REGEXP](#_bookmark139) | Negation of REGEXP |
| [OCT()](#_bookmark140) | Return a string containing octal representation of a number |
| [OCTET\_LENGTH()](#_bookmark141) | Synonym for LENGTH() |
| [ORD()](#_bookmark142) | Return character code for leftmost character of the argument |
| [POSITION()](#_bookmark143) | Synonym for LOCATE() |

|  |  |
| --- | --- |
| **Name** | **Description** |
| [QUOTE()](#_bookmark144) | Escape the argument for use in an SQL statement |
| [REGEXP](#_bookmark145) | Whether string matches regular expression |
| [REGEXP\_INSTR()](#_bookmark146) | Starting index of substring matching regular expression |
| [REGEXP\_LIKE()](#_bookmark147) | Whether string matches regular expression |
| [REGEXP\_REPLACE()](#_bookmark148) | Replace substrings matching regular expression |
| [REGEXP\_SUBSTR()](#_bookmark149) | Return substring matching regular expression |
| [REPEAT()](#_bookmark150) | Repeat a string the specified number of times |
| [REPLACE()](#_bookmark151) | Replace occurrences of a specified string |
| [REVERSE()](#_bookmark152) | Reverse the characters in a string |
| [RIGHT()](#_bookmark153) | Return the specified rightmost number of characters |
| [RLIKE](#_bookmark145) | Whether string matches regular expression |
| [RPAD()](#_bookmark154) | Append string the specified number of times |
| [RTRIM()](#_bookmark155) | Remove trailing spaces |
| [SOUNDEX()](#_bookmark156) | Return a soundex string |
| [SOUNDS LIKE](#_bookmark157) | Compare sounds |
| [SPACE()](#_bookmark158) | Return a string of the specified number of spaces |
| [STRCMP()](#_bookmark159) | Compare two strings |
| [SUBSTR()](#_bookmark160) | Return the substring as specified |
| [SUBSTRING()](#_bookmark161) | Return the substring as specified |
| [SUBSTRING\_INDEX()](#_bookmark162) | Return a substring from a string before the specified number of occurrences of the delimiter |
| [TO\_BASE64()](#_bookmark163) | Return the argument converted to a base-64 string |
| [TRIM()](#_bookmark164) | Remove leading and trailing spaces |
| [UCASE()](#_bookmark165) | Synonym for UPPER() |
| [UNHEX()](#_bookmark166) | Return a string containing hex representation of a number |
| [UPPER()](#_bookmark167) | Convert to uppercase |
| [WEIGHT\_STRING()](#_bookmark168) | Return the weight string for a string |

String-valued functions return NULL if the length of the result would be greater than the value of the

max\_allowed\_packet system variable. See Section 5. 1. 1, “Configuring the Server” . For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

• [ASCII(*str*)](#_bookmark111)

Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns NULL if *str* is NULL. [ASCII()](#_bookmark111) works for 8-bit characters.

mysql> **SELECT** **ASCII('2');**

-> 50

mysql> **SELECT** **ASCII(2);**

-> 50

mysql> **SELECT** **ASCII('dx');**

-> 100

See also the [ORD()](#_bookmark142) function.

• [BIN(*N*)](#_bookmark112)

Returns a string representation of the binary value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to [CONV(*N*,10,2)](#_bookmark17). Returns NULL if *N* is NULL.

mysql> **SELECT** **BIN(12);**

-> '1100'

• [BIT\_LENGTH(*str*)](#_bookmark113)

Returns the length of the string *str* in bits. Returns NULL if *str* is NULL.

mysql> **SELECT** **BIT\_LENGTH('text');**

-> 32

• [CHAR(*N*,... [USING *charset\_name*])](#_bookmark114)

[CHAR()](#_bookmark114) interprets each argument *N*as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.

mysql> **SELECT** **CHAR(77,121,83,81,'76');**

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

| CHAR(77,121,83,81,'76') |

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

| 0x4D7953514C |

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

1 row in set (0.00 sec)

mysql> **SELECT** **CHAR(77,77** **.3,'77** **.3');**

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

| CHAR(77,77 .3,'77 .3') |

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

| 0x4D4D4D |

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

1 row in set (0.00 sec)

By default, [CHAR()](#_bookmark114) returns a binary string. To produce a string in a given character set, use the optional USING clause:

mysql> **SELECT** **CHAR(77,121,83,81,'76'** **USING** **utf8mb4);**

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

| CHAR(77,121,83,81,'76' USING utf8mb4) |

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

| MySQL |

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

1 row in set (0.00 sec)

mysql> **SELECT** **CHAR(77,77** **.3,'77** **.3'** **USING** **utf8mb4);**

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

| CHAR(77,77 .3,'77 .3' USING utf8mb4) |

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

| MMM |

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

1 row in set, 1 warning (0.00 sec)

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1292 | Truncated incorrect INTEGER value: '77 .3' |

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

1 row in set (0 .00 sec)

If USING is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from [CHAR()](#_bookmark114) becomes NULL.

If [CHAR()](#_bookmark114) is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

[CHAR()](#_bookmark114) arguments larger than 255 are converted into multiple result bytes. For example, [CHAR(256)](#_bookmark114) is equivalent to [CHAR(1,0)](#_bookmark114), and [CHAR(256\*256)](#_bookmark114) is equivalent to [CHAR(1,0,0)](#_bookmark114):

mysql> **SELECT** **HEX(CHAR(1,0)),** **HEX(CHAR(256));**

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

| HEX(CHAR(1,0)) | HEX(CHAR(256)) |

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

| 0100 | 0100 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **HEX(CHAR(1,0,0)),** **HEX(CHAR(256\*256));**

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

| HEX(CHAR(1,0,0)) | HEX(CHAR(256\*256)) |

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

| 010000 | 010000 |

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

1 row in set (0.00 sec)

• [CHAR\_LENGTH(*str*)](#_bookmark115)

Returns the length of the string *str*, measured in code points. A multibyte character counts as a single code point. This means that, for a string containing two 3-byte characters, [LENGTH()](#_bookmark128) returns 6, whereas [CHAR\_LENGTH()](#_bookmark115) returns 2, as shown here:

mysql> **SET** **@dolphin:='**海豚**';**

Query OK, 0 rows affected (0.01 sec)

mysql> **SELECT** **LENGTH(@dolphin),** **CHAR\_LENGTH(@dolphin);**

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

| LENGTH(@dolphin) | CHAR\_LENGTH(@dolphin) |

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

| 6 | 2 |

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

1 row in set (0.00 sec)

CHAR\_LENGTH() returns NULL if *str* is NULL.

• [CHARACTER\_LENGTH(*str*)](#_bookmark116)

[CHARACTER\_LENGTH()](#_bookmark116) is a synonym for [CHAR\_LENGTH()](#_bookmark115).

• [CONCAT(*str1*,*str2*,...)](#_bookmark117)

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form.

[CONCAT()](#_bookmark117) returns NULL if any argument is NULL.

mysql> **SELECT** **CONCAT('My',** **'S',** **'QL');**

-> 'MySQL'

mysql> **SELECT** **CONCAT('My',** **NULL,** **'QL');**

-> NULL

mysql> **SELECT** **CONCAT(14** **.3);**

-> '14.3'

For quoted strings, concatenation can be performed by placing the strings next to each other:

mysql> **SELECT** **'My'** **'S'** **'QL';**

-> 'MySQL'

If [CONCAT()](#_bookmark117) is invoked from within the mysql client, binary string results display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [CONCAT\_WS(*separator*,*str1*,*str2*,...)](#_bookmark118)

[CONCAT WS()](#_bookmark118)\_ stands for Concatenate With Separator and is a special form of [CONCAT()](#_bookmark117). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is NULL, the result is NULL.

mysql> **SELECT** **CONCAT\_WS(',','First** **name','Second** **name','Last** **Name');**

-> 'First name,Second name,Last Name'

mysql> **SELECT** **CONCAT\_WS(',','First** **name',NULL,'Last** **Name');**

-> 'First name,Last Name'

[CONCAT\_WS()](#_bookmark118) does not skip empty strings. However, it does skip any NULL values after the separator argument.

• [ELT(*N*,*str1*,*str2*,*str3*,...)](#_bookmark119)

[ELT()](#_bookmark119) returns the *N*th element of the list of strings: *str1* if *N* = 1, *str2* if *N* = 2, and so on. Returns NULL if *N* is less than 1, greater than the number of arguments, or NULL. [ELT()](#_bookmark119) is the complement of [FIELD()](#_bookmark121).

mysql> **SELECT** **ELT(1,** **'Aa',** **'Bb',** **'Cc',** **'Dd');**

-> 'Aa'

mysql> **SELECT** **ELT(4,** **'Aa',** **'Bb',** **'Cc',** **'Dd');**

-> 'Dd'

• [EXPORT\_SET(*bits*,*on*,*off*[,*separator*[,*number\_of\_bits*]])](#_bookmark120)

Returns a string such that for every bit set in the value *bits*, you get an *on* string and for every bit not set in the value, you get an *off* string. Bits in *bits* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *separator* string (the default being the comma character ,). The number of bits examined is given by *number\_of\_bits*, which has a default of 64 if not specified. *number\_of\_bits* is silently clipped to 64 if larger than 64. It is treated as an unsigned integer, so a value of − 1 is effectively the same as 64.

mysql> **SELECT** **EXPORT\_SET(5,'Y','N',',',4);**

-> 'Y,N,Y,N'

mysql> **SELECT** **EXPORT\_SET(6,'1','0',',',10);**

-> '0,1,1,0,0,0,0,0,0,0'

• [FIELD(*str*,*str1*,*str2*,*str3*,...)](#_bookmark121)

Returns the index (position) of *str* in the *str1*, *str2*, *str3*, ... list. Returns 0 if *str* is not found.

If all arguments to [FIELD()](#_bookmark121) are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

If *str* is NULL, the return value is 0 because NULL fails equality comparison with any value. [FIELD()](#_bookmark121) is the complement of [ELT()](#_bookmark119).

mysql> **SELECT** **FIELD('Bb',** **'Aa',** **'Bb',** **'Cc',** **'Dd',** **'Ff');**

-> 2

mysql> **SELECT** **FIELD('Gg',** **'Aa',** **'Bb',** **'Cc',** **'Dd',** **'Ff');**



-> 0

• [FIND\_IN\_SET(*str*,*strlist*)](#_bookmark122)

Returns a value in the range of 1 to *N* if the string *str* is in the string list *strlist* consisting of *N*substrings. A string list is a string composed of substrings separated by , characters. If the first argument is a constant string and the second is a column of type SET, the [FIND\_IN\_SET()](#_bookmark122) function is optimized to use bit arithmetic. Returns 0 if *str* is not in *strlist* or if *strlist* is the empty string. Returns NULL if either argument is NULL. This function does not work properly if the first argument contains a comma ( ,) character.

mysql> **SELECT** **FIND\_IN\_SET('b','a,b,c,d');**

-> 2

• [FORMAT(*X*,*D*[,*locale*])](#_bookmark46)

Formats the number *X*to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. If *D* is 0, the result has no decimal point or fractional part. If *X* or *D* is NULL, the function returns NULL.

The optional third parameter enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the lc\_time\_names system variable (see Section 10.16, “MySQL Server Locale Support”). If the locale is NULL or not specified, the default locale is 'en\_US'.

mysql> **SELECT** **FORMAT(12332** **.123456,** **4);**

-> '12,332 .1235'

mysql> **SELECT** **FORMAT(12332** **.1,4);**

-> '12,332 .1000'

mysql> **SELECT** **FORMAT(12332** **.2,0);**

-> '12,332'

mysql> **SELECT** **FORMAT(12332** **.2,2,'de\_DE');**

-> '12.332,20'

• [FROM\_BASE64(*str*)](#_bookmark123)

Takes a string encoded with the base-64 encoded rules used by [TO\_BASE64()](#_bookmark163) and returns the decoded result as a binary string. The result is NULL if the argument is NULL or not a valid base-64 string. See the description of [TO\_BASE64()](#_bookmark163) for details about the encoding and decoding rules.

mysql> **SELECT** **TO\_BASE64('abc'),** **FROM\_BASE64(TO\_BASE64('abc'));**

-> 'JWJj', 'abc'

If [FROM\_BASE64()](#_bookmark123) is invoked from within the mysql client, binary strings display using hexadecimal notation. You can disable this behavior by setting the value of the --binary-as-hex to 0 when starting the mysql client. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [HEX(*str*)](#_bookmark48), [HEX(*N*)](#_bookmark48)

For a string argument *str*, [HEX()](#_bookmark48) returns a hexadecimal string representation of *str* where each byte of each character in *str* is converted to two hexadecimal digits. (Multibyte characters therefore become more than two digits.) The inverse of this operation is performed by the [UNHEX()](#_bookmark166) function.

For a numeric argument *N*, [HEX()](#_bookmark48) returns a hexadecimal string representation of the value of *N* treated as a longlong (BIGINT) number. This is equivalent to [CONV(*N*,10,16)](#_bookmark17). The inverse of this operation is performed by [CONV(HEX(*N*),16,10)](#_bookmark17).

For a NULL argument, this function returns NULL.

mysql> **SELECT** **X'616263',** **HEX('abc'),** **UNHEX(HEX('abc'));**

-> 'abc', 616263, 'abc'

mysql> **SELECT** **HEX(255),** **CONV(HEX(255),16,10);**

-> 'FF', 255

• [INSERT(*str*,*pos*,*len*,*newstr*)](#_bookmark124)

Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns NULL if any argument is NULL.

mysql> **SELECT** **INSERT('Quadratic',** **3,** **4,** **'What');**

-> 'QuWhattic'

mysql> **SELECT** **INSERT('Quadratic',** **-1,** **4,** **'What');**

-> 'Quadratic'

mysql> **SELECT** **INSERT('Quadratic',** **3,** **100,** **'What');**

-> 'QuWhat'

This function is multibyte safe.

• [INSTR(*str*,*substr*)](#_bookmark125)

Returns the position of the first occurrence of substring *substr* in string *str*. This is the same as the two-argument form of [LOCATE()](#_bookmark131), except that the order of the arguments is reversed.

mysql> **SELECT** **INSTR('foobarbar',** **'bar');**

-> 4

mysql> **SELECT** **INSTR('xbar',** **'foobar');**

-> 0

This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string. If either argument is NULL, this functions returns NULL.

• [LCASE(*str*)](#_bookmark126)

[LCASE()](#_bookmark126) is a synonym for [LOWER()](#_bookmark132).

LCASE() used in a view is rewritten as LOWER() when storing the view's definition. (Bug #12844279)

• [LEFT(*str*,*len*)](#_bookmark127)

Returns the leftmost *len* characters from the string *str*, or NULL if any argument is NULL.

mysql> **SELECT** **LEFT('foobarbar',** **5);**

-> 'fooba'

This function is multibyte safe.

• [LENGTH(*str*)](#_bookmark128)

Returns the length of the string *str*, measured in bytes. A multibyte character counts as multiple bytes. This means that for a string containing five 2-byte characters, [LENGTH()](#_bookmark128) returns 10, whereas [CHAR\_LENGTH()](#_bookmark115) returns 5. Returns NULL if *str* is NULL.

mysql> **SELECT** **LENGTH('text');**

-> 4

**Note**

The Length() OpenGIS spatial function is named [ST\_Length()](#_bookmark169) in MySQL.

• [LOAD\_FILE(*file\_name*)](#_bookmark130)

Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the FILE privilege. The file must be readable by the server and its size less than max\_allowed\_packet bytes. If the secure\_file\_priv system variable is set to a nonempty directory name, the file to be

**WHERE** **id=1;**

loaded must be located in that directory. (Prior to MySQL 8.0.17, the file must be readable by all, not just readable by the server.)

If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns NULL.

The character\_set\_filesystem system variable controls interpretation of file names that are given as literal strings.

mysql> **UPDATE** **t**

**SET** **blob\_col=LOAD\_FILE('/tmp/picture')**

• [LOCATE(*substr*,*str*)](#_bookmark131), [LOCATE(*substr*,*str*,*pos*)](#_bookmark131)

The first syntax returns the position of the first occurrence of substring *substr* in string *str*. The second syntax returns the position of the first occurrence of substring *substr* in string *str*, starting at position *pos*. Returns 0 if *substr* is not in *str*. Returns NULL if any argument is NULL.

mysql> **SELECT** **LOCATE('bar',** **'foobarbar');**

-> 4

mysql> **SELECT** **LOCATE('xbar',** **'foobar');**

-> 0

mysql> **SELECT** **LOCATE('bar',** **'foobarbar',** **5);**

-> 7

This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string.

• [LOWER(*str*)](#_bookmark132)

Returns the string *str* with all characters changed to lowercase according to the current character set mapping, or NULL if *str* is NULL. The default character set is utf8mb4.

mysql> **SELECT** **LOWER('QUADRATICALLY');**

-> 'quadratically'

[LOWER()](#_bookmark132) (and [UPPER()](#_bookmark167)) are ineffective when applied to binary strings (BINARY, VARBINARY, BLOB). To perform lettercase conversion of a binary string, first convert it to a nonbinary string using a character set appropriate for the data stored in the string:

mysql> **SET** **@str** **=** **BINARY** **'New** **York';**

mysql> **SELECT** **LOWER(@str),** **LOWER(CONVERT(@str** **USING** **utf8mb4));**

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

| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |

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

| New York | new york |

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

For collations of Unicode character sets, [LOWER()](#_bookmark132) and [UPPER()](#_bookmark167) work according to the Unicode Collation Algorithm (UCA) version in the collation name, if there is one, and UCA 4.0.0 if no

version is specified. For example, utf8mb4\_0900\_ai\_ci and utf8mb3\_unicode\_520\_ci work according to UCA 9.0.0 and 5.2.0, respectively, whereas utf8mb3\_unicode\_ci works according to

UCA 4.0.0. See Section 10.10.1, “Unicode Character Sets” .

This function is multibyte safe.

LCASE() used within views is rewritten as LOWER().

• [LPAD(*str*,*len*,*padstr*)](#_bookmark133)

Returns the string *str*, left-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.



2217

mysql> **SELECT** **LPAD('hi',1,'??');**

-> 'h'

Returns NULL if any of its arguments are NULL.

• [LTRIM(*str*)](#_bookmark134)

Returns the string *str* with leading space characters removed. Returns NULL if *str* is NULL.

mysql> **SELECT** **LTRIM('** **barbar');**

-> 'barbar'

This function is multibyte safe.

• [MAKE\_SET(*bits*,*str1*,*str2*,...)](#_bookmark135)

Returns a set value (a string containing substrings separated by , characters) consisting of the strings that have the corresponding bit in *bits* set. *str1* corresponds to bit 0, *str2* to bit 1, and so on. NULL values in *str1*, *str2*, ... are not appended to the result.

mysql> **SELECT** **MAKE\_SET(1,'a','b','c');**

-> 'a'

mysql> **SELECT** **MAKE\_SET(1**

**4,'hello','nice','world');**

**4,'hello','nice',NULL,'world');**

**|**

**|**

-> 'hello,world'

mysql> **SELECT** **MAKE\_SET(1**

-> 'hello'

mysql> **SELECT** **MAKE\_SET(0,'a','b','c');**

-> ''

• [MID(*str*,*pos*,*len*)](#_bookmark137)

[MID(*str*,*pos*,*len*)](#_bookmark137) is a synonym for [SUBSTRING(*str*,*pos*,*len*)](#_bookmark161).

• [OCT(*N*)](#_bookmark140)

Returns a string representation of the octal value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to [CONV(*N*,10,8)](#_bookmark17). Returns NULL if *N* is NULL.

mysql> **SELECT** **OCT(12);**

-> '14'

• [OCTET\_LENGTH(*str*)](#_bookmark141)

[OCTET LENGTH()](#_bookmark141)\_ is a synonym for [LENGTH()](#_bookmark128).

• [ORD(*str*)](#_bookmark142)

If the leftmost character of the string *str* is a multibyte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

(1st byte code)

+ (2nd byte code \* 256)

+ (3rd byte code \* 256^2) ...

If the leftmost character is not a multibyte character, [ORD()](#_bookmark142) returns the same value as the [ASCII()](#_bookmark111) function. The function returns NULL if *str* is NULL.

mysql> **SELECT** **ORD('2');**

-> 50

• [POSITION(*substr* IN *str*)](#_bookmark143)

[POSITION(*substr* IN *str*)](#_bookmark143) is a synonym for [LOCATE(*substr*,*str*)](#_bookmark131).

• [QUOTE(*str*)](#_bookmark144)

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash (\), single quote ( '), ASCII NUL, and Control+Z preceded by a backslash. If the argument is NULL, the return value is the word “NULL” without enclosing single quotation marks.

mysql> **SELECT** **QUOTE('Don\'t!');**

-> 'Don\'t!'

mysql> **SELECT** **QUOTE(NULL);**

-> NULL

For comparison, see the quoting rules for literal strings and within the C API in Section 9.1.1, “String Literals” , and [mysql\_real\_escape\_string\_quote()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-escape-string-quote.html).

• [REPEAT(*str*,*count*)](#_bookmark150)

Returns a string consisting of the string *str* repeated *count* times. If *count* is less than 1, returns an empty string. Returns NULL if *str* or *count* is NULL.

mysql> **SELECT** **REPEAT('MySQL',** **3);**

-> 'MySQLMySQLMySQL'

• [REPLACE(*str*,*from\_str*,*to\_str*)](#_bookmark151)

Returns the string *str* with all occurrences of the string *from\_str* replaced by the string *to\_str*. [REPLACE()](#_bookmark151) performs a case-sensitive match when searching for *from\_str*.

mysql> **SELECT** **REPLACE('www.mysql** **.com',** **'w',** **'Ww');**

-> 'WwWwWw.mysql.com'

This function is multibyte safe. It returns NULL if any of its arguments are NULL.

• [REVERSE(*str*)](#_bookmark152)

Returns the string *str* with the order of the characters reversed, or NULL if *str* is NULL.

mysql> **SELECT** **REVERSE('abc');**

-> 'cba'

This function is multibyte safe.

• [RIGHT(*str*,*len*)](#_bookmark153)

Returns the rightmost *len* characters from the string *str*, or NULL if any argument is NULL.

mysql> **SELECT** **RIGHT('foobarbar',** **4);**

-> 'rbar'

This function is multibyte safe.

• [RPAD(*str*,*len*,*padstr*)](#_bookmark154)

Returns the string *str*, right-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters. If *str*, *padstr*, or *len* is NULL, the function returns NULL.

mysql> **SELECT** **RPAD('hi',5,'?');**

-> 'hi???'

mysql> **SELECT** **RPAD('hi',1,'?');**

-> 'h'

This function is multibyte safe.

• [RTRIM(*str*)](#_bookmark155)



Returns the string *str* with trailing space characters removed.

mysql> **SELECT** **RTRIM('barbar** **');**

-> 'barbar'

This function is multibyte safe, and returns NULL if *str* is NULL.

• [SOUNDEX(*str*)](#_bookmark156)

Returns a soundex string from *str*, or NULL if *str* is NULL. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the [SOUNDEX()](#_bookmark156) function returns an arbitrarily long string. You can use [SUBSTRING()](#_bookmark161) on the result to get a standard soundex string. All nonalphabetic characters in *str* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

**Important**

When using [SOUNDEX()](#_bookmark156), you should be aware of the following limitations:

• This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.

• This function is not guaranteed to provide consistent results with strings that use multibyte character sets, including utf-8. See Bug #22638 for more information.

mysql> **SELECT** **SOUNDEX('Hello');**

-> 'H400'

mysql> **SELECT** **SOUNDEX('Quadratically');**

-> 'Q36324'

**Note**

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

• [*expr1* SOUNDS LIKE *expr2*](#_bookmark157)

This is the same as [SOUNDEX(*expr1*) = SOUNDEX(*expr2*)](#_bookmark156).

• [SPACE(*N*)](#_bookmark158)

Returns a string consisting of *N*space characters, or NULL if *N* is NULL.

mysql> **SELECT** **SPACE(6);**

-> ' '

• [SUBSTR(*str*,*pos*)](#_bookmark160), [SUBSTR(*str* FROM *pos*)](#_bookmark160), [SUBSTR(*str*,*pos*,*len*)](#_bookmark160), [SUBSTR(*str* FROM](#_bookmark160) [*pos* FOR *len*)](#_bookmark160)

[SUBSTR()](#_bookmark160) is a synonym for [SUBSTRING()](#_bookmark161).

• [SUBSTRING(*str*,*pos*)](#_bookmark161), [SUBSTRING(*str* FROM *pos*)](#_bookmark161), [SUBSTRING(*str*,*pos*,*len*)](#_bookmark161), [SUBSTRING(*str* FROM *pos* FOR *len*)](#_bookmark161)

The forms without a *len* argument return a substring from string *str* starting at position *pos*. The forms with a *len* argument return a substring *len* characters long from string *str*, starting at position *pos*. The forms that use FROM are standard SQL syntax. It is also possible to use a negative value for *pos*. In this case, the beginning of the substring is *pos* characters from the end of the string, rather than the beginning. A negative value may be used for *pos* in any of the forms of this function. A value of 0 for *pos* returns an empty string.

For all forms of [SUBSTRING()](#_bookmark161), the position of the first character in the string from which the substring is to be extracted is reckoned as 1.

mysql> **SELECT** **SUBSTRING('Quadratically',5);**

-> 'ratically'

mysql> **SELECT** **SUBSTRING('foobarbar'** **FROM** **4);**

-> 'barbar'

mysql> **SELECT** **SUBSTRING('Quadratically',5,6);**

-> 'ratica'

mysql> **SELECT** **SUBSTRING('Sakila',** **-3);**

-> 'ila'

mysql> **SELECT** **SUBSTRING('Sakila',** **-5,** **3);**

-> 'aki'

mysql> **SELECT** **SUBSTRING('Sakila'** **FROM** **-4** **FOR** **2);**

-> 'ki'

This function is multibyte safe. It returns NULL if any of its arguments are NULL. If *len* is less than 1, the result is the empty string.

• [SUBSTRING\_INDEX(*str*,*delim*,*count*)](#_bookmark162)

Returns the substring from string *str* before *count* occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned. [SUBSTRING\_INDEX()](#_bookmark162) performs a case-sensitive match when searching for *delim*.

mysql> **SELECT** **SUBSTRING\_INDEX('www.mysql** **.com',** **'** **.** **',** **2);**

-> 'www .mysql'

mysql> **SELECT** **SUBSTRING\_INDEX('www** **.mysql** **.com',** **'** **.** **',** **-2);**

-> 'mysql.com'

This function is multibyte safe.

SUBSTRING\_INDEX() returns NULL if any of its arguments are NULL.

• [TO\_BASE64(*str*)](#_bookmark163)

Converts the string argument to base-64 encoded form and returns the result as a character string with the connection character set and collation. If the argument is not a string, it is converted to a string before conversion takes place. The result is NULL if the argument is NULL. Base-64 encoded strings can be decoded using the [FROM\_BASE64()](#_bookmark123) function.

mysql> **SELECT** **TO\_BASE64('abc'),** **FROM\_BASE64(TO\_BASE64('abc'));**

-> 'JWJj', 'abc'

Different base-64 encoding schemes exist. These are the encoding and decoding rules used by [TO\_BASE64()](#_bookmark163) and [FROM\_BASE64()](#_bookmark123):

• The encoding for alphabet value 62 is '+'.

• The encoding for alphabet value 63 is '/'.

• Encoded output consists of groups of 4 printable characters. Each 3 bytes of the input data are encoded using 4 characters. If the last group is incomplete, it is padded with '=' characters to a length of 4.

• A newline is added after each 76 characters of encoded output to divide long output into multiple lines.

• Decoding recognizes and ignores newline, carriage return, tab, and space.

• [TRIM([{BOTH | LEADING | TRAILING} [*remstr*] FROM] *str*)](#_bookmark164), [TRIM([*remstr* FROM]](#_bookmark164) [*str*)](#_bookmark164)

Returns the string *str* with all *remstr* prefixes or suffixes removed. If none of the specifiers BOTH, LEADING, or TRAILING is given, BOTH is assumed. *remstr* is optional and, if not specified, spaces are removed.

mysql> **SELECT** **TRIM('** **bar** **');**

-> 'bar'

mysql> **SELECT** **TRIM(LEADING** **'x'** **FROM** **'xxxbarxxx');**

-> 'barxxx'

mysql> **SELECT** **TRIM(BOTH** **'x'** **FROM** **'xxxbarxxx');**

-> 'bar'

mysql> **SELECT** **TRIM(TRAILING** **'xyz'** **FROM** **'barxxyz');**

-> 'barx'

This function is multibyte safe. It returns NULL if any of its arguments are NULL. • [UCASE(*str*)](#_bookmark165)

[UCASE()](#_bookmark165) is a synonym for [UPPER()](#_bookmark167).

UCASE() used within views is rewritten as UPPER().

• [UNHEX(*str*)](#_bookmark166)

For a string argument *str*, [UNHEX(*str*)](#_bookmark166) interprets each pair of characters in the argument as a hexadecimal number and converts it to the byte represented by the number. The return value is a binary string.

mysql> **SELECT** **UNHEX('4D7953514C');**

-> 'MySQL'

mysql> **SELECT** **X'4D7953514C';**

-> 'MySQL'

mysql> **SELECT** **UNHEX(HEX('string'));**

-> 'string'

mysql> **SELECT** **HEX(UNHEX('1267'));**

-> '1267'

The characters in the argument string must be legal hexadecimal digits: '0' .. '9', 'A' .. 'F', 'a' .. 'f'. If the argument contains any nonhexadecimal digits, or is itself NULL, the result is NULL:

mysql> **SELECT** **UNHEX('GG');**

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

| UNHEX('GG') |

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

| NULL |

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

mysql> **SELECT** **UNHEX(NULL);**

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

| UNHEX(NULL) |

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

| NULL |

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

A NULL result can also occur if the argument to [UNHEX()](#_bookmark166) is a BINARY column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example, '41' is stored into a CHAR(3) column as '41 ' and retrieved as '41' (with the trailing pad space stripped), so [UNHEX()](#_bookmark166) for the column value returns X'41'. By contrast, '41' is stored into



a BINARY(3) column as '41\0' and retrieved as '41\0' (with the trailing pad 0x00 byte not stripped). '\0' is not a legal hexadecimal digit, so [UNHEX()](#_bookmark166) for the column value returns NULL.

For a numeric argument *N*, the inverse of [HEX(*N*](#_bookmark48)) is not performed by [UNHEX()](#_bookmark166). Use [CONV(HEX(*N*),16,10)](#_bookmark17) instead. See the description of [HEX()](#_bookmark48).

If [UNHEX()](#_bookmark166) is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [UPPER(*str*)](#_bookmark167)

Returns the string *str* with all characters changed to uppercase according to the current character set mapping, or NULL if *str* is NULL. The default character set is utf8mb4.

mysql> **SELECT** **UPPER('Hej');**

-> 'HEJ'

See the description of [LOWER()](#_bookmark132) for information that also applies to [UPPER()](#_bookmark167). This included information about how to perform lettercase conversion of binary strings (BINARY, VARBINARY, BLOB) for which these functions are ineffective, and information about case folding for Unicode character sets.

This function is multibyte safe.

UCASE() used within views is rewritten as UPPER().

• [WEIGHT\_STRING(*str* [AS {CHAR|BINARY}(*N*)] [*flags*])](#_bookmark168)

This function returns the weight string for the input string. The return value is a binary string that represents the comparison and sorting value of the string, or NULL if the argument is NULL. It has these properties:

• If [WEIGHT\_STRING(*str1*)](#_bookmark168) = [WEIGHT\_STRING(*str2*)](#_bookmark168), then *str1* = *str2* (*str1* and *str2* are considered equal)

• If [WEIGHT\_STRING(*str1*)](#_bookmark168) < [WEIGHT\_STRING(*str2*)](#_bookmark168), then *str1* < *str2* (*str1* sorts before *str2*)

[WEIGHT\_STRING()](#_bookmark168) is a debugging function intended for internal use. Its behavior can change without notice between MySQL versions. It can be used for testing and debugging of collations, especially if you are adding a new collation. See Section 10.14, “Adding a Collation to a Character

Set” .

This list briefly summarizes the arguments. More details are given in the discussion following the list.

• *str*: The input string expression.

• AS clause: Optional; cast the input string to a given type and length.

• *flags*: Optional; unused.

The input string, *str*, is a string expression. If the input is a nonbinary (character) string such as a CHAR, VARCHAR, or TEXT value, the return value contains the collation weights for the string. If the input is a binary (byte) string such as a BINARY, VARBINARY, or BLOB value, the return value is the same as the input (the weight for each byte in a binary string is the byte value). If the input is NULL, [WEIGHT\_STRING()](#_bookmark168) returns NULL.

Examples:

mysql> **SET** **@s** **=** **\_utf8mb4** **'AB'** **COLLATE** **utf8mb4\_0900\_ai\_ci;**

mysql> **SELECT** **@s,** **HEX(@s),** **HEX(WEIGHT\_STRING(@s));**



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

| @s | HEX(@s) | HEX(WEIGHT\_STRING(@s)) |

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

| AB | 4142 | 1C471C60 |

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

mysql> **SET** **@s** **=** **\_utf8mb4** **'ab'** **COLLATE** **utf8mb4\_0900\_ai\_ci;**

mysql> **SELECT** **@s,** **HEX(@s),** **HEX(WEIGHT\_STRING(@s));**

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

| @s | HEX(@s) | HEX(WEIGHT\_STRING(@s)) |

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

| ab | 6162 | 1C471C60 |

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

mysql> **SET** **@s** **=** **CAST('AB'** **AS** **BINARY);**

mysql> **SELECT** **@s,** **HEX(@s),** **HEX(WEIGHT\_STRING(@s));**

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

| @s | HEX(@s) | HEX(WEIGHT\_STRING(@s)) |

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

| AB | 4142 | 4142 |

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

mysql> **SET** **@s** **=** **CAST('ab'** **AS** **BINARY);**

mysql> **SELECT** **@s,** **HEX(@s),** **HEX(WEIGHT\_STRING(@s));**

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

| @s | HEX(@s) | HEX(WEIGHT\_STRING(@s)) |

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

| ab | 6162 | 6162 |

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

The preceding examples use [HEX()](#_bookmark48) to display the [WEIGHT STRING()](#_bookmark168)\_ result. Because the result is a binary value, [HEX()](#_bookmark48) can be especially useful when the result contains nonprinting values, to display it in printable form:

mysql> **SET** **@s** **=** **CONVERT(X'C39F'** **USING** **utf8mb4)** **COLLATE** **utf8mb4\_czech\_ci;**

mysql> **SELECT** **HEX(WEIGHT\_STRING(@s));**

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

| HEX(WEIGHT\_STRING(@s)) |

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

|  |  |
| --- | --- |
| | 0FEA0FEA | | |

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

For non-NULL return values, the data type of the value is VARBINARY if its length is within the maximum length for VARBINARY, otherwise the data type is BLOB.

The AS clause may be given to cast the input string to a nonbinary or binary string and to force it to a given length:

• AS CHAR(*N*) casts the string to a nonbinary string and pads it on the right with spaces to a length of *N*characters. *N* must be at least 1. If *N* is less than the length of the input string, the string is truncated to *N*characters. No warning occurs for truncation.

• AS BINARY(*N*) is similar but casts the string to a binary string, *N* is measured in bytes (not characters), and padding uses 0x00 bytes (not spaces).

mysql> **SET** **NAMES** **'latin1';**

mysql> **SELECT** **HEX(WEIGHT\_STRING('ab'** **AS** **CHAR(4)));**

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

| HEX(WEIGHT\_STRING('ab' AS CHAR(4))) |

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

| 41422020 |

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

mysql> **SET** **NAMES** **'utf8mb4';**

mysql> **SELECT** **HEX(WEIGHT\_STRING('ab'** **AS** **CHAR(4)));**

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

| HEX(WEIGHT\_STRING('ab' AS CHAR(4))) |

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

| 1C471C60 |

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

mysql> **SELECT** **HEX(WEIGHT\_STRING('ab'** **AS** **BINARY(4)));**

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

| HEX(WEIGHT\_STRING('ab' AS BINARY(4))) |

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

|  |  |
| --- | --- |
| | 61620000 | | |

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

The *flags* clause currently is unused.

If [WEIGHT\_STRING()](#_bookmark168) is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

**12.8.1** **String** **Comparison** **Functions** **and** **Operators**

**Table** **12.13** **String** **Comparison** **Functions** **and** **Operators**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [LIKE](#_bookmark129) | Simple pattern matching |
| [NOT LIKE](#_bookmark138) | Negation of simple pattern matching |
| [STRCMP()](#_bookmark159) | Compare two strings |

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case-sensitive, the comparison is performed in case-sensitive fashion.

If a string function is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [*expr* LIKE *pat* [ESCAPE '*escape\_char*']](#_bookmark129)

Pattern matching using an SQL pattern. Returns 1 (TRUE) or 0 (FALSE). If either *expr* or *pat* is NULL, the result is NULL.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column. In the latter case, the column must be defined as one of the MySQL string types (see Section 11.3, “String Data Types”).

Per the SQL standard, [LIKE](#_bookmark129) performs matching on a per-character basis, thus it can produce results different from the = comparison operator:

mysql> **SELECT** **'ä'** **LIKE** **'ae'** **COLLATE** **latin1\_german2\_ci;**

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

| 'ä' LIKE 'ae' COLLATE latin1\_german2\_ci |

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

| 0 |

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

mysql> **SELECT** **'ä'** **=** **'ae'** **COLLATE** **latin1\_german2\_ci;**

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

| 'ä' = 'ae' COLLATE latin1\_german2\_ci |

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

| 1 |

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

In particular, trailing spaces are always significant. This differs from comparisons performed with the = operator, for which the significance of trailing spaces in nonbinary strings (CHAR, VARCHAR, and TEXT values) depends on the pad attribute of the collation used for the comparison. For more information, see Trailing Space Handling in Comparisons.

With [LIKE](#_bookmark129) you can use the following two wildcard characters in the pattern:

• % matches any number of characters, even zero characters.

• \_ matches exactly one character.

mysql> **SELECT** **'David!'** **LIKE** **'David\_';**

-> 1

mysql> **SELECT** **'David!'** **LIKE** **'%D%v%';**

-> 1

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the ESCAPE character, \ is assumed, unless the NO\_BACKSLASH\_ESCAPES SQL mode is enabled. In that case, no escape character is used.

• \% matches one % character.

• \\_ matches one \_ character.

mysql> **SELECT** **'David!'** **LIKE** **'David\\_';**

-> 0

mysql> **SELECT** **'David\_'** **LIKE** **'David\\_';**

-> 1

To specify a different escape character, use the ESCAPE clause:

mysql> **SELECT** **'David\_'** **LIKE** **'David|\_'** **ESCAPE** **'** **|** **';**

-> 1

The escape sequence should be one character long to specify the escape character, or empty to specify that no escape character is used. The expression must evaluate as a constant at execution time. If the NO\_BACKSLASH\_ESCAPES SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case-sensitive unless one of the operands is case-sensitive (uses a case-sensitive collation or is a binary string):

mysql> **SELECT** **'abc'** **LIKE** **'ABC';**

-> 1

mysql> **SELECT** **'abc'** **LIKE** **\_utf8mb4** **'ABC'** **COLLATE** **utf8mb4\_0900\_as\_cs;**

-> 0

mysql> **SELECT** **'abc'** **LIKE** **\_utf8mb4** **'ABC'** **COLLATE** **utf8mb4\_bin;**

-> 0

mysql> **SELECT** **'abc'** **LIKE** **BINARY** **'ABC';**

-> 0

As an extension to standard SQL, MySQL permits [LIKE](#_bookmark129) on numeric expressions.

mysql> **SELECT** **10** **LIKE** **'1%';**

-> 1

MySQL attempts in such cases to perform implicit conversion of the expression to a string. See

Section 12.3, “Type Conversion in Expression Evaluation” .

**Note**

MySQL uses C escape syntax in strings (for example, \n to represent the newline character). If you want a [LIKE](#_bookmark129) string to contain a literal \, you must double it. (Unless the NO\_BACKSLASH\_ESCAPES SQL mode is enabled, in which case no escape character is used.) For example, to search for \n, specify it as \\n. To search for \, specify it as \\\\; this is because the



backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against.

Exception: At the end of the pattern string, backslash can be specified as \\. At the end of the string, backslash stands for itself because there is nothing following to escape. Suppose that a table contains the following values:

mysql> **SELECT** **filename** **FROM** **t1;**

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

| filename |

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

| C: |

| C:\ |

| C:\Programs |

| C:\Programs\ |

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

To test for values that end with backslash, you can match the values using either of the following patterns:

mysql> **SELECT** **filename,** **filename** **LIKE** **'%\\'** **FROM** **t1;**

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

| filename | filename LIKE '%\\' |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  | | C:  C:\  C:\Programs  C:\Programs\ | |  |  |  | | 0  1  0  1 | |  |  |  | |

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

mysql> **SELECT** **filename,** **filename** **LIKE** **'%\\\\'** **FROM** **t1;**

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

| filename | filename LIKE '%\\\\' |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  | | C:  C:\  C:\Programs  C:\Programs\ | |  |  |  | | 0  1  0  1 | |  |  |  | |

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

• [*expr* NOT LIKE *pat* [ESCAPE '*escape\_char*']](#_bookmark138)

This is the same as NOT (*expr* LIKE *pat* [ESCAPE '*escape\_char*']).

**Note**

Aggregate queries involving [NOT LIKE](#_bookmark138) comparisons with columns containing NULL may yield unexpected results. For example, consider the following table and data:

CREATE TABLE foo (bar VARCHAR(10));

INSERT INTO foo VALUES (NULL), (NULL);

The query SELECT COUNT(\*) FROM foo WHERE bar LIKE '%baz%'; returns 0. You might assume that SELECT COUNT(\*) FROM foo WHERE bar NOT LIKE '%baz%'; would return 2. However, this is not the case: The second query returns 0. This is because NULL NOT LIKE *expr* always returns NULL, regardless of the value of *expr*. The same is true for aggregate queries involving NULL and comparisons using [NOT RLIKE](#_bookmark139) or [NOT REGEXP](#_bookmark139). In such cases, you must test explicitly for NOT NULL using OR (and not AND), as shown here:

SELECT COUNT(\*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;

• [STRCMP(*expr1*,*expr2*)](#_bookmark159)

[STRCMP()](#_bookmark159) returns 0 if the strings are the same, -1 if the first argument is smaller than the second according to the current sort order, and NULL if either argument is NULL. It returns 1 otherwise.

|  |  |
| --- | --- |
| mysql> **SELECT**  -> -1  mysql> **SELECT**  -> 1  mysql> **SELECT**  -> 0 | **STRCMP('text',** **'text2');**  **STRCMP('text2',** **'text');**  **STRCMP('text',** **'text');** |

[STRCMP()](#_bookmark159) performs the comparison using the collation of the arguments.

mysql> **SET** **@s1** **=** **\_utf8mb4** **'x'** **COLLATE** **utf8mb4\_0900\_ai\_ci;**

mysql> **SET** **@s2** **=** **\_utf8mb4** **'X'** **COLLATE** **utf8mb4\_0900\_ai\_ci;**

mysql> **SET** **@s3** **=** **\_utf8mb4** **'x'** **COLLATE** **utf8mb4\_0900\_as\_cs;**

mysql> **SET** **@s4** **=** **\_utf8mb4** **'X'** **COLLATE** **utf8mb4\_0900\_as\_cs;**

mysql> **SELECT** **STRCMP(@s1,** **@s2),** **STRCMP(@s3,** **@s4);**

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

| STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |

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

| 0 | -1 |

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

If the collations are incompatible, one of the arguments must be converted to be compatible with the other. See Section 10.8.4, “Collation Coercibility in Expressions” .

mysql> SET @s1 = \_utf8mb4 'x' COLLATE utf8mb4\_0900\_ai\_ci;

mysql> SET @s2 = \_utf8mb4 'X' COLLATE utf8mb4\_0900\_ai\_ci;

mysql> SET @s3 = \_utf8mb4 'x' COLLATE utf8mb4\_0900\_as\_cs;

mysql> SET @s4 = \_utf8mb4 'X' COLLATE utf8mb4\_0900\_as\_cs;

-->

mysql> **SELECT** **STRCMP(@s1,** **@s3);**

ERROR 1267 (HY000): Illegal mix of collations (utf8mb4\_0900\_ai\_ci,IMPLICIT)

and (utf8mb4\_0900\_as\_cs,IMPLICIT) for operation 'strcmp'

mysql> **SELECT** **STRCMP(@s1,** **@s3** **COLLATE** **utf8mb4\_0900\_ai\_ci);**

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

| STRCMP(@s1, @s3 COLLATE utf8mb4\_0900\_ai\_ci) |

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

| 0 |

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

**12.8.2** **Regular** **Expressions**

**Table** **12.14** **Regular** **Expression** **Functions** **and** **Operators**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [NOT REGEXP](#_bookmark139) | Negation of REGEXP |
| [REGEXP](#_bookmark145) | Whether string matches regular expression |
| [REGEXP\_INSTR()](#_bookmark146) | Starting index of substring matching regular expression |
| [REGEXP\_LIKE()](#_bookmark147) | Whether string matches regular expression |
| [REGEXP\_REPLACE()](#_bookmark148) | Replace substrings matching regular expression |
| [REGEXP\_SUBSTR()](#_bookmark149) | Return substring matching regular expression |
| [RLIKE](#_bookmark145) | Whether string matches regular expression |

A regular expression is a powerful way of specifying a pattern for a complex search. This section discusses the functions and operators available for regular expression matching and illustrates, with examples, some of the special characters and constructs that can be used for regular expression operations. See also Section 3.3.4.7, “Pattern Matching” .

MySQL implements regular expression support using International Components for Unicode (ICU), which provides full Unicode support and is multibyte safe. (Prior to MySQL 8.0.4, MySQL used Henry

Spencer's implementation of regular expressions, which operates in byte-wise fashion and is not multibyte safe. For information about ways in which applications that use regular expressions may be affected by the implementation change, see [Regular Expression Compatibility Considerations](#_bookmark171).)

Prior to MySQL 8.0.22, it was possible to use binary string arguments with these functions, but they yielded inconsistent results. In MySQL 8.0.22 and later, use of a binary string with any of the MySQL regular expression functions is rejected with [ER\_CHARACTER\_SET\_MISMATCH](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_character_set_mismatch).

• [Regular Expression Function and Operator Descriptions](#_bookmark172)

• [Regular Expression Syntax](#_bookmark173)

• [Regular Expression Resource Control](#_bookmark174)

• [Regular Expression Compatibility Considerations](#_bookmark171)

**Regular** **Expression** **Function** **and** **Operator** **Descriptions**

• [*expr* NOT REGEXP *pat*](#_bookmark139), [*expr* NOT RLIKE *pat*](#_bookmark139) This is the same as NOT (*expr* REGEXP *pat*).

• [*expr* REGEXP *pat*](#_bookmark145), [*expr* RLIKE *pat*](#_bookmark145)

Returns 1 if the string *expr* matches the regular expression specified by the pattern *pat*, 0 otherwise. If *expr* or *pat* is NULL, the return value is NULL.

[REGEXP](#_bookmark145) and [RLIKE](#_bookmark145) are synonyms for [REGEXP LIKE()](#_bookmark147)\_.

For additional information about how matching occurs, see the description for [REGEXP\_LIKE()](#_bookmark147).

mysql> **SELECT** **'Michael!'** **REGEXP** **'** **.\*';**

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

| 'Michael!' REGEXP ' .\*' |

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

| 1 |

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

mysql> **SELECT** **'new\*\n\*line'** **REGEXP** **'new\\\*** **.\\\*line';**

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

| 'new\*\n\*line' REGEXP 'new\\\* .\\\*line' |

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

| 0 |

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

mysql> **SELECT** **'a'** **REGEXP** **'^[a-d]';**

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

| 'a' REGEXP '^[a-d]' |

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

| 1 |

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

• [REGEXP\_INSTR(*expr*, *pat*[, *pos*[, *occurrence*[, *return\_option*[,](#_bookmark146) [*match\_type*]]]])](#_bookmark146)

Returns the starting index of the substring of the string *expr* that matches the regular expression specified by the pattern *pat*, 0 if there is no match. If *expr* or *pat* is NULL, the return value is NULL. Character indexes begin at 1.

[REGEXP\_INSTR()](#_bookmark146) takes these optional arguments:

• *pos*: The position in *expr* at which to start the search. If omitted, the default is 1.

• *occurrence*: Which occurrence of a match to search for. If omitted, the default is 1.

• *return\_option*: Which type of position to return. If this value is 0, [REGEXP\_INSTR()](#_bookmark146) returns the position of the matched substring's first character. If this value is 1, [REGEXP\_INSTR()](#_bookmark146) returns the position following the matched substring. If omitted, the default is 0.

• *match\_type*: A string that specifies how to perform matching. The meaning is as described for [REGEXP\_LIKE()](#_bookmark147).

For additional information about how matching occurs, see the description for [REGEXP\_LIKE()](#_bookmark147).

mysql> **SELECT** **REGEXP\_INSTR('dog** **cat** **dog',** **'dog');**

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

| REGEXP\_INSTR('dog cat dog', 'dog') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_INSTR('dog** **cat** **dog',** **'dog',** **2);**

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

| REGEXP\_INSTR('dog cat dog', 'dog', 2) |

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

| 9 |

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

mysql> **SELECT** **REGEXP\_INSTR('aa** **aaa** **aaaa',** **'a{2}');**

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

| REGEXP\_INSTR('aa aaa aaaa', 'a{2}') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_INSTR('aa** **aaa** **aaaa',** **'a{4}');**

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

| REGEXP\_INSTR('aa aaa aaaa', 'a{4}') |

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

| 8 |

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

• [REGEXP\_LIKE(*expr*, *pat*[, *match\_type*])](#_bookmark147)

Returns 1 if the string *expr* matches the regular expression specified by the pattern *pat*, 0 otherwise. If *expr* or *pat* is NULL, the return value is NULL.

The pattern can be an extended regular expression, the syntax for which is discussed in [Regular](#_bookmark173) [Expression Syntax](#_bookmark173). The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

The optional *match\_type* argument is a string that may contain any or all the following characters specifying how to perform matching:

• c: Case-sensitive matching.

• i: Case-insensitive matching.

• m: Multiple-line mode. Recognize line terminators within the string. The default behavior is to match line terminators only at the start and end of the string expression.

• n: The . character matches line terminators. The default is for . matching to stop at the end of a line.

• u: Unix-only line endings. Only the newline character is recognized as a line ending by the ., ^, and $ match operators.

If characters specifying contradictory options are specified within *match\_type*, the rightmost one takes precedence.

By default, regular expression operations use the character set and collation of the *expr* and *pat* arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in Section 10.8.4,



“Collation Coercibility in Expressions” . Arguments may be specified with explicit collation indicators to change comparison behavior.

mysql> **SELECT** **REGEXP\_LIKE('CamelCase',** **'CAMELCASE');**

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

| REGEXP\_LIKE('CamelCase', 'CAMELCASE') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_LIKE('CamelCase',** **'CAMELCASE'** **COLLATE** **utf8mb4\_0900\_as\_cs);**

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

| REGEXP\_LIKE('CamelCase', 'CAMELCASE' COLLATE utf8mb4\_0900\_as\_cs) |

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

| 0 |

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

*match\_type* may be specified with the c or i characters to override the default case sensitivity. Exception: If either argument is a binary string, the arguments are handled in case-sensitive fashion as binary strings, even if *match\_type* contains the i character.

**Note**

MySQL uses C escape syntax in strings (for example, \n to represent the newline character). If you want your *expr* or *pat* argument to contain a literal \, you must double it. (Unless the NO\_BACKSLASH\_ESCAPES SQL mode is enabled, in which case no escape character is used.)

mysql> **SELECT** **REGEXP\_LIKE('Michael!',** **'** **.\*');**

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

| REGEXP\_LIKE('Michael!', ' .\*') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_LIKE('new\*\n\*line',** **'new\\\*** **.\\\*line');**

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

| REGEXP\_LIKE('new\*\n\*line', 'new\\\* .\\\*line') |

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

| 0 |

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

mysql> **SELECT** **REGEXP\_LIKE('a',** **'^[a-d]');**

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

| REGEXP\_LIKE('a', '^[a-d]') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_LIKE('abc',** **'ABC');**

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

| REGEXP\_LIKE('abc', 'ABC') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_LIKE('abc',** **'ABC',** **'c');**

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

| REGEXP\_LIKE('abc', 'ABC', 'c') |

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

| 0 |

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

• [REGEXP\_REPLACE(*expr*, *pat*, *repl*[, *pos*[, *occurrence*[, *match\_type*]]])](#_bookmark148)

Replaces occurrences in the string *expr* that match the regular expression specified by the pattern *pat* with the replacement string *repl*, and returns the resulting string. If *expr*, *pat*, or *repl* is NULL, the return value is NULL.

[REGEXP\_REPLACE()](#_bookmark148) takes these optional arguments:

• *pos*: The position in *expr* at which to start the search. If omitted, the default is 1.

• *occurrence*: Which occurrence of a match to replace. If omitted, the default is 0 (which means “replace all occurrences”).

• *match\_type*: A string that specifies how to perform matching. The meaning is as described for [REGEXP\_LIKE()](#_bookmark147).

Prior to MySQL 8.0.17, the result returned by this function used the UTF-16 character set; in MySQL 8.0.17 and later, the character set and collation of the expression searched for matches is used. (Bug #94203, Bug #29308212)

For additional information about how matching occurs, see the description for [REGEXP\_LIKE()](#_bookmark147).

mysql> **SELECT** **REGEXP\_REPLACE('a** **b** **c',** **'b',** **'X');**

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

| REGEXP\_REPLACE('a b c', 'b', 'X') |

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

| a X c |

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

mysql> **SELECT** **REGEXP\_REPLACE('abc** **def** **ghi',** **'[a-z]+',** **'X',** **1,** **3);**

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

| REGEXP\_REPLACE('abc def ghi', '[a-z]+', 'X', 1, 3) |

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

| abc def X |

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

• [REGEXP\_SUBSTR(*expr*, *pat*[, *pos*[, *occurrence*[, *match\_type*]]])](#_bookmark149)

Returns the substring of the string *expr* that matches the regular expression specified by the pattern *pat*, NULL if there is no match. If *expr* or *pat* is NULL, the return value is NULL.

[REGEXP\_SUBSTR()](#_bookmark149) takes these optional arguments:

• *pos*: The position in *expr* at which to start the search. If omitted, the default is 1.

• *occurrence*: Which occurrence of a match to search for. If omitted, the default is 1.

• *match\_type*: A string that specifies how to perform matching. The meaning is as described for [REGEXP\_LIKE()](#_bookmark147).

Prior to MySQL 8.0.17, the result returned by this function used the UTF-16 character set; in MySQL 8.0.17 and later, the character set and collation of the expression searched for matches is used. (Bug #94203, Bug #29308212)

For additional information about how matching occurs, see the description for [REGEXP\_LIKE()](#_bookmark147).

mysql> **SELECT** **REGEXP\_SUBSTR('abc** **def** **ghi',** **'[a-z]+');**

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

| REGEXP\_SUBSTR('abc def ghi', '[a-z]+') |

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

| abc |

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

mysql> **SELECT** **REGEXP\_SUBSTR('abc** **def** **ghi',** **'[a-z]+',** **1,** **3);**

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

| REGEXP\_SUBSTR('abc def ghi', '[a-z]+', 1, 3) |

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

-> 1

mysql> **SELECT** **REGEXP\_LIKE('fo\r\nfo',** **'^f.\*$',** **'m');**

|  |  |
| --- | --- |
| | ghi | | |

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

**Regular** **Expression** **Syntax**

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression hello matches hello and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression hello |world contains the | alternation operator and matches either the hello or world.

As a more complex example, the regular expression B[an]\*s matches any of the strings Bananas, Baaaaas, Bs, and any other string starting with a B, ending with an s, and containing any number of a or n characters in between.

The following list covers some of the basic special characters and constructs that can be used in regular expressions. For information about the full regular expression syntax supported by the ICU library used to implement regular expression support, visit the [International Components for Unicode](https://unicode-org.github.io/icu/userguide/) [web site](https://unicode-org.github.io/icu/userguide/).

• ^

Match the beginning of a string.

mysql> **SELECT** **REGEXP\_LIKE('fo\nfo',** **'^fo$');**

-> 0

-> 1

mysql> **SELECT** **REGEXP\_LIKE('fofo',** **'^fo');**

• $

Match the end of a string.

|  |  |
| --- | --- |
| mysql> **SELECT** **REGEXP\_LIKE('fo\no',** **'^fo\no$');**  mysql> **SELECT** **REGEXP\_LIKE('fo\no',** **'^fo$');** | -> 1  -> 0 |

• .

Match any character (including carriage return and newline, although to match these in the middle of a string, the m (multiple line) match-control character or the (?m) within-pattern modifier must be given).

mysql> **SELECT** **REGEXP\_LIKE('fofo',** **'^f** **.\*$');**

-> 1

mysql> **SELECT** **REGEXP\_LIKE('fo\r\nfo',** **'^f.\*$');**

-> 0

mysql> **SELECT** **REGEXP\_LIKE('fo\r\nfo',** **'(?m)^f.\*$');**

-> 1

• a\*

Match any sequence of zero or more a characters.

mysql> **SELECT** **REGEXP\_LIKE('Ban',** **'^Ba\*n');**

-> 1

-> 1

-> 1

mysql> **SELECT** **REGEXP\_LIKE('Baaan',** **'^Ba\*n');**

mysql> **SELECT** **REGEXP\_LIKE('Bn',** **'^Ba\*n');**

• a+

Match any sequence of one or more a characters.

mysql> **SELECT** **REGEXP\_LIKE('Ban',** **'^Ba+n');**

-> 1

-> 0

mysql> **SELECT** **REGEXP\_LIKE('Bn',** **'^Ba+n');**

• a?

Match either zero or one a character.

-> 1

mysql> **SELECT** **REGEXP\_LIKE('Bn',** **'^Ba?n');**

mysql> **SELECT** **REGEXP\_LIKE('Ban',** **'^Ba?n');**

-> 1

-> 0

mysql> **SELECT** **REGEXP\_LIKE('Baan',** **'^Ba?n');**

• de |abc

Alternation; match either of the sequences de or abc.

mysql> **SELECT** **REGEXP\_LIKE('pi',** **'pi** **|apa');**

-> 1

-> 0

-> 1

-> 1

-> 1

-> 0

mysql> **SELECT** **REGEXP\_LIKE('axe',** **'pi** **|apa');**

mysql> **SELECT** **REGEXP\_LIKE('apa',** **'pi** **|apa');**

mysql> **SELECT** **REGEXP\_LIKE('apa',** **'^(pi** **|apa)$');**

mysql> **SELECT** **REGEXP\_LIKE('pi',** **'^(pi** **|apa)$');**

mysql> **SELECT** **REGEXP\_LIKE('pix',** **'^(pi** **|apa)$');**

• (abc)\*

Match zero or more instances of the sequence abc.

mysql> **SELECT** **REGEXP\_LIKE('pi',** **'^(pi)\*$');**

-> 1

-> 0

-> 1

mysql> **SELECT** **REGEXP\_LIKE('pip',** **'^(pi)\*$');**

mysql> **SELECT** **REGEXP\_LIKE('pipi',** **'^(pi)\*$');**

• {1}, {2,3}

Repetition; {*n*} and {*m*,*n*} notation provide a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. *m* and *n* are integers.

• a\*

Can be written as a{0,}.

• a+

Can be written as a{1,}.

• a?

Can be written as a{0,1}.

To be more precise, a{*n*} matches exactly *n* instances of a. a{*n*,} matches *n* or more instances of a. a{*m*,*n*} matches *m* through *n* instances of a, inclusive. If both *m* and *n* are given, *m* must be less than or equal to *n*.

-> 0

mysql> **SELECT** **REGEXP\_LIKE('abcde',** **'a[bcd]{2}e');** mysql> **SELECT** **REGEXP\_LIKE('abcde',** **'a[bcd]{3}e');** mysql> **SELECT** **REGEXP\_LIKE('abcde',** **'a[bcd]{1,10}e');**

-> 1

-> 1

Matches any character that is (or is not, if ^ is used) either a, b, c, d or X. A - character between two other characters forms a range that matches all characters from the first character to the second. For example, [0-9] matches any decimal digit. To include a literal ] character, it must immediately follow the opening bracket [. To include a literal - character, it must be written first or last. Any character that does not have a defined special meaning inside a [] pair matches only itself.

• [a-dX], [^a-dX]

-> 1

mysql> **SELECT**

mysql> **SELECT**

mysql> **SELECT**

mysql> **SELECT**

mysql> **SELECT**

mysql> **SELECT**

**REGEXP\_LIKE('aXbc',** **'[a-dXYZ]');**

-> 0

**REGEXP\_LIKE('aXbc',** **'^[a-dXYZ]$');** **REGEXP\_LIKE('aXbc',** **'^[a-dXYZ]+$');** **REGEXP\_LIKE('aXbc',** **'^[^a-dXYZ]+$');** **REGEXP\_LIKE('gheis',** **'^[^a-dXYZ]+$');** **REGEXP\_LIKE('gheisa',** **'^[^a-dXYZ]+$');**

-> 1

-> 0

-> 1

-> 0

• [=character\_class=]

Within a bracket expression (written using [ and ]), [=character\_class=] represents an equivalence class. It matches all characters with the same collation value, including itself. For

example, if o and (+) are the members of an equivalence class, [[=o=]], [[=(+)=]], and [o(+)] are all synonymous. An equivalence class may not be used as an endpoint of a range.

• [:character\_class:]

Within a bracket expression (written using [ and ]), [:character\_class:] represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the ctype(3) manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

|  |  |
| --- | --- |
| **Character** **Class** **Name** | **Meaning** |
| alnum | Alphanumeric characters |
| alpha | Alphabetic characters |
| blank | Whitespace characters |
| cntrl | Control characters |
| digit | Digit characters |
| graph | Graphic characters |
| lower | Lowercase alphabetic characters |
| print | Graphic or space characters |
| punct | Punctuation characters |
| space | Space, tab, newline, and carriage return |
| upper | Uppercase alphabetic characters |
| xdigit | Hexadecimal digit characters |

|  |  |
| --- | --- |
| mysql> **SELECT** **REGEXP\_LIKE('justalnums',** **'[[:alnum:]]+');**  mysql> **SELECT** **REGEXP\_LIKE('!!',** **'[[:alnum:]]+');** | -> 1  -> 0 |

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string 1+2 that contains the special + character, only the last of the following regular expressions is the correct one:

|  |  |
| --- | --- |
| mysql> **SELECT** **REGEXP\_LIKE('1+2',** **'1+2');**  mysql> **SELECT** **REGEXP\_LIKE('1+2',** **'1\+2');**  mysql> **SELECT** **REGEXP\_LIKE('1+2',** **'1\\+2');** | -> 0  -> 0  -> 1 |

**Regular** **Expression** **Resource** **Control**

[REGEXP\_LIKE()](#_bookmark147) and similar functions use resources that can be controlled by setting system variables:

• The match engine uses memory for its internal stack. To control the maximum available memory for the stack in bytes, set the regexp\_stack\_limit system variable.

• The match engine operates in steps. To control the maximum number of steps performed by the engine (and thus indirectly the execution time), set the regexp\_time\_limit system variable. Because this limit is expressed as number of steps, it affects execution time only indirectly. Typically, it is on the order of milliseconds.

**Regular** **Expression** **Compatibility** **Considerations**

Prior to MySQL 8.0.4, MySQL used the Henry Spencer regular expression library to support regular expression operations, rather than International Components for Unicode (ICU). The following discussion describes differences between the Spencer and ICU libraries that may affect applications:

• With the Spencer library, the [REGEXP](#_bookmark145) and [RLIKE](#_bookmark145) operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

ICU has full Unicode support and is multibyte safe. Its regular expression functions treat all strings as UTF-16. You should keep in mind that positional indexes are based on 16-bit chunks and not on code points. This means that, when passed to such functions, characters using more than one chunk may produce unanticipated results, such as those shown here:

mysql> **SELECT** **REGEXP\_INSTR('** **',** **'b');**

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

| REGEXP\_INSTR('??b', 'b') |

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

| 5 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **REGEXP\_INSTR('****xxx',** **'b',** **4);**

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

| REGEXP\_INSTR('??bxxx', 'b', 4) |

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

| 5 |

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

1 row in set (0.00 sec)

Characters within the Unicode Basic Multilingual Plane, which includes characters used by most modern languages, are safe in this regard:

mysql> **SELECT** **REGEXP\_INSTR('**6ж**b',** **'b');**

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

| REGEXP\_INSTR('6жb', 'b') |

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

| 3 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **REGEXP\_INSTR('**בע**b',** **'b');**

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

| REGEXP\_INSTR('בעb', 'b') |

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

| 3 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **REGEXP\_INSTR('µå**周**ç**6**',** **'**周**');**

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

| REGEXP\_INSTR('µå周ç6 ', '周 ') |

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

| 3 |

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

1 row in set (0.00 sec)

Emoji, such as the “sushi” character #(U+1F363) used in the first two examples, are not included in the Basic Multilingual Plane, but rather in Unicode's Supplementary Multilingual Plane. Another issue can arise with emoji and other 4-byte characters when [REGEXP\_SUBSTR()](#_bookmark149) or a similar function begins searching in the middle of a character. Each of the two statements in the following example starts from the second 2-byte position in the first argument. The first statement works on a string consisting solely of 2-byte (BMP) characters. The second statement contains 4-byte characters which are incorrectly interpreted in the result because the first two bytes are stripped off and so the remainder of the character data is misaligned.

mysql> **SELECT** **REGEXP\_SUBSTR('**周周周周 **',** **'** **.\*',** **2);**

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

| REGEXP\_SUBSTR('周周周周', '.\*', 2) |

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

| 周周周 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **REGEXP\_SUBSTR('** **,** **'** **.\*',** **2);**

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

| REGEXP\_SUBSTR('????', ' .\*', 2) |

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

| ?#揘#揘#揘 |

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

1 row in set (0.00 sec)

• For the . operator, the Spencer library matches line-terminator characters (carriage return, newline) anywhere in string expressions, including in the middle. To match line terminator characters in the middle of strings with ICU, specify the m match-control character.

• The Spencer library supports word-beginning and word-end boundary markers ( [[:<:]] and [[:>:]] notation). ICU does not. For ICU, you can use \b to match word boundaries; double the backslash because MySQL interprets it as the escape character within strings.

• The Spencer library supports collating element bracket expressions ( [.characters.] notation). ICU does not.

• For repetition counts ({n} and {m,n} notation), the Spencer library has a maximum of 255. ICU has no such limit, although the maximum number of match engine steps can be limited by setting the regexp\_time\_limit system variable.

• ICU interprets parentheses as metacharacters. To specify a literal open or close parenthesis ( in a regular expression, it must be escaped:

mysql> **SELECT** **REGEXP\_LIKE('(',** **'(');**

ERROR 3692 (HY000): Mismatched parenthesis in regular expression .

mysql> **SELECT** **REGEXP\_LIKE('(',** **'\\(');**

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

| REGEXP\_LIKE('(', '\\(') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_LIKE(')',** **')');**

ERROR 3692 (HY000): Mismatched parenthesis in regular expression .

mysql> **SELECT** **REGEXP\_LIKE(')',** **'\\)');**

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

| REGEXP\_LIKE(')', '\\)') |

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

| 1 |

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

• ICU also interprets square brackets as metacharacters, but only the opening square bracket need be escaped to be used as a literal character:

mysql> **SELECT** **REGEXP\_LIKE('[',** **'[');**

ERROR 3696 (HY000): The regular expression contains an

unclosed bracket expression .

mysql> **SELECT** **REGEXP\_LIKE('[',** **'\\[');**

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

| REGEXP\_LIKE('[', '\\[') |

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

| 1 |

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

mysql> **SELECT** **REGEXP\_LIKE(']',** **']');**

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

| REGEXP\_LIKE(']', ']') |

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

| 1 |

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

**12.8.3** **Character** **Set** **and** **Collation** **of** **Function** **Results**

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, [UPPER(*X*)](#_bookmark167) returns

a string with the same character string and collation as *X*. The same applies for [INSTR()](#_bookmark125), [LCASE()](#_bookmark126), [LOWER()](#_bookmark132), [LTRIM()](#_bookmark134), [MID()](#_bookmark137), [REPEAT()](#_bookmark150), [REPLACE()](#_bookmark151), [REVERSE()](#_bookmark152), [RIGHT()](#_bookmark153), [RPAD()](#_bookmark154), [RTRIM()](#_bookmark155), [SOUNDEX()](#_bookmark156), [SUBSTRING()](#_bookmark161), [TRIM()](#_bookmark164), [UCASE()](#_bookmark165), and [UPPER()](#_bookmark167).

**Note**

The [REPLACE()](#_bookmark151) function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has the binary character set and collation. This can be checked by using the [CHARSET()](#_bookmark175) and [COLLATION()](#_bookmark176) functions, both of which return binary for a binary string argument:

mysql> **SELECT** **CHARSET(BINARY** **'a'),** **COLLATION(BINARY** **'a');**

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

| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |

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

|  |  |  |
| --- | --- | --- |
| | binary | | binary | | |

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

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

• If an explicit COLLATE *Y* occurs, use *Y*.

• If explicit COLLATE *Y* and COLLATE *Z* occur, raise an error.

• Otherwise, if all collations are *Y*, use *Y*.

• Otherwise, the result has no collation.

For example, with CASE ... WHEN a THEN b WHEN b THEN c COLLATE *X* END, the resulting collation is *X*. The same applies for UNION, ||, [CONCAT()](#_bookmark117), [ELT()](#_bookmark119), GREATEST(), [IF()](#_bookmark1), and

LEAST().

For operations that convert to character data, the character set and collation of the strings

that result from the operations are defined by the character\_set\_connection and collation\_connection system variables that determine the default connection character set and collation (see Section 10.4, “Connection Character Sets and Collations” ). This applies only to BIN\_TO\_UUID(), [CAST()](#_bookmark109), [CONV()](#_bookmark17), [FORMAT()](#_bookmark46), [HEX()](#_bookmark48), and [SPACE()](#_bookmark158).

An exception to the preceding principle occurs for expressions for virtual generated columns. In such expressions, the table character set is used for BIN\_TO\_UUID(), [CONV()](#_bookmark17), or [HEX()](#_bookmark48) results, regardless of connection character set.

If there is any question about the character set or collation of the result returned by a string function, use the [CHARSET()](#_bookmark175) or [COLLATION()](#_bookmark176) function to find out:

mysql> **SELECT** **USER(),** **CHARSET(USER()),** **COLLATION(USER());**

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

| USER() | CHARSET(USER()) | COLLATION(USER()) |

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

| test@localhost | utf8mb3 | utf8mb3\_general\_ci |

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

mysql> **SELECT** **CHARSET(COMPRESS('abc')),** **COLLATION(COMPRESS('abc'));**

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

| CHARSET(COMPRESS('abc')) | COLLATION(COMPRESS('abc')) |

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

|  |  |  |
| --- | --- | --- |
| | binary | | binary | | |

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

**12.9** **What** **Calendar** **Is** **Used** **By** **MySQL?**

MySQL uses what is known as a *proleptic* *Gregorian* *calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 1 | 2 | 3 | 4 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are nonexistent.

A calendar applied to dates when it was not actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL DATE or DATETIME values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

**12.10** **Full-Text** **Search** **Functions**

[MATCH (*col1*,*col2*,...) AGAINST (*expr* [*search\_modifier*])](#_bookmark136)

|  |  |
| --- | --- |
| *search\_modifier:*  {  IN NATURAL  | IN NATURAL  | IN BOOLEAN  | WITH QUERY  } | LANGUAGE MODE  LANGUAGE MODE WITH QUERY EXPANSION  MODE  EXPANSION |

MySQL has support for full-text indexing and searching:

• A full-text index in MySQL is an index of type FULLTEXT.

• Full-text indexes can be used only with InnoDB or MyISAM tables, and can be created only for CHAR, VARCHAR, or TEXT columns.

• MySQL provides a built-in full-text ngram parser that supports Chinese, Japanese, and Korean

(CJK), and an installable MeCab full-text parser plugin for Japanese. Parsing differences are outlined in [Section 12.10.8, “ngram Full-Text Parser”](#_bookmark177) , and [Section 12.10.9, “MeCab Full-Text Parser Plugin”](#_bookmark178) .

• A FULLTEXT index definition can be given in the CREATE TABLE statement when a table is created, or added later using ALTER TABLE or CREATE INDEX.

• For large data sets, it is much faster to load your data into a table that has no FULLTEXT index and then create the index after that, than to load data into a table that has an existing FULLTEXT index.

Full-text searching is performed using [MATCH() AGAINST()](#_bookmark136) syntax. [MATCH()](#_bookmark136) takes a comma- separated list that names the columns to be searched. AGAINST takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.

Previously, MySQL permitted the use of a rollup column with MATCH(), but queries employing this construct performed poorly and with unreliable results. (This is due to the fact that MATCH() is not implemented as a function of its arguments, but rather as a function of the row ID of the current row in the underlying scan of the base table.) As of MySQL 8.0.28, MySQL no longer allows such queries; more specifically, any query matching all of the criteria listed here is rejected with [ER\_FULLTEXT\_WITH\_ROLLUP](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_fulltext_with_rollup):

• MATCH() appears in the SELECT list, GROUP BY clause, HAVING clause, or ORDER BY clause of a query block.

• The query block contains a GROUP BY ... WITH ROLLUP clause.

• The argument of the call to the MATCH() function is one of the grouping columns. Some examples of such queries are shown here:

# MATCH() in SELECT list...

SELECT MATCH (a) AGAINST ('abc') FROM t GROUP BY a WITH ROLLUP;

SELECT 1 FROM t GROUP BY a, MATCH (a) AGAINST ('abc') WITH ROLLUP;

# ...in HAVING clause...

SELECT 1 FROM t GROUP BY a WITH ROLLUP HAVING MATCH (a) AGAINST ('abc');

# ...and in ORDER BY clause

SELECT 1 FROM t GROUP BY a WITH ROLLUP ORDER BY MATCH (a) AGAINST ('abc');

The use of MATCH() with a rollup column in the WHERE clause is permitted.

There are three types of full-text searches:

• A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators, with the exception of double quote (") characters. The stopword list applies. For more information about stopword lists, see [Section 12.10.4, “Full-Text Stopwords”](#_bookmark179) .

Full-text searches are natural language searches if the IN NATURAL LANGUAGE MODE modifier is given or if no modifier is given. For more information, see [Section 12.10.1, “Natural Language Full-](#_bookmark180) [Text Searches”](#_bookmark180) .

• A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Certain common words (stopwords) are omitted from the search index and do not match if present in the search string. The IN BOOLEAN MODE modifier specifies a boolean search. For more information, see [Section 12.10.2, “Boolean Full-Text Searches”](#_bookmark181) .

• A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION or WITH QUERY EXPANSION modifier specifies a query expansion search. For more information, see [Section 12.10.3, “Full-Text Searches with Query Expansion”](#_bookmark182) .

For information about FULLTEXT query performance, see Section 8.3.5, “Column Indexes” .

For more information about InnoDB FULLTEXT indexes, see Section 15.6.2.4, “InnoDB Full-Text

Indexes” .

Constraints on full-text searching are listed in [Section 12.10.5, “Full-Text Restrictions”](#_bookmark183) .

The myisam\_ftdump utility dumps the contents of a MyISAM full-text index. This may be helpful for debugging full-text queries. See Section 4.6.3, “myisam\_ftdump — Display Full-Text Index information” .

**12.10.1** **Natural** **Language** **Full-Text** **Searches**

By default or with the IN NATURAL LANGUAGE MODE modifier, the [MATCH()](#_bookmark136) function performs a natural language search for a string against a *text* *collection*. A collection is a set of one or more columns included in a FULLTEXT index. The search string is given as the argument to AGAINST(). For each row in the table, [MATCH()](#_bookmark136) returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the [MATCH()](#_bookmark136) list.

mysql> **CREATE** **TABLE** **articles** **(**

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

-> **title** **VARCHAR(200),**

-> **body** **TEXT,**

-> **FULLTEXT** **(title,body)**

-> **)** **ENGINE=InnoDB;**

Query OK, 0 rows affected (0.08 sec)

mysql> **INSERT** **INTO** **articles** **(title,body)** **VALUES**

-> **('MySQL** **Tutorial','DBMS** **stands** **for** **DataBase** **.** **.** **.** **'),**

-> **('How** **To** **Use** **MySQL** **Well','After** **you** **went** **through** **a** **.** **.** **.** **'),**

-> **('Optimizing** **MySQL','In** **this** **tutorial,** **we** **show** **.** **.** **.** **'),**

-> **('1001** **MySQL** **Tricks','1** **.** **Never** **run** **mysqld** **as** **root** **.** **2.** **.** **.** **.** **'),**

-> **('MySQL** **vs** **.** **YourSQL','In** **the** **following** **database** **comparison** **.** **.** **.** **'),**

-> **('MySQL** **Security','When** **configured** **properly,** **MySQL** **.** **.** **.** **');**

Query OK, 6 rows affected (0.01 sec)

Records: 6 Duplicates: 0 Warnings: [0](#_bookmark184)

mysql> **SELECT** **\*** **FROM** **articles**

-> **WHERE** **MATCH** **(title,body)**

-> **AGAINST** **('database'** **IN** **NATURAL** **LANGUAGE** **MODE);**

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

| id | title | body |

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

| 1 | MySQL Tutorial | DBMS stands for DataBase ... |

| 5 | MySQL vs . YourSQL | In the following database comparison ... |

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

2 rows in set (0.00 sec)

By default, the search is performed in case-insensitive fashion. To perform a case-sensitive full-text search, use a case-sensitive or binary collation for the indexed columns. For example, a column that uses the utf8mb4 character set of can be assigned a collation of utf8mb4\_0900\_as\_cs or utf8mb4\_bin to make it case-sensitive for full-text searches.

When [MATCH()](#_bookmark136) is used in a WHERE clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first as long as the following conditions are met:

• There must be no explicit ORDER BY clause.

• The search must be performed using a full-text index scan rather than a table scan.

• If the query joins tables, the full-text index scan must be the leftmost non-constant table in the join.

Given the conditions just listed, it is usually less effort to specify using ORDER BY an explicit sort order when one is necessary or desired.

Relevance values are nonnegative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row (document), the number of unique words in the row, the total number of words in the collection, and the number of rows that contain a particular word.

**Note**

The term “document” may be used interchangeably with the term “row” , and both terms refer to the indexed part of the row. The term “collection” refers to the indexed columns and encompasses all rows.

To simply count matches, you could use a query like this:

mysql> **SELECT** **COUNT(\*)** **FROM** **articles**

-> **WHERE** **MATCH** **(title,body)**

-> **AGAINST** **('database'** **IN** **NATURAL** **LANGUAGE** **MODE);**

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

| COUNT(\*) |

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

| 2 |

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

1 row in set (0.00 sec)

You might find it quicker to rewrite the query as follows:

mysql> **SELECT**

-> **COUNT(IF(MATCH** **(title,body)** **AGAINST** **('database'** **IN** **NATURAL** **LANGUAGE** **MODE),** **1,** **NULL))**

-> **AS** **count**

-> **FROM** **articles;**

+-------+

| count |

+-------+

| 2 |

+-------+

1 row in set (0.03 sec)

The first query does some extra work (sorting the results by relevance) but also can use an index lookup based on the WHERE clause. The index lookup might make the first query faster if the search matches few rows. The second query performs a full table scan, which might be faster than the index lookup if the search term was present in most rows.

For natural-language full-text searches, the columns named in the [MATCH()](#_bookmark136) function must be the same columns included in some FULLTEXT index in your table. For the preceding query, note that the columns named in the [MATCH()](#_bookmark136) function (title and body) are the same as those named in the definition of the article table's FULLTEXT index. To search the title or body separately, you would create separate FULLTEXT indexes for each column.

You can also perform a boolean search or a search with query expansion. These search types are described in [Section 12.10.2, “Boolean Full-Text Searches”](#_bookmark181) , and [Section 12.10.3, “Full-Text Searches](#_bookmark182) [with Query Expansion”](#_bookmark182) .

A full-text search that uses an index can name columns only from a single table in the [MATCH()](#_bookmark136) clause because an index cannot span multiple tables. For MyISAM tables, a boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the [MATCH()](#_bookmark136) function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the SELECT statement includes neither WHERE nor ORDER BY clauses:

mysql> **SELECT** **id,** **MATCH** **(title,body)**

-> **AGAINST** **('Tutorial'** **IN** **NATURAL** **LANGUAGE** **MODE)** **AS** **score**

-> **FROM** **articles;**

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

|  |  |
| --- | --- |
| | id | score | | |

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

|

|

|

|

|

0.22764469683170319

0

0.22764469683170319

0

0

1

2

3

4

5

|

|

|

|

|

|

|

|

|

|



| 6 | 0 |

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

6 rows in set (0.00 sec)

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, specify [MATCH()](#_bookmark136) twice: once in the SELECT list and once in the WHERE clause. This causes no additional overhead, because the MySQL optimizer notices that the two [MATCH()](#_bookmark136) calls are identical and invokes the full-text search code only once.

mysql> **SELECT** **id,** **body,** **MATCH** **(title,body)**

-> **AGAINST** **('Security** **implications** **of** **running** **MySQL** **as** **root'**

-> **IN** **NATURAL** **LANGUAGE** **MODE)** **AS** **score**

-> **FROM** **articles**

-> **WHERE** **MATCH** **(title,body)**

-> **AGAINST('Security** **implications** **of** **running** **MySQL** **as** **root'**

-> **IN** **NATURAL** **LANGUAGE** **MODE);**

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

| id | body | score |

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

| 4 | 1. Never run mysqld as root . 2. ... | 1.5219271183014 |

| 6 | When configured properly, MySQL ... | 1.3114095926285 |

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

2 rows in set (0.00 sec)

A phrase that is enclosed within double quote ( ") characters matches only rows that contain the phrase

*literally,* *as* *it* *was* *typed*. The full-text engine splits the phrase into words and performs a search in the FULLTEXT index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, "test phrase" matches "test, phrase". If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The MySQL FULLTEXT implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes ( '), but not more than one in a row. This means that aaa'bbb is regarded as one word, but aaa''bbb is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the FULLTEXT parser; 'aaa'bbb' would be parsed as aaa'bbb.

The built-in FULLTEXT parser determines where words start and end by looking for certain delimiter characters; for example, (space), , (comma), and . (period). If words are not separated by delimiters (as in, for example, Chinese), the built-in FULLTEXT parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a FULLTEXT index that uses the built-in FULLTEXT parser, you must preprocess them so that they are separated by some arbitrary delimiter. Alternatively, you can create FULLTEXT indexes using the ngram parser plugin (for Chinese, Japanese, or Korean) or the MeCab parser plugin (for Japanese).

It is possible to write a plugin that replaces the built-in full-text parser. For details, see [The MySQL](https://dev.mysql.com/doc/extending-mysql/8.0/en/plugin-api.html) [Plugin API](https://dev.mysql.com/doc/extending-mysql/8.0/en/plugin-api.html). For example parser plugin source code, see the plugin/fulltext directory of a MySQL source distribution.

Some words are ignored in full-text searches:

• Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is three characters for InnoDB search indexes, or four characters for MyISAM. You can control the cutoff by setting a configuration option before creating the index: innodb\_ft\_min\_token\_size configuration option for InnoDB search indexes, or ft\_min\_word\_len for MyISAM.

**Note**

This behavior does not apply to FULLTEXT indexes that use the ngram parser. For the ngram parser, token length is defined by the ngram\_token\_size option.

• Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overridden by a user-defined list. The stopword lists and related configuration options are different for InnoDB search indexes and MyISAM ones. Stopword processing is controlled by the configuration options innodb\_ft\_enable\_stopword, innodb\_ft\_server\_stopword\_table, and innodb\_ft\_user\_stopword\_table for InnoDB search indexes, and ft\_stopword\_file for MyISAM ones.

See [Section 12.10.4, “Full-Text Stopwords”](#_bookmark179) to view default stopword lists and how to change them. The default minimum word length can be changed as described in [Section 12.10.6, “Fine-Tuning MySQL](#_bookmark185) [Full-Text Search”](#_bookmark185) .

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Thus, a word that is present in many documents has a lower weight, because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row. This technique works best with large collections.

**MyISAM** **Limitation**

For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results for search indexes on MyISAM tables. For example, although the word “MySQL” is present in every row of the articles table shown earlier, a search for the word in a MyISAM search index produces no results:

mysql> **SELECT** **\*** **FROM** **articles**

-> **WHERE** **MATCH** **(title,body)**

-> **AGAINST** **('MySQL'** **IN** **NATURAL** **LANGUAGE** **MODE);**

Empty set (0.00 sec)

The search result is empty because the word “MySQL” is present in at least 50% of the rows, and so is effectively treated as a stopword. This filtering technique is more suitable for large data sets, where you might not want the result set to return every second row from a 1GB table, than for small data sets where it might cause poor results for popular terms.

The 50% threshold can surprise you when you first try full-text searching to see how it works, and makes InnoDB tables more suited to experimentation with full-text searches. If you create a MyISAM table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results until the table contains more rows. Users who need to bypass the 50% limitation can build search indexes on InnoDB tables, or use the boolean search mode explained in [Section 12.10.2, “Boolean](#_bookmark181) [Full-Text Searches”](#_bookmark181) .

**12.10.2** **Boolean** **Full-Text** **Searches**

MySQL can perform boolean full-text searches using the IN BOOLEAN MODE modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string. In the following query, the + and - operators indicate that a word must be present or absent, respectively, for a match to occur. Thus, the query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL” :

mysql> **SELECT** **\*** **FROM** **articles** **WHERE** **MATCH** **(title,body)**

-> **AGAINST** **('+MySQL** **-YourSQL'** **IN** **BOOLEAN** **MODE);**

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

|  |  |  |
| --- | --- | --- |
| | id | title | | body | | |

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

|  |  |  |
| --- | --- | --- |
| |  |  | | 1 | MySQL Tutorial | DBMS stands for DataBase ...  2 | How To Use MySQL Well | After you went through a ...  3 | Optimizing MySQL | In this tutorial, we show ... | |  |  | |



| 4 | 1001 MySQL Tricks | 1. Never run mysqld as root . 2. ... |

| 6 | MySQL Security | When configured properly, MySQL ... |

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

**Note**

In implementing this feature, MySQL uses what is sometimes referred to as *implied* *Boolean* *logic*, in which

• + stands for AND

• - stands for NOT

• [*no* *operator*] implies OR

Boolean full-text searches have these characteristics:

• They do not automatically sort rows in order of decreasing relevance.

• InnoDB tables require a FULLTEXT index on all columns of the [MATCH()](#_bookmark136) expression to perform boolean queries. Boolean queries against a MyISAM search index can work even without a FULLTEXT index, although a search executed in this fashion would be quite slow.

• The minimum and maximum word length full-text parameters apply to FULLTEXT indexes created using the built-in FULLTEXT parser and MeCab parser plugin. innodb\_ft\_min\_token\_size and innodb\_ft\_max\_token\_size are used for InnoDB search indexes. ft\_min\_word\_len and

ft\_max\_word\_len are used for MyISAM search indexes.

Minimum and maximum word length full-text parameters do not apply to FULLTEXT indexes created using the ngram parser. ngram token size is defined by the ngram\_token\_size option.

• The stopword list applies, controlled by innodb\_ft\_enable\_stopword, innodb\_ft\_server\_stopword\_table, and innodb\_ft\_user\_stopword\_table for InnoDB search indexes, and ft\_stopword\_file for MyISAM ones.

• InnoDB full-text search does not support the use of multiple operators on a single search word, as in this example: '++apple'. Use of multiple operators on a single search word returns a syntax error to standard out. MyISAM full-text search successfully processes the same search, ignoring all operators except for the operator immediately adjacent to the search word.

• InnoDB full-text search only supports leading plus or minus signs. For example, InnoDB supports '+apple' but does not support 'apple+'. Specifying a trailing plus or minus sign causes InnoDB to report a syntax error.

• InnoDB full-text search does not support the use of a leading plus sign with wildcard ('+\*'), a plus and minus sign combination ('+-'), or leading a plus and minus sign combination ('+-apple'). These invalid queries return a syntax error.

• InnoDB full-text search does not support the use of the @ symbol in boolean full-text searches. The @ symbol is reserved for use by the @distance proximity search operator.

• They do not use the 50% threshold that applies to MyISAM search indexes. The boolean full-text search capability supports the following operators:

• +

A leading or trailing plus sign indicates that this word *must* be present in each row that is returned. InnoDB only supports leading plus signs.

• -

A leading or trailing minus sign indicates that this word must *not* be present in any of the rows that are returned. InnoDB only supports leading minus signs.

Note: The - operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by - returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

• (no operator)

By default (when neither + nor - is specified), the word is optional, but the rows that contain it are rated higher. This mimics the behavior of [MATCH() AGAINST()](#_bookmark136) without the IN BOOLEAN MODE modifier.

• @*distance*

This operator works on InnoDB tables only. It tests whether two or more words all start within a specified distance from each other, measured in words. Specify the search words within a double-quoted string immediately before the @*distance* operator, for example, MATCH(col1)

AGAINST('"word1 word2 word3" @8' IN BOOLEAN MODE)

• > <

These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The > operator increases the contribution and the < operator decreases it. See the example following this list.

• ( )

Parentheses group words into subexpressions. Parenthesized groups can be nested.

• ~

A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the - operator.

• \*

The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it is *appended* to the word to be affected. Words match if they begin with the word preceding the \* operator.

If a word is specified with the truncation operator, it is not stripped from a boolean query, even if it is too short or a stopword. Whether a word is too short is determined from the innodb\_ft\_min\_token\_size setting for InnoDB tables, or ft\_min\_word\_len for MyISAM tables. These options are not applicable to FULLTEXT indexes that use the ngram parser.

The wildcarded word is considered as a prefix that must be present at the start of one or more words. If the minimum word length is 4, a search for '+*word* +the\*' could return fewer rows than a search for '+*word* +the', because the second query ignores the too-short search term the.

• "

A phrase that is enclosed within double quote ( ") characters matches only rows that contain the phrase *literally,* *as* *it* *was* *typed*. The full-text engine splits the phrase into words and performs a search in the FULLTEXT index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, "test phrase" matches "test, phrase".

If the phrase contains no words that are in the index, the result is empty. The words might not be in the index because of a combination of factors: if they do not exist in the text, are stopwords, or are shorter than the minimum length of indexed words.

The following examples demonstrate some search strings that use boolean full-text operators:

• 'apple banana'

Find rows that contain at least one of the two words.

• '+apple +juice'

Find rows that contain both words.

• '+apple macintosh'

Find rows that contain the word “apple” , but rank rows higher if they also contain “macintosh” .

• '+apple -macintosh'

Find rows that contain the word “apple” but not “macintosh” .

• '+apple ~macintosh'

Find rows that contain the word “apple” , but if the row also contains the word “macintosh” , rate it lower than if row does not. This is “softer” than a search for '+apple -macintosh', for which the presence of “macintosh” causes the row not to be returned at all.

• '+apple +(>turnover <strudel)'

Find rows that contain the words “apple” and “turnover” , or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel” .

• 'apple\*'

Find rows that contain words such as “apple” , “apples” , “applesauce” , or “applet” .

• '"some words"'

Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the " characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotation marks that enclose the search string itself.

**Relevancy** **Rankings** **for** **InnoDB** **Boolean** **Mode** **Search**

InnoDB full-text search is modeled on the [Sphinx](http://sphinxsearch.com/) full-text search engine, and the algorithms used are based on [BM25](http://en.wikipedia.org/wiki/Okapi_BM25) and [TF-IDF](http://en.wikipedia.org/wiki/TF-IDF) ranking algorithms. For these reasons, relevancy rankings for InnoDB boolean full-text search may differ from MyISAM relevancy rankings.

InnoDB uses a variation of the “term frequency-inverse document frequency” (TF-IDF) weighting system to rank a document's relevance for a given full-text search query. The TF-IDF weighting is based on how frequently a word appears in a document, offset by how frequently the word appears in all documents in the collection. In other words, the more frequently a word appears in a document, and the less frequently the word appears in the document collection, the higher the document is ranked.

**How** **Relevancy** **Ranking** **is** **Calculated**

The term frequency (TF) value is the number of times that a word appears in a document. The inverse document frequency ( IDF) value of a word is calculated using the following formula, where total\_records is the number of records in the collection, and matching\_records is the number of records that the search term appears in.

${IDF} = log10( ${total\_records} / ${matching\_records} )

When a document contains a word multiple times, the IDF value is multiplied by the TF value:

${TF} \* ${IDF}

Using the TF and IDF values, the relevancy ranking for a document is calculated using this formula:

-> **MATCH** **(title,body)** **AGAINST** **('database'** **IN** **BOOLEAN** **MODE)** **AS** **score**

-> **FROM** **articles** **ORDER** **BY** **score** **DESC;**

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

| id | title | body | score |

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

| 6 | Database, Database, Database | database database database | 1.0886961221694946 |

| 3 | Optimizing Your Database | In this database tutorial | 0.36289870738983154 |

| 1 | MySQL Tutorial | This database tutorial | 0.18144935369491577 |

| 2 | How To Use MySQL | After you went through a | 0 |

| 4 | MySQL vs . YourSQL | When comparing databases | 0 |

| 5 | MySQL Security | When configured properly, MySQL | 0 |

| 7 | 1001 MySQL Tricks | 1. Never run mysqld as root . 2 | 0 |

| 8 | MySQL Full-Text Indexes | MySQL fulltext indexes use a | 0 |

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

8 rows in set (0.00 sec)

${rank} = ${TF} \* ${IDF} \* ${IDF}

The formula is demonstrated in the following examples.

**Relevancy** **Ranking** **for** **a** **Single** **Word** **Search**

This example demonstrates the relevancy ranking calculation for a single-word search.

mysql> **CREATE** **TABLE** **articles** **(**

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

-> **title** **VARCHAR(200),**

-> **body** **TEXT,**

-> **FULLTEXT** **(title,body)**

->) **ENGINE=InnoDB;**

Query OK, 0 rows affected (1.04 sec)

mysql> **INSERT** **INTO** **articles** **(title,body)** **VALUES**

-> **('MySQL** **Tutorial','This** **database** **tutorial** **.** **.** **.** **'),**

-> **("How** **To** **Use** **MySQL",'After** **you** **went** **through** **a** **.** **.** **.** **'),**

-> **('Optimizing** **Your** **Database','In** **this** **database** **tutorial** **.** **.** **.** **'),**

-> **('MySQL** **vs** **.** **YourSQL','When** **comparing** **databases** **.** **.** **.** **'),**

-> **('MySQL** **Security','When** **configured** **properly,** **MySQL** **.** **.** **.** **'),**

-> **('Database,** **Database,** **Database','database** **database** **database'),**

-> **('1001** **MySQL** **Tricks','1** **.** **Never** **run** **mysqld** **as** **root** **.** **2.** **.** **.** **.** **'),**

-> **('MySQL** **Full-Text** **Indexes',** **'MySQL** **fulltext** **indexes** **use** **a** **.** **.** **');**

Query OK, 8 rows affected (0.06 sec)

Records: 8 Duplicates: 0 Warnings: [0](#_bookmark186)

mysql> **SELECT** **id,** **title,** **body,**

There are 8 records in total, with 3 that match the “database” search term. The first record (id 6) contains the search term 6 times and has a relevancy ranking of 1.0886961221694946. This ranking value is calculated using a TF value of 6 (the “database” search term appears 6 times in record id 6) and an IDF value of 0.42596873216370745, which is calculated as follows (where 8 is the total number of records and 3 is the number of records that the search term appears in):

${IDF} = LOG10( 8 / 3 ) = 0.42596873216370745

The TF and IDF values are then entered into the ranking formula:

${rank} = ${TF} \* ${IDF} \* ${IDF}

Performing the calculation in the MySQL command-line client returns a ranking value of

1.088696164686938.

mysql> **SELECT** **6\*LOG10(8/3)\*LOG10(8/3);**

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

| 6\*LOG10(8/3)\*LOG10(8/3) |

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

| 1.088696164686938 |

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

1 row in set (0.00 sec)



**Note**

You may notice a slight difference in the ranking values returned by the SELECT ... MATCH ... AGAINST statement and the MySQL command-line client (1.0886961221694946 versus 1.088696164686938). The difference is due to how the casts between integers and floats/doubles are performed internally by InnoDB (along with related precision and rounding decisions), and how they are performed elsewhere, such as in the MySQL command-line client or other types of calculators.

**Relevancy** **Ranking** **for** **a** **Multiple** **Word** **Search**

This example demonstrates the relevancy ranking calculation for a multiple-word full-text search based on the articles table and data used in the previous example.

If you search on more than one word, the relevancy ranking value is a sum of the relevancy ranking value for each word, as shown in this formula:

${rank} = ${TF} \* ${IDF} \* ${IDF} + ${TF} \* ${IDF} \* ${IDF}

Performing a search on two terms ('mysql tutorial') returns the following results:

mysql> **SELECT** **id,** **title,** **body,** **MATCH** **(title,body)**

-> **AGAINST** **('mysql** **tutorial'** **IN** **BOOLEAN** **MODE)** **AS** **score**

-> FROM articles ORDER BY score DESC;

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

|

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

|

|

|

|

|

|

|

|

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

8 rows in set (0.00 sec)

This database tutorial ...

In this database tutorial ...

When configured properly, MySQL

MySQL fulltext indexes use a ..

After you went through a ...

When comparing databases ...

1. Never run mysqld as root . 2.

database database database

MySQL Tutorial

Optimizing Your Database

MySQL Security

MySQL Full-Text Indexes

How To Use MySQL

MySQL vs. YourSQL

1001 MySQL Tricks

Database, Database, Database

0.7405621409416199

0.3624762296676636

0.031219376251101494

0.031219376251101494

0.015609688125550747

0.015609688125550747

0.015609688125550747

0

| id | title

1

3

5

8

2

4

7

6

| body

score

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

...

...

In the first record (id 8), 'mysql' appears once and 'tutorial' appears twice. There are six matching records for 'mysql' and two matching records for 'tutorial'. The MySQL command-line client returns the expected ranking value when inserting these values into the ranking formula for a multiple word search:

mysql> SELECT (1\*log10(8/6)\*log10(8/6)) + (2\*log10(8/2)\*log10(8/2));

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

| (1\*log10(8/6)\*log10(8/6)) + (2\*log10(8/2)\*log10(8/2)) |

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

| 0.7405621541938003 |

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

1 row in set (0.00 sec)

**Note**

The slight difference in the ranking values returned by the SELECT ... MATCH ... AGAINST statement and the MySQL command-line client is explained in the preceding example.

**12.10.3** **Full-Text** **Searches** **with** **Query** **Expansion**

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL” , “Oracle” , “DB2” , and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding WITH QUERY EXPANSION or IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL” , the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database” . The following example shows this difference:

mysql> **SELECT** **\*** **FROM** **articles**

**WHERE** **MATCH** **(title,body)**

**AGAINST** **('database'** **IN** **NATURAL** **LANGUAGE** **MODE);**

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

| id | title | body |

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

| 1 | MySQL Tutorial | DBMS stands for DataBase ... |

| 5 | MySQL vs . YourSQL | In the following database comparison ... |

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

2 rows in set (0.00 sec)

mysql> **SELECT** **\*** **FROM** **articles**

**WHERE** **MATCH** **(title,body)**

**AGAINST** **('database'** **WITH** **QUERY** **EXPANSION);**

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

| id | title | body |

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

| 5 | MySQL vs . YourSQL | In the following database comparison ... |

| 1 | MySQL Tutorial | DBMS stands for DataBase ... |

| 3 | Optimizing MySQL | In this tutorial we show ... |

| 6 | MySQL Security | When configured properly, MySQL ... |

| 2 | How To Use MySQL Well | After you went through a ... |

| 4 | 1001 MySQL Tricks | 1. Never run mysqld as root . 2. ... |

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

6 rows in set (0.00 sec)

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret” . A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

**Note**

Because blind query expansion tends to increase noise significantly by returning nonrelevant documents, use it only when a search phrase is short.

**12.10.4** **Full-Text** **Stopwords**

The stopword list is loaded and searched for full-text queries using the server character set and collation (the values of the character\_set\_server and collation\_server system variables). False hits or misses might occur for stopword lookups if the stopword file or columns used for full-text indexing or searches have a character set or collation different from character\_set\_server or collation\_server.

Case sensitivity of stopword lookups depends on the server collation. For example, lookups are case-insensitive if the collation is utf8mb4\_0900\_ai\_ci, whereas lookups are case-sensitive if the collation is utf8mb4\_0900\_as\_cs or utf8mb4\_bin.

• [Stopwords for InnoDB Search Indexes](#_bookmark187)

• [Stopwords for MyISAM Search Indexes](#_bookmark188)

**Stopwords** **for** **InnoDB** **Search** **Indexes**

InnoDB has a relatively short list of default stopwords, because documents from technical, literary, and other sources often use short words as keywords or in significant phrases. For example, you might

search for “to be or not to be” and expect to get a sensible result, rather than having all those words ignored.

To see the default InnoDB stopword list, query the Information Schema

INNODB\_FT\_DEFAULT\_STOPWORD table.

mysql> SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_FT\_DEFAULT\_STOPWORD;

+-------+

| value |

+-------+

| a |

| about |

| an |

| are |

| as |

| at |

| be |

| by |

| com |

| de |

| en |

| for |

| from |

| how |

| i |

| in |

| is |

| it |

| la |

| of |

| on |

| or |

| that |

| the |

| this |

| to |

| was |

| what |

| when |

| where |

| who |

| will |

| with |

| und |

| the |

| www |

+-------+

36 rows in set (0.00 sec)

To define your own stopword list for all InnoDB tables, define a table with the same structure as the INNODB\_FT\_DEFAULT\_STOPWORD table, populate it with stopwords, and set the value of the innodb\_ft\_server\_stopword\_table option to a value in the form *db\_name*/*table\_name* before creating the full-text index. The stopword table must have a single VARCHAR column named value. The following example demonstrates creating and configuring a new global stopword table for InnoDB.

-- Create a new stopword table

mysql> CREATE TABLE my\_stopwords(value VARCHAR(30)) ENGINE = INNODB;

Query OK, 0 rows affected (0.01 sec)

-- Insert stopwords (for simplicity, a single stopword is used in this example)

mysql> INSERT INTO my\_stopwords(value) VALUES ('Ishmael');

Query OK, 1 row affected (0.00 sec)

-- Create the table

mysql> CREATE TABLE opening\_lines (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

opening\_line TEXT(500),

author VARCHAR(200),



title VARCHAR(200)

) ENGINE=InnoDB;

Query OK, 0 rows affected (0.01 sec)

-- Insert data into the table

mysql> INSERT INTO opening\_lines(opening\_line,author,title) VALUES

('Call me Ishmael . ','Herman Melville','Moby-Dick'),

('A screaming comes across the sky . ','Thomas Pynchon','Gravity\'s Rainbow'),

('I am an invisible man . ','Ralph Ellison','Invisible Man'),

('Where now? Who now? When now?','Samuel Beckett','The Unnamable'),

('It was love at first sight . ','Joseph Heller','Catch-22'),

('All this happened, more or less . ','Kurt Vonnegut','Slaughterhouse-Five'),

('Mrs . Dalloway said she would buy the flowers herself . ','Virginia Woolf','Mrs . Dalloway'),

('It was a pleasure to burn . ','Ray Bradbury','Fahrenheit 451');

Query OK, 8 rows affected (0.00 sec)

Records: 8 Duplicates: 0 Warnings: [0](#_bookmark189)

-- Set the innodb\_ft\_server\_stopword\_table option to the new stopword table

mysql> SET GLOBAL innodb\_ft\_server\_stopword\_table = 'test/my\_stopwords';

Query OK, 0 rows affected (0.00 sec)

-- Create the full-text index (which rebuilds the table if no FTS\_DOC\_ID column is defined)

mysql> CREATE FULLTEXT INDEX idx ON opening\_lines(opening\_line);

Query OK, 0 rows affected, 1 warning (1.17 sec)

Records: 0 Duplicates: 0 Warnings: 1

Verify that the specified stopword ('Ishmael') does not appear by querying the Information Schema INNODB\_FT\_INDEX\_TABLE table.

**Note**

By default, words less than 3 characters in length or greater than 84 characters in length do not appear in an InnoDB full-text search index. Maximum and minimum word length values are configurable using the innodb\_ft\_max\_token\_size and innodb\_ft\_min\_token\_size variables. This default behavior does not apply to the ngram parser plugin. ngram token size is defined by the ngram\_token\_size option.

mysql> SET GLOBAL innodb\_ft\_aux\_table='test/opening\_lines';

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT word FROM INFORMATION\_SCHEMA.INNODB\_FT\_INDEX\_TABLE LIMIT 15;

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

| word |

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

| across |

| all |

| burn |

| buy |

| call |

| comes |

| dalloway |

| first |

| flowers |

| happened |

| herself |

| invisible |

| less |

| love |

| man |

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

15 rows in set (0.00 sec)

To create stopword lists on a table-by-table basis, create other stopword tables and use the innodb\_ft\_user\_stopword\_table option to specify the stopword table that you want to use before you create the full-text index.

**Stopwords** **for** **MyISAM** **Search** **Indexes**

The stopword file is loaded and searched using latin1 if character\_set\_server is ucs2, utf16, utf16le, or utf32.

To override the default stopword list for MyISAM tables, set the ft\_stopword\_file system variable. (See Section 5.1.8, “Server System Variables” .) The variable value should be the path name of the file containing the stopword list, or the empty string to disable stopword filtering. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your FULLTEXT indexes.

The stopword list is free-form, separating stopwords with any nonalphanumeric character such as newline, space, or comma. Exceptions are the underscore character (\_) and a single apostrophe ( ') which are treated as part of a word. The character set of the stopword list is the server's default character set; see Section 10.3.2, “Server Character Set and Collation” .

The following list shows the default stopwords for MyISAM search indexes. In a MySQL source distribution, you can find this list in the storage/myisam/ft\_static.c file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a's | able | about | above | according |
| accordingly | across | actually | after | afterwards |
| again | against | ain't | all | allow |
| allows | almost | alone | along | already |
| also | although | always | am | among |
| amongst | an | and | another | any |
| anybody | anyhow | anyone | anything | anyway |
| anyways | anywhere | apart | appear | appreciate |
| appropriate | are | aren't | around | as |
| aside | ask | asking | associated | at |
| available | away | awfully | be | became |
| because | become | becomes | becoming | been |
| before | beforehand | behind | being | believe |
| below | beside | besides | best | better |
| between | beyond | both | brief | but |
| by | c'mon | c's | came | can |
| can't | cannot | cant | cause | causes |
| certain | certainly | changes | clearly | co |
| com | come | comes | concerning | consequently |
| consider | considering | contain | containing | contains |
| corresponding | could | couldn't | course | currently |
| definitely | described | despite | did | didn't |
| different | do | does | doesn't | doing |
| don't | done | down | downwards | during |
| each | edu | eg | eight | either |
| else | elsewhere | enough | entirely | especially |
| et | etc | even | ever | every |
| everybody | everyone | everything | everywhere | ex |
| exactly | example | except | far | few |
| fifth | first | five | followed | following |
| follows | for | former | formerly | forth |
| four | from | further | furthermore | get |
| gets | getting | given | gives | go |
| goes | going | gone | got | gotten |
| greetings | had | hadn't | happens | hardly |
| has | hasn't | have | haven't | having |
| he | he's | hello | help | hence |
| her | here | here's | hereafter | hereby |
| herein | hereupon | hers | herself | hi |
| him | himself | his | hither | hopefully |
| how | howbeit | however | i'd | i'll |
| i'm | i've | ie | if | ignored |
| immediate | in | inasmuch | inc | indeed |
| indicate | indicated | indicates | inner | insofar |
| instead | into | inward | is | isn't |
| it | it'd | it'll | it's | its |
| itself | just | keep | keeps | kept |
| know | known | knows | last | lately |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| later | latter | latterly | least | less |
| lest | let | let's | like | liked |
| likely | little | look | looking | looks |
| ltd | mainly | many | may | maybe |
| me | mean | meanwhile | merely | might |
| more | moreover | most | mostly | much |
| must | my | myself | name | namely |
| nd | near | nearly | necessary | need |
| needs | neither | never | nevertheless | new |
| next | nine | no | nobody | non |
| none | noone | nor | normally | not |
| nothing | novel | now | nowhere | obviously |
| of | off | often | oh | ok |
| okay | old | on | once | one |
| ones | only | onto | or | other |
| others | otherwise | ought | our | ours |
| ourselves | out | outside | over | overall |
| own | particular | particularly | per | perhaps |
| placed | please | plus | possible | presumably |
| probably | provides | que | quite | qv |
| rather | rd | re | really | reasonably |
| regarding | regardless | regards | relatively | respectively |
| right | said | same | saw | say |
| saying | says | second | secondly | see |
| seeing | seem | seemed | seeming | seems |
| seen | self | selves | sensible | sent |
| serious | seriously | seven | several | shall |
| she | should | shouldn't | since | six |
| so | some | somebody | somehow | someone |
| something | sometime | sometimes | somewhat | somewhere |
| soon | sorry | specified | specify | specifying |
| still | sub | such | sup | sure |
| t's | take | taken | tell | tends |
| th | than | thank | thanks | thanx |
| that | that's | thats | the | their |
| theirs | them | themselves | then | thence |
| there | there's | thereafter | thereby | therefore |
| therein | theres | thereupon | these | they |
| they'd | they'll | they're | they've | think |
| third | this | thorough | thoroughly | those |
| though | three | through | throughout | thru |
| thus | to | together | too | took |
| toward | towards | tried | tries | truly |
| try | trying | twice | two | un |
| under | unfortunately | unless | unlikely | until |
| unto | up | upon | us | use |
| used | useful | uses | using | usually |
| value | various | very | via | viz |
| vs | want | wants | was | wasn't |
| way | we | we'd | we'll | we're |
| we've | welcome | well | went | were |
| weren't | what | what's | whatever | when |
| whence | whenever | where | where's | whereafter |
| whereas | whereby | wherein | whereupon | wherever |
| whether | which | while | whither | who |
| who's | whoever | whole | whom | whose |
| why | will | willing | wish | with |
| within | without | won't | wonder | would |
| wouldn't | yes | yet | you | you'd |
| you'll | you're | you've | your | yours |
| yourself | yourselves | zero |  |  |

**12.10.5** **Full-Text** **Restrictions**

• Full-text searches are supported for InnoDB and MyISAM tables only.

• Full-text searches are not supported for partitioned tables. See Section 24.6, “Restrictions and Limitations on Partitioning” .

• Full-text searches can be used with most multibyte character sets. The exception is that for Unicode, the utf8mb3 or utf8mb4 character set can be used, but not the ucs2 character set. Although

FULLTEXT indexes on ucs2 columns cannot be used, you can perform IN BOOLEAN MODE searches on a ucs2 column that has no such index.

The remarks for utf8mb3 also apply to utf8mb4, and the remarks for ucs2 also apply to utf16, utf16le, and utf32.

• Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the built-in full-text parser *cannot* *determine* *where* *words* *begin* *and* *end* *in* *these* *and* *other* *such* *languages*.

A character-based ngram full-text parser that supports Chinese, Japanese, and Korean (CJK), and a word-based MeCab parser plugin that supports Japanese are provided for use with InnoDB and MyISAM tables.

• Although the use of multiple character sets within a single table is supported, all columns in a FULLTEXT index must use the same character set and collation.

• The [MATCH()](#_bookmark136) column list must match exactly the column list in some FULLTEXT index definition for the table, unless this [MATCH()](#_bookmark136) is IN BOOLEAN MODE on a MyISAM table. For MyISAM tables, boolean-mode searches can be done on nonindexed columns, although they are likely to be slow.

• The argument to AGAINST() must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.

As of MySQL 8.0.28, the argument to [MATCH()](#_bookmark136) cannot use a rollup column.

• Index hints are more limited for FULLTEXT searches than for non-FULLTEXT searches. See Section 8.9.4, “Index Hints” .

• For InnoDB, all DML operations ( INSERT, UPDATE, DELETE) involving columns with full-text indexes are processed at transaction commit time. For example, for an INSERT operation, an inserted string is tokenized and decomposed into individual words. The individual words are then added to full-text index tables when the transaction is committed. As a result, full-text searches only return committed data.

• The '%' character is not a supported wildcard character for full-text searches.

**12.10.6** **Fine-Tuning** **MySQL** **Full-Text** **Search**

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See Section 2.8, “Installing MySQL from Source” .

Full-text search is carefully tuned for effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do* *not* *alter* *the* *MySQL* *sources* *unless* *you* *know* *what* *you* *are* *doing*.

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the FULLTEXT indexes in your tables. Instructions for doing so are given later in this section.

• [Configuring Minimum and Maximum Word Length](#_bookmark190)

• [Configuring the Natural Language Search Threshold](#_bookmark191)

• [Modifying Boolean Full-Text Search Operators](#_bookmark192)

• [Character Set Modifications](#_bookmark193)

• [Rebuilding InnoDB Full-Text Indexes](#_bookmark194)

• [Optimizing InnoDB Full-Text Indexes](#_bookmark195)

• [Rebuilding MyISAM Full-Text Indexes](#_bookmark196)

**Configuring** **Minimum** **and** **Maximum** **Word** **Length**

The minimum and maximum lengths of words to be indexed are defined by the innodb\_ft\_min\_token\_size and innodb\_ft\_max\_token\_size for InnoDB search indexes, and ft\_min\_word\_len and ft\_max\_word\_len for MyISAM ones.

**Note**

Minimum and maximum word length full-text parameters do not apply to FULLTEXT indexes created using the ngram parser. ngram token size is defined by the ngram\_token\_size option.

After changing any of these options, rebuild your FULLTEXT indexes for the change to take effect. For example, to make two-character words searchable, you could put the following lines in an option file:

[mysqld]

innodb\_ft\_min\_token\_size=2

ft\_min\_word\_len=2

Then restart the server and rebuild your FULLTEXT indexes. For MyISAM tables, note the remarks regarding myisamchk in the instructions that follow for rebuilding MyISAM full-text indexes.

**Configuring** **the** **Natural** **Language** **Search** **Threshold**

For MyISAM search indexes, the 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in storage/myisam/ ftdefs.h:

#define GWS\_IN\_USE GWS\_PROB

Change that line to this:

#define GWS\_IN\_USE GWS\_FREQ

Then recompile MySQL. There is no need to rebuild the indexes in this case.

**Note**

By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the [MATCH()](#_bookmark136) function. If you really need to search for such common words, it would be better to search using IN BOOLEAN MODE instead, which does not observe the 50% threshold.

**Modifying** **Boolean** **Full-Text** **Search** **Operators**

To change the operators used for boolean full-text searches on MyISAM tables, set the ft\_boolean\_syntax system variable. ( InnoDB does not have an equivalent setting.) This variable can be changed while the server is running, but you must have privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”). No rebuilding of indexes is necessary in this case.

**Character** **Set** **Modifications**

For the built-in full-text parser, you can change the set of characters that are considered word characters in several ways, as described in the following list. After making the modification, rebuild the indexes for each table that contains any FULLTEXT indexes. Suppose that you want to treat the hyphen character ('-') as a word character. Use one of these methods:

• Modify the MySQL source: In storage/innobase/handler/ha\_innodb.cc (for InnoDB), or in storage/myisam/ftdefs.h (for MyISAM), see the true\_word\_char() and misc\_word\_char() macros. Add '-' to one of those macros and recompile MySQL.

• Modify a character set file: This requires no recompilation. The true\_word\_char() macro uses a “character type” table to distinguish letters and numbers from other characters. . You can edit the contents of the <ctype><map> array in one of the character set XML files to specify that '-' is a “letter.” Then use the given character set for your FULLTEXT indexes. For information about the <ctype><map> array format, see Section 10.13.1, “Character Definition Arrays” .

• Add a new collation for the character set used by the indexed columns, and alter the columns to use that collation. For general information about adding collations, see Section 10. 14, “Adding a Collation to a Character Set” . For an example specific to full-text indexing, see [Section 12.10.7, “Adding a](#_bookmark197)

[User-Defined Collation for Full-Text Indexing”](#_bookmark197) .

**Rebuilding** **InnoDB** **Full-Text** **Indexes**

For the changes to take effect, FULLTEXT indexes must be rebuilt after modifying any of the following full-text index variables: innodb\_ft\_min\_token\_size; innodb\_ft\_max\_token\_size; innodb\_ft\_server\_stopword\_table; innodb\_ft\_user\_stopword\_table; innodb\_ft\_enable\_stopword; ngram\_token\_size. Modifying innodb\_ft\_min\_token\_size, innodb\_ft\_max\_token\_size, or ngram\_token\_size requires restarting the server.

To rebuild FULLTEXT indexes for an InnoDB table, use ALTER TABLE with the DROP INDEX and ADD INDEX options to drop and re-create each index.

**Optimizing** **InnoDB** **Full-Text** **Indexes**

Running OPTIMIZE TABLE on a table with a full-text index rebuilds the full-text index, removing deleted Document IDs and consolidating multiple entries for the same word, where possible.

To optimize a full-text index, enable innodb\_optimize\_fulltext\_only and run OPTIMIZE

TABLE.

mysql> set GLOBAL innodb\_optimize\_fulltext\_only=ON;

Query OK, 0 rows affected (0.01 sec)

mysql> OPTIMIZE TABLE opening\_lines;

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

| Table | Op | Msg\_type | Msg\_text |

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

| test .opening\_lines | optimize | status | OK |

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

1 row in set (0.01 sec)

To avoid lengthy rebuild times for full-text indexes on large tables, you can use the innodb\_ft\_num\_word\_optimize option to perform the optimization in stages. The innodb\_ft\_num\_word\_optimize option defines the number of words that are optimized each time OPTIMIZE TABLE is run. The default setting is 2000, which means that 2000 words are optimized each time OPTIMIZE TABLE is run. Subsequent OPTIMIZE TABLE operations continue from where the preceding OPTIMIZE TABLE operation ended.

**Rebuilding** **MyISAM** **Full-Text** **Indexes**

If you modify full-text variables that affect indexing (ft\_min\_word\_len, ft\_max\_word\_len, or ft\_stopword\_file), or if you change the stopword file itself, you must rebuild your FULLTEXT indexes after making the changes and restarting the server.

To rebuild the FULLTEXT indexes for a MyISAM table, it is sufficient to do a QUICK repair operation:

mysql> **REPAIR** **TABLE** ***tb1\_name*** **QUICK;**

Alternatively, use ALTER TABLE as just described. In some cases, this may be faster than a repair operation.

Each table that contains any FULLTEXT index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table causes the server to see the table as corrupt and in need of repair.

If you use myisamchk to perform an operation that modifies MyISAM table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or stopword file values used by the server, specify the same ft\_min\_word\_len, ft\_max\_word\_len, and ft\_stopword\_file values for myisamchk that you use for mysqld. For example, if you have set the minimum word length to 3, you can repair a table with myisamchk like this:

myisamchk --recover --ft\_min\_word\_len=3 *tbl\_name*.MYI

To ensure that myisamchk and the server use the same values for full-text parameters, place each one in both the [mysqld] and [myisamchk] sections of an option file:

[mysqld]

ft\_min\_word\_len=3

[myisamchk]

ft\_min\_word\_len=3

An alternative to using myisamchk for MyISAM table index modification is to use the REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE, or ALTER TABLE statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

**12.10.7** **Adding** **a** **User-Defined** **Collation** **for** **Full-Text** **Indexing**

**Warning**

User-defined collations are deprecated; you should expect support for them to be removed in a future version of MySQL. As of MySQL 8.0.33, the server issues a warning for any use of COLLATE *user\_defined\_collation* in an SQL statement; a warning is also issued when the server is started with -- collation-server set equal to the name of a user-defined collation.

This section describes how to add a user-defined collation for full-text searches using the built-in full- text parser. The sample collation is like latin1\_swedish\_ci but treats the '-' character as a letter rather than as a punctuation character so that it can be indexed as a word character. General information about adding collations is given in Section 10.14, “Adding a Collation to a Character Set” ; it is assumed that you have read it and are familiar with the files involved.

To add a collation for full-text indexing, use the following procedure. The instructions here add a collation for a simple character set, which as discussed in Section 10.14, “Adding a Collation to a Character Set” , can be created using a configuration file that describes the character set properties. For a complex character set such as Unicode, create collations using C source files that describe the character set properties.

1. Add a collation to the Index.xml file. The permitted range of IDs for user-defined collations is given in Section 10.14.2, “Choosing a Collation ID” . The ID must be unused, so choose a value different from 1025 if that ID is already taken on your system.

<charset name="latin1">

...

<collation name="latin1\_fulltext\_ci" id="1025"/>

</charset>

2. Declare the sort order for the collation in the latin1.xml file. In this case, the order can be copied from latin1\_swedish\_ci:

**)** **ENGINE=InnoDB;**

Query OK, 0 rows affected (0.47 sec)

|  |  |  |  |
| --- | --- | --- | --- |
| <collation name="latin1\_fulltext\_ci">  <mapmap>  </collation> | |  |  | | --- | --- | | 0D 0E  1D 1E  2D 2E  3D 3E  4D 4E  5D 5E  4D 4E  7D 7E  8D 8E  9D 9E | 0F  1F  2F  3F  4F  5F  4F  7F  8F  9F |   AD AE AF  BD BE BF  49 49 49  59 DE DF  49 49 49  59 DE FF |

3. Modify the ctype array in latin1.xml. Change the value corresponding to 0x2D (which is the code for the '-' character) from 10 (punctuation) to 01 (uppercase letter). In the following array, this is the element in the fourth row down, third value from the end.

<ctype>

<map>

00

20 20 20 20 20 20 20 20 20 28 28 28 28 28 20 20

20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20

48 10 10 10 10 10 10 10 10 10 10 10 10 *01* 10 10

84 84 84 84 84 84 84 84 84 84 10 10 10 10 10 10

10 81 81 81 81 81 81 01 01 01 01 01 01 01 01 01

01 01 01 01 01 01 01 01 01 01 01 10 10 10 10 10

10 82 82 82 82 82 82 02 02 02 02 02 02 02 02 02

02 02 02 02 02 02 02 02 02 02 02 10 10 10 10 20

10 00 10 02 10 10 10 10 10 10 01 10 01 00 01 00

00 10 10 10 10 10 10 10 10 10 02 10 02 00 02 01

48 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01

01 01 01 01 01 01 01 10 01 01 01 01 01 01 01 02

02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02

02 02 02 02 02 02 02 10 02 02 02 02 02 02 02 02

</map>

</ctype>

4. Restart the server.

5. To employ the new collation, include it in the definition of columns that are to use it:

mysql> **DROP** **TABLE** **IF** **EXISTS** **t1;**

Query OK, 0 rows affected (0.13

mysql> **CREATE** **TABLE** **t1** **(**

**a** **TEXT** **CHARACTER** **SET** **latin1**

**FULLTEXT** **INDEX(a)**

**COLLATE** **latin1\_fulltext\_ci,**

sec)

6. Test the collation to verify that hyphen is considered as a word character:

mysql> **INSERT** **INTO** **t1** **VALUEs** **('----** **'),('** **.** **.** **.** **.** **'),('abcd');**

Query OK, 3 rows affected (0.22 sec)

Records: 3 Duplicates: 0 Warnings: [0](#_bookmark49)

mysql> **SELECT** **\*** **FROM** **t1** **WHERE** **MATCH** **a** **AGAINST** **('----** **'** **IN** **BOOLEAN** **MODE);**

+------+

| a |

+------+



| ---- |

+------+

1 row in set (0.00 sec)

**12.10.8** **ngram** **Full-Text** **Parser**

The built-in MySQL full-text parser uses the white space between words as a delimiter to determine where words begin and end, which is a limitation when working with ideographic languages that do not use word delimiters. To address this limitation, MySQL provides an ngram full-text parser that supports Chinese, Japanese, and Korean (CJK). The ngram full-text parser is supported for use with InnoDB and MyISAM.

**Note**

MySQL also provides a MeCab full-text parser plugin for Japanese, which tokenizes documents into meaningful words. For more information, see [Section 12.10.9, “MeCab Full-Text Parser Plugin”](#_bookmark178) .

An ngram is a contiguous sequence of *n* characters from a given sequence of text. The ngram parser tokenizes a sequence of text into a contiguous sequence of *n* characters. For example, you can tokenize “abcd” for different values of *n* using the ngram full-text parser.

n=1: 'a', 'b', 'c', 'd'

n=2: 'ab', 'bc', 'cd'

n=3: 'abc', 'bcd'

n=4: 'abcd'

The ngram full-text parser is a built-in server plugin. As with other built-in server plugins, it is automatically loaded when the server is started.

The full-text search syntax described in [Section 12.10, “Full-Text Search Functions”](#_bookmark136) applies to the ngram parser plugin. Differences in parsing behavior are described in this section. Full- text-related configuration options, except for minimum and maximum word length options (innodb\_ft\_min\_token\_size, innodb\_ft\_max\_token\_size, ft\_min\_word\_len, ft\_max\_word\_len) are also applicable.

**Configuring** **ngram** **Token** **Size**

The ngram parser has a default ngram token size of 2 (bigram). For example, with a token size of 2, the ngram parser parses the string “abc def” into four tokens: “ab” , “bc” , “de” and “ef” .

ngram token size is configurable using the ngram\_token\_size configuration option, which has a minimum value of 1 and maximum value of 10.

Typically, ngram\_token\_size is set to the size of the largest token that you want to search for. If you only intend to search for single characters, set ngram\_token\_size to 1. A smaller token size produces a smaller full-text search index, and faster searches. If you need to search for words comprised of more than one character, set ngram\_token\_size accordingly. For example, “Happy Birthday” is “生日快乐” in simplified Chinese, where “生日” is “birthday” , and “快乐” translates as “happy” . To search on two-character words such as these, set ngram\_token\_size to a value of 2 or higher.

As a read-only variable, ngram\_token\_size may only be set as part of a startup string or in a configuration file:

• Startup string:

mysqld --ngram\_token\_size=2

• Configuration file:

[mysqld]

ngram\_token\_size=2



**Note**

The following minimum and maximum word length configuration options are ignored for FULLTEXT indexes that use the ngram parser: innodb\_ft\_min\_token\_size, innodb\_ft\_max\_token\_size, ft\_min\_word\_len, and ft\_max\_word\_len.

**Creating** **a** **FULLTEXT** **Index** **that** **Uses** **the** **ngram** **Parser**

To create a FULLTEXT index that uses the ngram parser, specify WITH PARSER ngram with CREATE TABLE, ALTER TABLE, or CREATE INDEX.

The following example demonstrates creating a table with an ngram FULLTEXT index, inserting sample data (Simplified Chinese text), and viewing tokenized data in the Information Schema INNODB\_FT\_INDEX\_CACHE table.

mysql> USE test;

mysql> CREATE TABLE articles (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

title VARCHAR(200),

body TEXT,

FULLTEXT (title,body) WITH PARSER ngram

) ENGINE=InnoDB CHARACTER SET utf8mb4;

mysql> SET NAMES utf8mb4;

INSERT INTO articles (title,body) VALUES

('数据库管理','在本教程中我将向你展示如何管理数据库'),

('数据库应用开发','学习开发数据库应用程序');

mysql> SET GLOBAL innodb\_ft\_aux\_table="test/articles";

mysql> SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_FT\_INDEX\_CACHE ORDER BY doc\_id, position;

To add a FULLTEXT index to an existing table, you can use ALTER TABLE or CREATE INDEX. For example:

CREATE TABLE articles (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

title VARCHAR(200),

body TEXT

) ENGINE=InnoDB CHARACTER SET utf8mb4;

ALTER TABLE articles ADD FULLTEXT INDEX ft\_index (title,body) WITH PARSER ngram;

# Or:

CREATE FULLTEXT INDEX ft\_index ON articles (title,body) WITH PARSER ngram;

**ngram** **Parser** **Space** **Handling**

The ngram parser eliminates spaces when parsing. For example:

• “ab cd” is parsed to “ab” , “cd”

• “a bc” is parsed to “bc”

**ngram** **Parser** **Stopword** **Handling**

The built-in MySQL full-text parser compares words to entries in the stopword list. If a word is

equal to an entry in the stopword list, the word is excluded from the index. For the ngram parser, stopword handling is performed differently. Instead of excluding tokens that are equal to entries in the stopword list, the ngram parser excludes tokens that *contain* stopwords. For example, assuming ngram\_token\_size=2, a document that contains “a,b” is parsed to “a,” and “,b” . If a comma (“,”) is defined as a stopword, both “a,” and “,b” are excluded from the index because they contain a comma.

By default, the ngram parser uses the default stopword list, which contains a list of English stopwords. For a stopword list applicable to Chinese, Japanese, or Korean, you must create your own. For information about creating a stopword list, see [Section 12.10.4, “Full-Text Stopwords”](#_bookmark179) .

Stopwords greater in length than ngram\_token\_size are ignored.

**ngram** **Parser** **Term** **Search**

For *natural* *language* *mode* search, the search term is converted to a union of ngram terms. For example, the string “abc” (assuming ngram\_token\_size=2) is converted to “ab bc” . Given two documents, one containing “ab” and the other containing “abc” , the search term “ab bc” matches both documents.

For *boolean* *mode* *search*, the search term is converted to an ngram phrase search. For example, the string 'abc' (assuming ngram\_token\_size=2) is converted to '“ab bc”'. Given two documents, one containing 'ab' and the other containing 'abc', the search phrase '“ab bc”' only matches the document containing 'abc'.

**ngram** **Parser** **Wildcard** **Search**

Because an ngram FULLTEXT index contains only ngrams, and does not contain information about the beginning of terms, wildcard searches may return unexpected results. The following behaviors apply to wildcard searches using ngram FULLTEXT search indexes:

• If the prefix term of a wildcard search is shorter than ngram token size, the query returns all indexed rows that contain ngram tokens starting with the prefix term. For example, assuming ngram\_token\_size=2, a search on “a\*” returns all rows starting with “a” .

• If the prefix term of a wildcard search is longer than ngram token size, the prefix term is converted to an ngram phrase and the wildcard operator is ignored. For example, assuming ngram\_token\_size=2, an “abc\*” wildcard search is converted to “ab bc” .

**ngram** **Parser** **Phrase** **Search**

Phrase searches are converted to ngram phrase searches. For example, The search phrase “abc” is converted to “ab bc” , which returns documents containing “abc” and “ab bc” .

The search phrase “abc def” is converted to “ab bc de ef” , which returns documents containing “abc def” and “ab bc de ef” . A document that contains “abcdef” is not returned.

**12.10.9** **MeCab** **Full-Text** **Parser** **Plugin**

The built-in MySQL full-text parser uses the white space between words as a delimiter to determine where words begin and end, which is a limitation when working with ideographic languages that do not use word delimiters. To address this limitation for Japanese, MySQL provides a MeCab full-text parser plugin. The MeCab full-text parser plugin is supported for use with InnoDB and MyISAM.

**Note**

MySQL also provides an ngram full-text parser plugin that supports Japanese. For more information, see [Section 12.10.8, “ngram Full-Text Parser”](#_bookmark177) .

The MeCab full-text parser plugin is a full-text parser plugin for Japanese that tokenizes a sequence of text into meaningful words. For example, MeCab tokenizes “デー 夕ベー ス管理” (“Database Management”) into “デー 夕ベー ス” (“Database” ) and “管理” (“Management”). By comparison, the [ngram](#_bookmark177) full-text parser tokenizes text into a contiguous sequence of *n* characters, where *n* represents a number between 1 and 10.

In addition to tokenizing text into meaningful words, MeCab indexes are typically smaller than ngram indexes, and MeCab full-text searches are generally faster. One drawback is that it may take longer for the MeCab full-text parser to tokenize documents, compared to the ngram full-text parser.

The full-text search syntax described in [Section 12.10, “Full-Text Search Functions”](#_bookmark136) applies to the MeCab parser plugin. Differences in parsing behavior are described in this section. Full-text related configuration options are also applicable.

For additional information about the MeCab parser, refer to the [MeCab: Yet Another Part-of-Speech](http://taku910.github.io/mecab/) [and Morphological Analyzer](http://taku910.github.io/mecab/) project on Github.

**Installing** **the** **MeCab** **Parser** **Plugin**

The MeCab parser plugin requires mecab and mecab-ipadic.

On supported Fedora, Debian and Ubuntu platforms (except Ubuntu 12.04 where the system mecab version is too old), MySQL dynamically links to the system mecab installation if it is installed to the default location. On other supported Unix-like platforms, libmecab.so is statically linked in libpluginmecab.so, which is located in the MySQL plugin directory. mecab-ipadic is included in MySQL binaries and is located in *MYSQL\_HOME*\lib\mecab.

You can install mecab and mecab-ipadic using a native package management utility (on Fedora, Debian, and Ubuntu), or you can build mecab and mecab-ipadic from source. For information about installing mecab and mecab-ipadic using a native package management utility, see [Installing MeCab](#_bookmark198) [From a Binary Distribution (Optional)](#_bookmark198). If you want to build mecab and mecab-ipadic from source, see [Building MeCab From Source (Optional)](#_bookmark199).

On Windows, libmecab.dll is found in the MySQL bin directory. mecab-ipadic is located in *MYSQL\_HOME*/lib/mecab.

To install and configure the MeCab parser plugin, perform the following steps:

1. In the MySQL configuration file, set the mecab\_rc\_file configuration option to the location of the mecabrc configuration file, which is the configuration file for MeCab. If you are using the MeCab package distributed with MySQL, the mecabrc file is located in MYSQL\_HOME/lib/mecab/etc/.

[mysqld]

loose-mecab-rc-file=MYSQL\_HOME/lib/mecab/etc/mecabrc

The loose prefix is an option modifier. The mecab\_rc\_file option is not recognized by MySQL until the MeCaB parser plugin is installed but it must be set before attempting to install the MeCaB parser plugin. The loose prefix allows you restart MySQL without encountering an error due to an unrecognized variable.

If you use your own MeCab installation, or build MeCab from source, the location of the mecabrc configuration file may differ.

For information about the MySQL configuration file and its location, see Section 4.2.2.2, “Using Option Files” .

2. Also in the MySQL configuration file, set the minimum token size to 1 or 2, which are the values recommended for use with the MeCab parser. For InnoDB tables, minimum token size is defined by the innodb\_ft\_min\_token\_size configuration option, which has a default value of 3. For MyISAM tables, minimum token size is defined by ft\_min\_word\_len, which has a default value of 4.

[mysqld]

innodb\_ft\_min\_token\_size=1

3. Modify the mecabrc configuration file to specify the dictionary you want to use. The mecab- ipadic package distributed with MySQL binaries includes three dictionaries (ipadic\_euc-jp, ipadic\_sjis, and ipadic\_utf-8). The mecabrc configuration file packaged with MySQL contains and entry similar to the following:

dicdir = /path/to/mysql/lib/mecab/lib/mecab/dic/ipadic\_euc-jp

To use the ipadic\_utf-8 dictionary, for example, modify the entry as follows:

dicdir=*MYSQL\_HOME*/lib/mecab/dic/ipadic\_utf-8

If you are using your own MeCab installation or have built MeCab from source, the default dicdir entry in the mecabrc file is likely to differ, as are the dictionaries and their location.

**Note**

After the MeCab parser plugin is installed, you can use the mecab\_charset status variable to view the character set used with MeCab. The three MeCab dictionaries provided with the MySQL binary support the following character sets.

• The ipadic\_euc-jp dictionary supports the ujis and eucjpms character sets.

• The ipadic\_sjis dictionary supports the sjis and cp932 character sets.

• The ipadic\_utf-8 dictionary supports the utf8mb3 and utf8mb4 character sets.

mecab\_charset only reports the first supported character set. For example, the ipadic\_utf-8 dictionary supports both utf8mb3 and utf8mb4. mecab\_charset always reports utf8 when this dictionary is in use.

4. Restart MySQL.

5. Install the MeCab parser plugin:

The MeCab parser plugin is installed using INSTALL PLUGIN syntax. The plugin name is mecab, and the shared library name is libpluginmecab.so. For additional information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins” .

INSTALL PLUGIN mecab SONAME 'libpluginmecab.so';

Once installed, the MeCab parser plugin loads at every normal MySQL restart.

6. Verify that the MeCab parser plugin is loaded using the SHOW PLUGINS statement. mysql> SHOW PLUGINS;

A mecab plugin should appear in the list of plugins.

**Creating** **a** **FULLTEXT** **Index** **that** **uses** **the** **MeCab** **Parser**

To create a FULLTEXT index that uses the mecab parser, specify WITH PARSER ngram with CREATE TABLE, ALTER TABLE, or CREATE INDEX.

This example demonstrates creating a table with a mecab FULLTEXT index, inserting sample data, and viewing tokenized data in the Information Schema INNODB\_FT\_INDEX\_CACHE table:

mysql> USE test;

mysql> CREATE TABLE articles (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

title VARCHAR(200),

body TEXT,

FULLTEXT (title,body) WITH PARSER mecab

) ENGINE=InnoDB CHARACTER SET utf8mb4;

mysql> SET NAMES utf8mb4;

mysql> INSERT INTO articles (title,body) VALUES

('デー タベー ス管理',' このチュ ー ト リアルでは 、私はどのようにデー タベー スを管理する方法を紹介します'), ('デー タベー スアプリケー ション開発','デー タベー スアプリケー ションを開発することを学ぶ');

mysql> SET GLOBAL innodb\_ft\_aux\_table="test/articles";

mysql> SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_FT\_INDEX\_CACHE ORDER BY doc\_id, position;

To add a FULLTEXT index to an existing table, you can use ALTER TABLE or CREATE INDEX. For example:

CREATE TABLE articles (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

title VARCHAR(200),

body TEXT

) ENGINE=InnoDB CHARACTER SET utf8mb4;

ALTER TABLE articles ADD FULLTEXT INDEX ft\_index (title,body) WITH PARSER mecab;

# Or:

CREATE FULLTEXT INDEX ft\_index ON articles (title,body) WITH PARSER mecab;

**MeCab** **Parser** **Space** **Handling**

The MeCab parser uses spaces as separators in query strings. For example, the MeCab parser tokenizes デ ー タベー ス管理 as デ ー タベー ス and 管理.

**MeCab** **Parser** **Stopword** **Handling**

By default, the MeCab parser uses the default stopword list, which contains a short list of English stopwords. For a stopword list applicable to Japanese, you must create your own. For information about creating stopword lists, see [Section 12.10.4, “Full-Text Stopwords”](#_bookmark179) .

**MeCab** **Parser** **Term** **Search**

For natural language mode search, the search term is converted to a union of tokens. For example, デ ー タベー ス管理 is converted to デ ー タベー ス 管理.

SELECT COUNT(\*) FROM articles WHERE MATCH(title,body) AGAINST('デー タベー ス管理 ' IN NATURAL LANGUAGE MODE

For boolean mode search, the search term is converted to a search phrase. For example,

デ ー タベー ス管理 is converted to デ ー タベー ス 管理. SELECT COUNT(\*) FROM articles WHERE MATCH(title,body) AGAINST('デー タベー ス管理 ' IN BOOLEAN MODE);

**MeCab** **Parser** **Wildcard** **Search**

Wildcard search terms are not tokenized. A search on デ ー タベー ス管理\* is performed on the prefix, デ ー タベー ス管理.

SELECT COUNT(\*) FROM articles WHERE MATCH(title,body) AGAINST('デー タベー ス\*' IN BOOLEAN MODE);

**MeCab** **Parser** **Phrase** **Search**

Phrases are tokenized. For example, デ ー タベー ス管理 is tokenized as デ ー タベー ス 管理. SELECT COUNT(\*) FROM articles WHERE MATCH(title,body) AGAINST('"デー タベー ス管理"' IN BOOLEAN MODE);

**Installing** **MeCab** **From** **a** **Binary** **Distribution** **(Optional)**

This section describes how to install mecab and mecab-ipadic from a binary distribution using a native package management utility. For example, on Fedora, you can use Yum to perform the installation:

yum mecab-devel

On Debian or Ubuntu, you can perform an APT installation:

apt-get install mecab

apt-get install mecab-ipadic

**Installing** **MeCab** **From** **Source** **(Optional)**

If you want to build mecab and mecab-ipadic from source, basic installation steps are provided below. For additional information, refer to the MeCab documentation.

1. Download the tar.gz packages for mecab and mecab-ipadic from [http://taku910.github.io/mecab/](http://taku910.github.io/mecab/#download) [#download](http://taku910.github.io/mecab/#download). As of February, 2016, the latest available packages are mecab-0.996.tar.gz and mecab-ipadic-2.7.0-20070801.tar.gz.

2. Install mecab:

tar zxfv mecab-0.996.tar

cd mecab-0 .996

./configure

make

make check

su

make install

3. Install mecab-ipadic:

tar zxfv mecab-ipadic-2.7.0-20070801.tar

cd mecab-ipadic-2 .7 .0-20070801

./configure

make

su

make install

4. Compile MySQL using the WITH\_MECAB CMake option. Set the WITH\_MECAB option to system if you have installed mecab and mecab-ipadic to the default location.

-DWITH\_MECAB=system

If you defined a custom installation directory, set WITH\_MECAB to the custom directory. For example:

-DWITH\_MECAB=/path/to/mecab

**12.11** **Cast** **Functions** **and** **Operators**

**Table** **12.15** **Cast** **Functions** **and** **Operators**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Deprecated** |
| [BINARY](#_bookmark200) | Cast a string to a binary string | 8.0.27 |
| [CAST()](#_bookmark109) | Cast a value as a certain type |  |
| [CONVERT()](#_bookmark201) | Cast a value as a certain type |  |

Cast functions and operators enable conversion of values from one data type to another.

• [Cast Function and Operator Descriptions](#_bookmark202)

• [Character Set Conversions](#_bookmark203)

• [Character Set Conversions for String Comparisons](#_bookmark204)

• [Cast Operations on Spatial Types](#_bookmark205)



• [Other Uses for Cast Operations](#_bookmark206)

**Cast** **Function** **and** **Operator** **Descriptions**

• [BINARY](#_bookmark200) *expr*

The [BINARY](#_bookmark200) operator converts the expression to a binary string (a string that has the binary character set and binary collation). A common use for [BINARY](#_bookmark200) is to force a character string comparison to be done byte by byte using numeric byte values rather than character by character. The [BINARY](#_bookmark200) operator also causes trailing spaces in comparisons to be significant. For information about the differences between the binary collation of the binary character set and the \_bin collations of nonbinary character sets, see Section 10.8.5, “The binary Collation Compared to \_bin

Collations” .

The BINARY operator is deprecated as of MySQL 8.0.27, and you should expect its removal in a future version of MySQL. Use [CAST(... AS BINARY)](#_bookmark109) instead.

mysql> **SET** **NAMES** **utf8mb4** **COLLATE** **utf8mb4\_general\_ci;**

-> OK

mysql> **SELECT** **'a'** **=** **'A';**

-> 1

mysql> **SELECT** **BINARY** **'a'** **=** **'A';**

-> 0

mysql> **SELECT** **'a'** **=** **'a** **';**

-> 1

mysql> **SELECT** **BINARY** **'a'** **=** **'a** **';**

-> 0

In a comparison, [BINARY](#_bookmark200) affects the entire operation; it can be given before either operand with the same result.

To convert a string expression to a binary string, these constructs are equivalent:

CONVERT(*expr* USING BINARY)

CAST(*expr* AS BINARY)

BINARY *expr*

If a value is a string literal, it can be designated as a binary string without converting it by using the \_binary character set introducer:

mysql> **SELECT** **'a'** **=** **'A';**

-> 1

mysql> **SELECT** **\_binary** **'a'** **=** **'A';**

-> 0

For information about introducers, see Section 10.3.8, “Character Set Introducers” .

The [BINARY](#_bookmark200) operator in expressions differs in effect from the BINARY attribute in character column definitions. For a character column defined with the BINARY attribute, MySQL assigns the table default character set and the binary (\_bin) collation of that character set. Every nonbinary character set has a \_bin collation. For example, if the table default character set is utf8mb4, these two column definitions are equivalent:

CHAR(10) BINARY

CHAR(10) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin

The use of CHARACTER SET binary in the definition of a CHAR, VARCHAR, or TEXT column causes the column to be treated as the corresponding binary string data type. For example, the following pairs of definitions are equivalent:

CHAR(10) CHARACTER SET binary

BINARY(10)

VARCHAR(10) CHARACTER SET binary

VARBINARY(10)

TEXT CHARACTER SET binary

BLOB

If [BINARY](#_bookmark200) is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [CAST(*expr* AS *type* [ARRAY])](#_bookmark109)

[CAST(*timestamp\_value* AT TIME ZONE *timezone\_specifier* AS](#_bookmark109)

[DATETIME[(*precision*)])](#_bookmark109)

*timezone\_specifier*: [INTERVAL] '+00:00' | 'UTC'

With [CAST(*expr* AS](#_bookmark109)*type* syntax, the [CAST()](#_bookmark109) function takes an expression of any type and produces a result value of the specified type. This operation may also be expressed as [CONVERT(*expr*, *type*)](#_bookmark201), which is equivalent. If *expr* is NULL, CAST() returns NULL.

These *type* values are permitted:

• BINARY[(*N*)]

Produces a string with the VARBINARY data type, except that when the expression *expr* is empty (zero length), the result type is BINARY(0). If the optional length *N* is given, BINARY(*N*) causes the cast to use no more than *N* bytes of the argument. Values shorter than *N* bytes are padded with 0x00 bytes to a length of *N*. If the optional length *N* is not given, MySQL calculates the maximum length from the expression. If the supplied or calculated length is greater than an internal threshold, the result type is BLOB. If the length is still too long, the result type is LONGBLOB.

For a description of how casting to BINARY affects comparisons, see Section 11.3.3, “The BINARY and VARBINARY Types” .

• CHAR[(*N*)] [*charset\_info*]

Produces a string with the VARCHAR data type. except that when the expression *expr* is empty (zero length), the result type is CHAR(0). If the optional length *N* is given, CHAR(*N*) causes the cast to use no more than *N*characters of the argument. No padding occurs for values shorter than *N*characters. If the optional length *N* is not given, MySQL calculates the maximum length from the expression. If the supplied or calculated length is greater than an internal threshold, the result type is TEXT. If the length is still too long, the result type is LONGTEXT.

With no *charset\_info* clause, CHAR produces a string with the default character set. To specify the character set explicitly, these *charset\_info* values are permitted:

• CHARACTER SET *charset\_name*: Produces a string with the given character set.

• ASCII: Shorthand for CHARACTER SET latin1.

• UNICODE: Shorthand for CHARACTER SET ucs2.

In all cases, the string has the character set default collation.

• DATE

Produces a DATE value.

• DATETIME[(*M*)]

Produces a DATETIME value. If the optional *M*value is given, it specifies the fractional seconds precision.

• DECIMAL[(*M*[,*D*])]

Produces a DECIMAL value. If the optional *M*and *D* values are given, they specify the maximum number of digits (the precision) and the number of digits following the decimal point (the scale). If *D* is omitted, 0 is assumed. If *M* is omitted, 10 is assumed.

• DOUBLE

Produces a DOUBLE result. Added in MySQL 8.0.17.

• FLOAT[(*p*)]

If the precision *p* is not specified, produces a result of type FLOAT. If *p* is provided and 0 <= < *p* <= 24, the result is of type FLOAT. If 25 <= *p* <= 53, the result is of type DOUBLE. If *p* < 0 or *p* > 53, an error is returned. Added in MySQL 8.0.17.

• JSON

Produces a JSON value. For details on the rules for conversion of values between JSON and other types, see Comparison and Ordering of JSON Values.

• NCHAR[(*N*)]

Like CHAR, but produces a string with the national character set. See Section 10.3.7, “The National

Character Set” .

Unlike CHAR, NCHAR does not permit trailing character set information to be specified.

• REAL

Produces a result of type REAL. This is actually FLOAT if the REAL\_AS\_FLOAT SQL mode is enabled; otherwise the result is of type DOUBLE.

• SIGNED [INTEGER]

Produces a signed BIGINT value.

• *spatial\_type*

As of MySQL 8.0.24, [CAST()](#_bookmark109) and [CONVERT()](#_bookmark201) support casting geometry values from one spatial type to another, for certain combinations of spatial types. For details, see [Cast Operations on](#_bookmark205) [Spatial Types](#_bookmark205).

• TIME[(*M*)]

Produces a TIME value. If the optional *M*value is given, it specifies the fractional seconds precision.

• UNSIGNED [INTEGER]

Produces an unsigned BIGINT value.

• YEAR

Produces a YEAR value. Added in MySQL 8.0.22. These rules govern conversion to YEAR:

• For a four-digit number in the range 1901-2155 inclusive, or for a string which can be interpreted as a four-digit number in this range, return the corresponding YEAR value.

• For a number consisting of one or two digits, or for a string which can be interpreted as such a number, return a YEAR value as follows:

• If the number is in the range 1-69 inclusive, add 2000 and return the sum.

• If the number is in the range 70-99 inclusive, add 1900 and return the sum.

• For a string which evaluates to 0, return 2000.

• For the number 0, return 0.

• For a DATE, DATETIME, or TIMESTAMP value, return the YEAR portion of the value. For a TIME value, return the current year.

If you do not specify the type of a TIME argument, you may get a different result from what you expect, as shown here:

mysql> **SELECT** **CAST("11:35:00"** **AS** **YEAR),** **CAST(TIME** **"11:35:00"** **AS** **YEAR);**

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

| CAST("11:35:00" AS YEAR) | CAST(TIME "11:35:00" AS YEAR) |

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

| 2011 | 2021 |

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

• If the argument is of type DECIMAL, DOUBLE, DECIMAL, or REAL, round the value to the nearest integer, then attempt to cast the value to YEAR using the rules for integer values, as shown here:

mysql> **SELECT** **CAST(1944** **.35** **AS** **YEAR),** **CAST(1944** **.50** **AS** **YEAR);**

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

| CAST(1944 .35 AS YEAR) | CAST(1944 .50 AS YEAR) |

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

| 1944 | 1945 |

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

mysql> **SELECT** **CAST(66** **.35** **AS** **YEAR),** **CAST(66** **.50** **AS** **YEAR);**

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

| CAST(66 .35 AS YEAR) | CAST(66 .50 AS YEAR) |

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

| 2066 | 2067 |

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

• An argument of type GEOMETRY cannot be converted to YEAR.

• For a value that cannot be successfully converted to YEAR, return NULL.

A string value containing non-numeric characters which must be truncated prior to conversion raises a warning, as shown here:

mysql> **SELECT** **CAST("1979aaa"** **AS** **YEAR);**

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

| CAST("1979aaa" AS YEAR) |

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

|  |  |
| --- | --- |
| | | 1979 | |

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



1 row in set, 1 warning (0 .00 sec)

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1292 | Truncated incorrect YEAR value: '1979aaa' |

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

In MySQL 8.0.17 and higher, InnoDB allows the use of an additional ARRAY keyword for creating a multi-valued index on a JSON array as part of CREATE INDEX, CREATE TABLE, and ALTER TABLE statements. ARRAY is not supported except when used to create a multi-valued index in one of these statements, in which case it is required. The column being indexed must be a column of type JSON. With ARRAY, the *type* following the AS keyword may specify any of the types supported by CAST(), with the exceptions of BINARY, JSON, and YEAR. For syntax information and examples, as well as other relevant information, see Multi-Valued Indexes.

**Note**

[CONVERT()](#_bookmark201), unlike [CAST()](#_bookmark109), does *not* support multi-valued index creation or the ARRAY keyword.

Beginning with MySQL 8.0.22, CAST() supports retrieval of a TIMESTAMP value as being in UTC, using the AT TIMEZONE operator. The only supported time zone is UTC; this can be specified as either of '+00:00' or 'UTC'. The only return type supported by this syntax is DATETIME, with an optional precision specifier in the range of 0 to 6, inclusive.

TIMESTAMP values that use timezone offsets are also supported.

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

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

| @@system\_time\_zone |

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

| EDT |

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

1 row in set (0.00 sec)

mysql> **CREATE** **TABLE** **TZ** **(c** **TIMESTAMP);**

Query OK, 0 rows affected (0.41 sec)

mysql> **INSERT** **INTO** **tz** **VALUES**

-> **ROW(CURRENT\_TIMESTAMP),**

-> **ROW('2020-07-28** **14:50:15+1:00');**

Query OK, 1 row affected (0.08 sec)

mysql> **TABLE** **tz;**

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

| c |

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

| 2020-07-28 09:22:41 |

| 2020-07-28 09:50:15 |

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

2 rows in set (0.00 sec)

mysql> **SELECT** **CAST(c** **AT** **TIME** **ZONE** **'+00:00'** **AS** **DATETIME)** **AS** **u** **FROM** **tz;**

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

| u |

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

| 2020-07-28 13:22:41 |

| 2020-07-28 13:50:15 |

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

2 rows in set (0.00 sec)

mysql> **SELECT** **CAST(c** **AT** **TIME** **ZONE** **'UTC'** **AS** **DATETIME(2))** **AS** **u** **FROM** **tz;**

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

| u |

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

| 2020-07-28 13:22:41.00 |



| 2020-07-28 13:50:15 .00 |

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

2 rows in set (0.00 sec)

If you use 'UTC' as the time zone specifier with this form of CAST(), and the server raises an error such as Unknown or incorrect time zone: 'UTC', you may need to install the MySQL time zone tables (see Populating the Time Zone Tables).

AT TIME ZONE does not support the ARRAY keyword, and is not supported by the [CONVERT()](#_bookmark201) function.

• [CONVERT(*expr* USING *transcoding\_name*)](#_bookmark201) [CONVERT(*expr*,*type*)](#_bookmark201)

[CONVERT(*expr* USING *transcoding\_name*)](#_bookmark201) is standard SQL syntax. The non-USING form of [CONVERT()](#_bookmark201) is ODBC syntax. Regardless of the syntax used, the function returns NULL if *expr* is

NULL.

[CONVERT(*expr* USING *transcoding\_name*)](#_bookmark201) converts data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string 'abc' in the default character set to the corresponding string in the utf8mb4 character set:

SELECT CONVERT('abc' USING utf8mb4);

[CONVERT(*expr*, *type*)](#_bookmark201) syntax (without USING) takes an expression and a *type* value specifying a result type, and produces a result value of the specified type. This operation may also be expressed as [CAST(*expr* AS *type*)](#_bookmark109), which is equivalent. For more information, see the description of [CAST()](#_bookmark109).

**Note**

Prior to MySQL 8.0.28, this function sometimes allowed invalid conversions of BINARY values to a nonbinary character set. When CONVERT() was used as part of the expression for an indexed generated column, this could lead to index corruption following an upgrade from a previous version of MySQL. See SQL Changes, for information about how to handle this situation.

**Character** **Set** **Conversions**

[CONVERT()](#_bookmark201) with a USING clause converts data between character sets:

CONVERT(*expr* USING *transcoding\_name*)

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

SELECT CONVERT('test' USING utf8mb4);

SELECT CONVERT(\_latin1'Müller' USING utf8mb4);

INSERT INTO utf8mb4\_table (utf8mb4\_column)

SELECT CONVERT(latin1\_column USING utf8mb4) FROM latin1\_table;

To convert strings between character sets, you can also use [CONVERT(*expr*, *type*)](#_bookmark201) syntax (without USING), or [CAST(*expr* AS *type*)](#_bookmark109), which is equivalent:

CONVERT(*string*, CHAR[(*N*)] CHARACTER SET *charset\_name*)

CAST(*string* AS CHAR[(*N*)] CHARACTER SET *charset\_name*)

Examples:

SELECT CONVERT('test', CHAR CHARACTER SET utf8mb4);

SELECT CAST('test' AS CHAR CHARACTER SET utf8mb4);

If you specify CHARACTER SET *charset\_name* as just shown, the character set and collation of the result are *charset\_name* and the default collation of *charset\_name*. If you omit CHARACTER SET *charset\_name*, the character set and collation of the result are defined by the character\_set\_connection and collation\_connection system variables that determine the default connection character set and collation (see Section 10.4, “Connection Character Sets and Collations” ).

A COLLATE clause is not permitted within a [CONVERT()](#_bookmark201) or [CAST()](#_bookmark109) call, but you can apply it to the function result. For example, these are legal:

SELECT CONVERT('test' USING utf8mb4) COLLATE utf8mb4\_bin;

SELECT CONVERT('test', CHAR CHARACTER SET utf8mb4) COLLATE utf8mb4\_bin;

SELECT CAST('test' AS CHAR CHARACTER SET utf8mb4) COLLATE utf8mb4\_bin;

But these are illegal:

SELECT CONVERT('test' USING utf8mb4 COLLATE utf8mb4\_bin);

SELECT CONVERT('test', CHAR CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin);

SELECT CAST('test' AS CHAR CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin);

For string literals, another way to specify the character set is to use a character set introducer. \_latin1 and \_latin2 in the preceding example are instances of introducers. Unlike conversion functions such as [CAST()](#_bookmark109), or [CONVERT()](#_bookmark201), which convert a string from one character set to another, an introducer designates a string literal as having a particular character set, with no conversion involved. For more information, see Section 10.3.8, “Character Set Introducers” .

**Character** **Set** **Conversions** **for** **String** **Comparisons**

Normally, you cannot compare a BLOB value or other binary string in case-insensitive fashion because binary strings use the binary character set, which has no collation with the concept of lettercase. To perform a case-insensitive comparison, first use the [CONVERT()](#_bookmark201) or [CAST()](#_bookmark109) function to convert the value to a nonbinary string. Comparisons of the resulting string use its collation. For example, if the conversion result collation is not case-sensitive, a [LIKE](#_bookmark129) operation is not case-sensitive. That is true for the following operation because the default utf8mb4 collation (utf8mb4\_0900\_ai\_ci) is not case- sensitive:

SELECT 'A' LIKE CONVERT(*blob\_col* USING utf8mb4)

FROM *tbl\_name*;

To specify a particular collation for the converted string, use a COLLATE clause following the [CONVERT()](#_bookmark201) call:

SELECT 'A' LIKE CONVERT(*blob\_col* USING utf8mb4) COLLATE utf8mb4\_unicode\_ci

FROM *tbl\_name*;

To use a different character set, substitute its name for utf8mb4 in the preceding statements (and similarly to use a different collation).

[CONVERT()](#_bookmark201) and [CAST()](#_bookmark109) can be used more generally for comparing strings represented in different character sets. For example, a comparison of these strings results in an error because they have different character sets:

mysql> **SET** **@s1** **=** **\_latin1** **'abc',** **@s2** **=** **\_latin2** **'abc';**

mysql> **SELECT** **@s1** **=** **@s2;**

ERROR 1267 (HY000): Illegal mix of collations (latin1\_swedish\_ci,IMPLICIT)

and (latin2\_general\_ci,IMPLICIT) for operation '='

Converting one of the strings to a character set compatible with the other enables the comparison to occur without error:

mysql> **SELECT** **@s1** **=** **CONVERT(@s2** **USING** **latin1);**

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

| @s1 = CONVERT(@s2 USING latin1) |

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

| 1 |

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

Character set conversion is also useful preceding lettercase conversion of binary strings. [LOWER()](#_bookmark132) and [UPPER()](#_bookmark167) are ineffective when applied directly to binary strings because the concept of lettercase does not apply. To perform lettercase conversion of a binary string, first convert it to a nonbinary string using a character set appropriate for the data stored in the string:

mysql> **SET** **@str** **=** **BINARY** **'New** **York';**

mysql> **SELECT** **LOWER(@str),** **LOWER(CONVERT(@str** **USING** **utf8mb4));**

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

| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |

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

|  |  |  |  |
| --- | --- | --- | --- |
| | New York | | new | york | | |

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

Be aware that if you apply [BINARY](#_bookmark200), [CAST()](#_bookmark109), or [CONVERT()](#_bookmark201) to an indexed column, MySQL may not be able to use the index efficiently.

**Cast** **Operations** **on** **Spatial** **Types**

As of MySQL 8.0.24, [CAST()](#_bookmark109) and [CONVERT()](#_bookmark201) support casting geometry values from one spatial type to another, for certain combinations of spatial types. The following list shows the permitted type combinations, where “MySQL extension” designates casts implemented in MySQL beyond those defined in the SQL/MM standard:

• From Point to:

• MultiPoint

• GeometryCollection

• From LineString to:

• Polygon (MySQL extension)

• MultiPoint (MySQL extension)

• MultiLineString

• GeometryCollection

• From Polygon to:

• LineString (MySQL extension)

• MultiLineString (MySQL extension)

• MultiPolygon

• GeometryCollection

• From MultiPoint to:

• Point

• LineString (MySQL extension)

• GeometryCollection

• From MultiLineString to:

• LineString

• Polygon (MySQL extension)

• MultiPolygon (MySQL extension)

• GeometryCollection

• From MultiPolygon to:

• Polygon

• MultiLineString (MySQL extension)

• GeometryCollection

• From GeometryCollection to:

• Point

• LineString

• Polygon

• MultiPoint

• MultiLineString

• MultiPolygon

In spatial casts, GeometryCollection and GeomCollection are synonyms for the same result type.

Some conditions apply to all spatial type casts, and some conditions apply only when the cast result is to have a particular spatial type. For information about terms such as “well-formed geometry,” see Section 11.4.4, “Geometry Well-Formedness and Validity” .

• [General Conditions for Spatial Casts](#_bookmark207)

• [Conditions for Casts to Point](#_bookmark208)

• [Conditions for Casts to LineString](#_bookmark209)

• [Conditions for Casts to Polygon](#_bookmark210)

• [Conditions for Casts to MultiPoint](#_bookmark211)

• [Conditions for Casts to MultiLineString](#_bookmark212)

• [Conditions for Casts to MultiPolygon](#_bookmark213)

• [Conditions for Casts to GeometryCollection](#_bookmark214)

**General** **Conditions** **for** **Spatial** **Casts**

These conditions apply to all spatial casts regardless of the result type:

• The result of a cast is in the same SRS as that of the expression to cast.

• Casting between spatial types does not change coordinate values or order.

• If the expression to cast is NULL, the function result is NULL.

• Casting to spatial types using the JSON\_VALUE() function with a RETURNING clause specifying a spatial type is not permitted.

• Casting to an ARRAY of spatial types is not permitted.

• If the spatial type combination is permitted but the expression to cast is not a syntactically well- formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If the spatial type combination is permitted but the expression to cast is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• If the expression to cast has a geographic SRS but has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs.

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

**Conditions** **for** **Casts** **to** **Point**

When the cast result type is Point, these conditions apply:

• If the expression to cast is a well-formed geometry of type Point, the function result is that Point.

• If the expression to cast is a well-formed geometry of type MultiPoint containing a single Point, the function result is that Point. If the expression contains more than one Point, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type GeometryCollection containing only a single Point, the function result is that Point. If the expression is empty, contains more than one Point, or contains other geometry types, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type other than Point, MultiPoint, GeometryCollection, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

**Conditions** **for** **Casts** **to** **LineString**

When the cast result type is LineString, these conditions apply:

• If the expression to cast is a well-formed geometry of type LineString, the function result is that LineString.

• If the expression to cast is a well-formed geometry of type Polygon that has no inner rings, the function result is a LineString containing the points of the outer ring in the same order. If the expression has inner rings, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type MultiPoint containing at least two points, the function result is a LineString containing the points of the MultiPoint in the order they appear in the expression. If the expression contains only one Point, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type MultiLineString containing a single LineString, the function result is that LineString. If the expression contains more than one LineString, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type GeometryCollection, containing only a single LineString, the function result is that LineString. If the expression

is empty, contains more than one LineString, or contains other geometry types, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type other than LineString, Polygon, MultiPoint, MultiLineString, or GeometryCollection, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

**Conditions** **for** **Casts** **to** **Polygon**

When the cast result type is Polygon, these conditions apply:

• If the expression to cast is a well-formed geometry of type LineString that is a ring (that

is, the start and end points are the same), the function result is a Polygon with an outer ring consisting of the points of the LineString in the same order. If the expression is not a ring, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs. If the ring is not in the correct order (the exterior ring must be counter-clockwise), an [ER\_INVALID\_CAST\_POLYGON\_RING\_DIRECTION](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_polygon_ring_direction) error occurs.

• If the expression to cast is a well-formed geometry of type Polygon, the function result is that Polygon.

• If the expression to cast is a well-formed geometry of type MultiLineString where all elements are rings, the function result is a Polygon with the first LineString as outer

ring and any additional LineString values as inner rings. If any element of the expression is not a ring, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs. If any ring is not in the correct order (the exterior ring must be counter-clockwise, interior rings must be clockwise), an [ER\_INVALID\_CAST\_POLYGON\_RING\_DIRECTION](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_polygon_ring_direction) error occurs.

• If the expression to cast is a well-formed geometry of type MultiPolygon containing a single Polygon, the function result is that Polygon. If the expression contains more than one Polygon, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type GeometryCollection containing only a single Polygon, the function result is that Polygon. If the expression is empty, contains more than one Polygon, or contains other geometry types, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type other than LineString, Polygon, MultiLineString, MultiPolygon, or GeometryCollection, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

**Conditions** **for** **Casts** **to** **MultiPoint**

When the cast result type is MultiPoint, these conditions apply:

• If the expression to cast is a well-formed geometry of type Point, the function result is a MultiPoint containing that Point as its sole element.

• If the expression to cast is a well-formed geometry of type LineString, the function result is a MultiPoint containing the points of the LineString in the same order.

• If the expression to cast is a well-formed geometry of type MultiPoint, the function result is that MultiPoint.

• If the expression to cast is a well-formed geometry of type GeometryCollection containing only points, the function result is a MultiPoint containing those points. If the GeometryCollection is empty or contains other geometry types, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type other than Point, LineString, MultiPoint, or GeometryCollection, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

**Conditions** **for** **Casts** **to** **MultiLineString**

When the cast result type is MultiLineString, these conditions apply:

• If the expression to cast is a well-formed geometry of type LineString, the function result is a MultiLineString containing that LineString as its sole element.

• If the expression to cast is a well-formed geometry of type Polygon, the function result is a MultiLineString containing the outer ring of the Polygon as its first element and any inner rings as additional elements in the order they appear in the expression.

• If the expression to cast is a well-formed geometry of type MultiLineString, the function result is that MultiLineString.

• If the expression to cast is a well-formed geometry of type MultiPolygon containing only polygons without inner rings, the function result is a MultiLineString containing the polygon rings in the order they appear in the expression. If the expression contains any polygons with inner rings, an [ER\_WRONG\_PARAMETERS\_TO\_STORED\_FCT](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_parameters_to_stored_fct) error occurs.

• If the expression to cast is a well-formed geometry of type GeometryCollection containing only linestrings, the function result is a MultiLineString containing those linestrings. If the expression is empty or contains other geometry types, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type other than LineString, Polygon, MultiLineString, MultiPolygon, or GeometryCollection, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

**Conditions** **for** **Casts** **to** **MultiPolygon**

When the cast result type is MultiPolygon, these conditions apply:

• If the expression to cast is a well-formed geometry of type Polygon, the function result is a MultiPolygon containing the Polygon as its sole element.

• If the expression to cast is a well-formed geometry of type MultiLineString where all elements are rings, the function result is a MultiPolygon containing a Polygon with only an outer ring for each element of the expression. If any element is not a ring, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs. If any ring is not in the correct order (exterior ring must be counter-clockwise), an [ER\_INVALID\_CAST\_POLYGON\_RING\_DIRECTION](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_polygon_ring_direction) error occurs.

• If the expression to cast is a well-formed geometry of type MultiPolygon, the function result is that MultiPolygon.

• If the expression to cast is a well-formed geometry of type GeometryCollection containing only polygons, the function result is a MultiPolygon containing those polygons. If the expression is empty or contains other geometry types, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

• If the expression to cast is a well-formed geometry of type other than Polygon, MultiLineString, MultiPolygon, or GeometryCollection, an [ER\_INVALID\_CAST\_TO\_GEOMETRY](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_cast_to_geometry) error occurs.

**Conditions** **for** **Casts** **to** **GeometryCollection**

When the cast result type is GeometryCollection, these conditions apply:

• GeometryCollection and GeomCollection are synonyms for the same result type.

• If the expression to cast is a well-formed geometry of type Point, the function result is a GeometryCollection containing that Point as its sole element.

• If the expression to cast is a well-formed geometry of type LineString, the function result is a GeometryCollection containing that LineString as its sole element.

• If the expression to cast is a well-formed geometry of type Polygon, the function result is a GeometryCollection containing that Polygon as its sole element.

• If the expression to cast is a well-formed geometry of type MultiPoint, the function result is a GeometryCollection containing the points in the order they appear in the expression.

• If the expression to cast is a well-formed geometry of type MultiLineString, the function result is a GeometryCollection containing the linestrings in the order they appear in the expression.

• If the expression to cast is a well-formed geometry of type MultiPolygon, the function result is a GeometryCollection containing the elements of the MultiPolygon in the order they appear in the expression.

• If the expression to cast is a well-formed geometry of type GeometryCollection, the function result is that GeometryCollection.

**Other** **Uses** **for** **Cast** **Operations**

The cast functions are useful for creating a column with a specific type in a CREATE TABLE ... SELECT statement:

mysql> **CREATE** **TABLE** **new\_table** **SELECT** **CAST('2000-01-01'** **AS** **DATE)** **AS** **c1;**

mysql> **SHOW** **CREATE** **TABLE** **new\_table\G**

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

Table: new\_table

Create Table: CREATE TABLE `new\_table` (

`c1` date DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4

The cast functions are useful for sorting ENUM columns in lexical order. Normally, sorting of ENUM columns occurs using the internal numeric values. Casting the values to CHAR results in a lexical sort:

SELECT *enum\_col* FROM *tbl\_name* ORDER BY CAST(*enum\_col* AS CHAR);

[CAST()](#_bookmark109) also changes the result if you use it as part of a more complex expression such as [CONCAT('Date: ',CAST(NOW() AS DATE))](#_bookmark117).

For temporal values, there is little need to use [CAST()](#_bookmark109) to extract data in different formats. Instead, use a function such as [EXTRACT()](#_bookmark67), [DATE FORMAT()](#_bookmark59)\_, or [TIME FORMAT()](#_bookmark92)\_. See [Section 12.7, “Date](#_bookmark44)and [Time Functions”](#_bookmark44) .

To cast a string to a number, it normally suffices to use the string value in numeric context:

mysql> **SELECT** **1+'1';**

-> 2

That is also true for hexadecimal and bit literals, which are binary strings by default:

mysql> **SELECT** **X'41',** **X'41'+0;**

-> 'A', 65

mysql> **SELECT** **b'1100001',** **b'1100001'+0;**

-> 'a', 97

A string used in an arithmetic operation is converted to a floating-point number during expression evaluation.

A number used in string context is converted to a string:

mysql> **SELECT** **CONCAT('hello** **you** **',2);**

-> 'hello you 2'

For information about implicit conversion of numbers to strings, see Section 12.3, “Type Conversion in Expression Evaluation” .

MySQL supports arithmetic with both signed and unsigned 64-bit values. For numeric operators (such as [+](#_bookmark6) or [-](#_bookmark7)) where one of the operands is an unsigned integer, the result is unsigned by default (see [Section 12.6.1, “Arithmetic Operators”](#_bookmark41)). To override this, use the SIGNED or UNSIGNED cast operator to cast a value to a signed or unsigned 64-bit integer, respectively.

mysql> **SELECT** **1** **-** **2;**

-> -1

mysql> **SELECT** **CAST(1** **-** **2** **AS** **UNSIGNED);**

-> 18446744073709551615

mysql> **SELECT** **CAST(CAST(1** **-** **2** **AS** **UNSIGNED)** **AS** **SIGNED);**

-> -1

If either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, DECIMAL column values are regarded as floating-point values.)

mysql> **SELECT** **CAST(1** **AS** **UNSIGNED)** **-** **2.0;**



-> -1 .0

The SQL mode affects the result of conversion operations (see Section 5.1.11, “Server SQL Modes” ). Examples:

• For conversion of a “zero” date string to a date, [CONVERT()](#_bookmark201) and [CAST()](#_bookmark109) return NULL and produce a warning when the NO\_ZERO\_DATE SQL mode is enabled.

• For integer subtraction, if the NO\_UNSIGNED\_SUBTRACTION SQL mode is enabled, the subtraction result is signed even if any operand is unsigned.

**12.12** **XML** **Functions**

**Table** **12.16** **XML** **Functions**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [ExtractValue()](#_bookmark215) | Extract a value from an XML string using XPath notation |
| [UpdateXML()](#_bookmark216) | Return replaced XML fragment |

This section discusses XML and related functionality in MySQL.

**Note**

It is possible to obtain XML-formatted output from MySQL in the mysql and mysqldump clients by invoking them with the --xml option. See Section 4.5.1, “mysql — The MySQL Command-Line Client” , and Section 4.5.4, “mysqldump — A Database Backup Program” .

Two functions providing basic XPath 1.0 (XML Path Language, version 1.0) capabilities are available. Some basic information about XPath syntax and usage is provided later in this section; however, an in-depth discussion of these topics is beyond the scope of this manual, and you should refer to the [XML Path Language (XPath) 1.0 standard](http://www.w3.org/TR/xpath) for definitive information. A useful resource for those new to XPath or who desire a refresher in the basics is the [Zvon.org XPath Tutorial](http://www.zvon.org/xxl/XPathTutorial/), which is available in several languages.

**Note**

These functions remain under development. We continue to improve these and other aspects of XML and XPath functionality in MySQL 8.0 and onwards. You may discuss these, ask questions about them, and obtain help from other users with them in the [MySQL XML User Forum](https://forums.mysql.com/list.php?44).

XPath expressions used with these functions support user variables and local stored program variables. User variables are weakly checked; variables local to stored programs are strongly checked (see also Bug #26518):

• **User** **variables** **(weak** **checking).** Variables using the syntax $@*variable\_name* (that is, user variables) are not checked. No warnings or errors are issued by the server if a variable has the wrong type or has previously not been assigned a value. This also means the user is fully responsible for any typographical errors, since no warnings are given if (for example)

$@myvairable is used where $@myvariable was intended.

Example:

mysql> **SET** **@xml** **=** **'<a><b>X</b><b>Y</b></a>';**

Query OK, 0 rows affected (0.00 sec)

mysql> **SET** **@i** **=1,** **@j** **=** **2;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **@i,** **ExtractValue(@xml,** **'//b[$@i]');**

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

| @i | ExtractValue(@xml, '//b[$@i]') |

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

| 1 | X |

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

1 row in set (0.00 sec)

mysql> **SELECT** **@j,** **ExtractValue(@xml,** **'//b[$@j]');**

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

| @j | ExtractValue(@xml, '//b[$@j]') |

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

| 2 | Y |

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

1 row in set (0.00 sec)

mysql> **SELECT** **@k,** **ExtractValue(@xml,** **'//b[$@k]');**

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

| @k | ExtractValue(@xml, '//b[$@k]') |

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

| NULL | |

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

1 row in set (0.00 sec)

• **Variables** **in** **stored** **programs** **(strong** **checking).** Variables using the syntax $*variable\_name* can be declared and used with these functions when they are called inside stored programs. Such variables are local to the stored program in which they are defined, and are strongly checked for type and value.

Example:

mysql> **DELIMITER** **|**

mysql> **CREATE** **PROCEDURE** **myproc** **()**

-> **BEGIN**

-> **DECLARE** **i** **INT** **DEFAULT** **1;**

-> **DECLARE** **xml** **VARCHAR(25)** **DEFAULT** **'<a>X</a><a>Y</a><a>Z</a>';**

->

-> **WHILE** **i** **<** **4** **DO**

-> **SELECT** **xml,** **i,** **ExtractValue(xml,** **'//a[$i]');**

-> **SET** **i** **=** **i+1;**

-> **END** **WHILE;**

-> **END** **|**

Query OK, 0 rows affected (0.01 sec)

mysql> **DELIMITER** **;**

mysql> **CALL** **myproc();**

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

| xml | i | ExtractValue(xml, '//a[$i]') |

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

| <a>X</a><a>Y</a><a>Z</a> | 1 | X |

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

1 row in set (0.00 sec)

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

| xml | i | ExtractValue(xml, '//a[$i]') |

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

| <a>X</a><a>Y</a><a>Z</a> | 2 | Y |

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

1 row in set (0.01 sec)

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

| xml | i | ExtractValue(xml, '//a[$i]') |

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

| <a>X</a><a>Y</a><a>Z</a> | 3 | Z |

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

1 row in set (0.01 sec)

**Parameters.** Variables used in XPath expressions inside stored routines that are passed in as parameters are also subject to strong checking.

Expressions containing user variables or variables local to stored programs must otherwise (except for notation) conform to the rules for XPath expressions containing variables as given in the XPath 1.0 specification.

**Note**

A user variable used to store an XPath expression is treated as an empty string. Because of this, it is not possible to store an XPath expression as a user variable. (Bug #32911)

• [ExtractValue(*xml\_frag*, *xpath\_expr*)](#_bookmark215)

[ExtractValue()](#_bookmark215) takes two string arguments, a fragment of XML markup *xml\_frag* and an XPath expression *xpath\_expr* (also known as a *locator*); it returns the text (CDATA) of the first text node which is a child of the element or elements matched by the XPath expression.

Using this function is the equivalent of performing a match using the *xpath\_expr* after appending /text(). In other words, [ExtractValue('<a><b>Sakila</b></a>', '/a/b')](#_bookmark215) and [ExtractValue('<a><b>Sakila</b></a>', '/a/b/text()')](#_bookmark215) produce the same result. If *xml\_frag* or *xpath\_expr* is NULL, the function returns NULL.

If multiple matches are found, the content of the first child text node of each matching element is returned (in the order matched) as a single, space-delimited string.

If no matching text node is found for the expression (including the implicit /text())—for whatever reason, as long as *xpath\_expr* is valid, and *xml\_frag* consists of elements which are properly nested and closed—an empty string is returned. No distinction is made between a match on an empty element and no match at all. This is by design.

If you need to determine whether no matching element was found in *xml\_frag* or such an element was found but contained no child text nodes, you should test the result of an expression that uses the XPath count() function. For example, both of these statements return an empty string, as shown here:

mysql> **SELECT** **ExtractValue('<a><b/></a>',** **'/a/b');**

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

| ExtractValue('<a><b/></a>', '/a/b') |

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

| |

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

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue('<a><c/></a>',** **'/a/b');**

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

| ExtractValue('<a><c/></a>', '/a/b') |

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

| |

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

1 row in set (0.00 sec)

However, you can determine whether there was actually a matching element using the following:

mysql> **SELECT** **ExtractValue('<a><b/></a>',** **'count(/a/b)');**

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

| ExtractValue('<a><b/></a>', 'count(/a/b)') |

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

| 1 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue('<a><c/></a>',** **'count(/a/b)');**

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

| ExtractValue('<a><c/></a>', 'count(/a/b)') |

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

| 0 |

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



1 row in set (0 .01 sec)

**Important**

[ExtractValue()](#_bookmark215) returns only CDATA, and does not return any tags that might be contained within a matching tag, nor any of their content (see the result returned as val1 in the following example).

mysql> **SELECT**

-> **ExtractValue('<a>ccc<b>ddd</b></a>',** **'/a')** **AS** **val1,**

-> **ExtractValue('<a>ccc<b>ddd</b></a>',** **'/a/b')** **AS** **val2,**

-> **ExtractValue('<a>ccc<b>ddd</b></a>',** **'//b')** **AS** **val3,**

-> **ExtractValue('<a>ccc<b>ddd</b></a>',** **'/b')** **AS** **val4,**

-> **ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>',** **'//b')** **AS** **val5;**

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

| val1 | val2 | val3 | val4 | val5 |

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

| ccc | ddd | ddd | | ddd eee |

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

This function uses the current SQL collation for making comparisons with contains(), performing the same collation aggregation as other string functions (such as [CONCAT()](#_bookmark117)), in taking into account the collation coercibility of their arguments; see Section 10.8.4, “Collation Coercibility in Expressions” , for an explanation of the rules governing this behavior.

(Previously, binary—that is, case-sensitive—comparison was always used.)

NULL is returned if *xml\_frag* contains elements which are not properly nested or closed, and a warning is generated, as shown in this example:

mysql> **SELECT** **ExtractValue('<a>c</a><b',** **'//a');**

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

| ExtractValue('<a>c</a><b', '//a') |

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

| NULL |

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

1 row in set, 1 warning (0.00 sec)

mysql> **SHOW** **WARNINGS\G**

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

Level: Warning

Code: 1525

Message: Incorrect XML value: 'parse error at line 1 pos 11:

END-OF-INPUT unexpected ('>' wanted)'

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue('<a>c</a><b/>',** **'//a');**

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

| ExtractValue('<a>c</a><b/>', '//a') |

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

| c |

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

1 row in set (0.00 sec)

• [UpdateXML(*xml\_target*, *xpath\_expr*, *new\_xml*)](#_bookmark216)

This function replaces a single portion of a given fragment of XML markup *xml\_target* with a new XML fragment *new\_xml*, and then returns the changed XML. The portion of *xml\_target* that is replaced matches an XPath expression *xpath\_expr* supplied by the user.

If no expression matching *xpath\_expr* is found, or if multiple matches are found, the function returns the original *xml\_target* XML fragment. All three arguments should be strings. If any of the arguments to UpdateXML() are NULL, the function returns NULL.

mysql> **SELECT**

-> **UpdateXML('<a><b>ccc</b><d></d></a>',** **'/a',** **'<e>fff</e>')** **AS** **val1,**



-> **UpdateXML('<a><b>ccc</b><d></d></a>',** **'/b',** **'<e>fff</e>')** **AS** **val2,**

-> **UpdateXML('<a><b>ccc</b><d></d></a>',** **'//b',** **'<e>fff</e>')** **AS** **val3,**

-> **UpdateXML('<a><b>ccc</b><d></d></a>',** **'/a/d',** **'<e>fff</e>')** **AS** **val4,**

-> **UpdateXML('<a><d></d><b>ccc</b><d></d></a>',** **'/a/d',** **'<e>fff</e>')** **AS** **val5**

-> **\G**

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

val1: <e>fff</e>

val2: <a><b>ccc</b><d></d></a>

val3: <a><e>fff</e><d></d></a>

val4: <a><b>ccc</b><e>fff</e></a>

val5: <a><d></d><b>ccc</b><d></d></a>

**Note**

A discussion in depth of XPath syntax and usage are beyond the scope of this manual. Please see the [XML Path Language (XPath) 1.0 specification](http://www.w3.org/TR/xpath) for definitive information. A useful resource for those new to XPath or who are wishing a refresher in the basics is the [Zvon.org XPath Tutorial](http://www.zvon.org/xxl/XPathTutorial/), which is available in several languages.

Descriptions and examples of some basic XPath expressions follow:

• /*tag*

Matches <*tag*/> if and only if <*tag*/> is the root element.

Example: /a has a match in <a><b/></a> because it matches the outermost (root) tag. It does not match the inner *a* element in <b><a/></b> because in this instance it is the child of another element.

• /*tag1*/*tag2*

Matches <*tag2*/> if and only if it is a child of <*tag1*/>, and <*tag1*/> is the root element.

Example: /a/b matches the *b* element in the XML fragment <a><b/></a> because it is a child of the root element *a*. It does not have a match in <b><a/></b> because in this case, *b* is the root element (and hence the child of no other element). Nor does the XPath expression have a match in <a><c><b/></c></a>; here, *b* is a descendant of *a*, but not actually a child of *a*.

This construct is extendable to three or more elements. For example, the XPath expression /a/b/c matches the *c* element in the fragment <a><b><c/></b></a>.

• //*tag*

Matches any instance of <*tag*>.

Example: //a matches the *a* element in any of the following: <a><b><c/></b></a>; <c><a><b/ ></a></b>; <c><b><a/></b></c>.

// can be combined with /. For example, //a/b matches the *b* element in either of the fragments <a><b/></a> or <c><a><b/></a></c>.

**Note**

//*tag* is the equivalent of /descendant-or-self::\*/*tag*. A common error is to confuse this with /descendant-or-self::*tag*, although the latter expression can actually lead to very different results, as can be seen here:

mysql> **SET** **@xml** **=** **'<a><b><c>w</c><b>x</b><d>y</d>z</b></a>';**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **@xml;**

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

| @xml |

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

| <a><b><c>w</c><b>x</b><d>y</d>z</b></a> |

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

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue(@xml,** **'//b[1]');**

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

| ExtractValue(@xml, '//b[1]') |

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

| x z |

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

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue(@xml,** **'//b[2]');**

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

| ExtractValue(@xml, '//b[2]') |

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

| |

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

1 row in set (0.01 sec)

mysql> **SELECT** **ExtractValue(@xml,** **'/descendant-or-self::\*/b[1]');**

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

| ExtractValue(@xml, '/descendant-or-self::\*/b[1]') |

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

| x z |

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

1 row in set (0.06 sec)

mysql> **SELECT** **ExtractValue(@xml,** **'/descendant-or-self::\*/b[2]');**

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

| ExtractValue(@xml, '/descendant-or-self::\*/b[2]') |

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

| |

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

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue(@xml,** **'/descendant-or-self::b[1]');**

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

| ExtractValue(@xml, '/descendant-or-self::b[1]') |

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

| z |

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

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue(@xml,** **'/descendant-or-self::b[2]');**

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

| ExtractValue(@xml, '/descendant-or-self::b[2]') |

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

| x |

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

1 row in set (0.00 sec)

• The \* operator acts as a “wildcard” that matches any element. For example, the expression /\*/b matches the *b* element in either of the XML fragments <a><b/></a> or <c><b/></c>. However, the expression does not produce a match in the fragment <b><a/></b> because *b* must be a child of some other element. The wildcard may be used in any position: The expression /\*/b/\* matches any child of a *b* element that is itself not the root element.

• You can match any of several locators using the | (UNION) operator. For example, the expression //b |//c matches all *b* and *c* elements in the XML target.

• It is also possible to match an element based on the value of one or more of its attributes. This done using the syntax *tag*[@*attribute*="*value*"]. For example, the expression //b[@id="idB"] matches the second *b* element in the fragment <a><b id="idA"/><c/><b id="idB"/></

a>. To match against *any* element having *attribute*="*value*", use the XPath expression // \*[*attribute*="*value*"].

To filter multiple attribute values, simply use multiple attribute-comparison clauses in succession. For example, the expression //b[@c="x"][@d="y"] matches the element <b c="x" d="y"/> occurring anywhere in a given XML fragment.

To find elements for which the same attribute matches any of several values, you can use multiple locators joined by the | operator. For example, to match all *b* elements whose *c* attributes have either of the values 23 or 17, use the expression //b[@c="23"] |//b[@c="17"]. You can also use the logical or operator for this purpose: //b[@c="23" or @c="17"].

**Note**

The difference between or and | is that orjoins conditions, while |joins result sets.

**XPath** **Limitations.** The XPath syntax supported by these functions is currently subject to the

following limitations:

• Nodeset-to-nodeset comparison (such as '/a/b[@c=@d]') is not supported.

• All of the standard XPath comparison operators are supported. (Bug #22823)

• Relative locator expressions are resolved in the context of the root node. For example, consider the following query and result:

mysql> **SELECT** **ExtractValue(**

-> **'<a><b** **c="1">X</b><b** **c="2">Y</b></a>',**

-> **'a/b'**

-> **)** **AS** **result;**

+--------+

| result |

+--------+

| X Y |

+--------+

1 row in set (0.03 sec)

In this case, the locator a/b resolves to /a/b.

Relative locators are also supported within predicates. In the following example, d[../@c="1"] is resolved as /a/b[@c="1"]/d:

mysql> **SELECT** **ExtractValue(**

->

**'<a>**

->

**<b** **c="1"><d>X</d></b>**

**<b** **c="2"><d>X</d></b>**

->

->

**</a>',**

->

**'a/b/d[../@c="1"]')**

-> **AS** **result;**

+--------+

| result |

+--------+

| X |

+--------+

1 row in set (0.00 sec)

• Locators prefixed with expressions that evaluate as scalar values—including variable references, literals, numbers, and scalar function calls—are not permitted, and their use results in an error.

• The :: operator is not supported in combination with node types such as the following:

• *axis*::comment()

• *axis*::text()

• *axis*::processing-instructions()

• *axis*::node()

However, name tests (such as *axis*::*name* and *axis*::\*) are supported, as shown in these examples:

mysql> **SELECT** **ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b');**

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

| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b') |

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

| x |

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

1 row in set (0.02 sec)

mysql> **SELECT** **ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::\*');**

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

| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::\*') |

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

| x y |

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

1 row in set (0.01 sec)

• “Up-and-down” navigation is not supported in cases where the path would lead “above” the root element. That is, you cannot use expressions which match on descendants of ancestors of a given element, where one or more of the ancestors of the current element is also an ancestor of the root element (see Bug #16321).

• The following XPath functions are not supported, or have known issues as indicated:

• id()

• lang()

• local-name()

• name()

• namespace-uri()

• normalize-space()

• starts-with()

• string()

• substring-after()

• substring-before()

• translate()

• The following axes are not supported:

• following-sibling

• following

• preceding-sibling

• preceding

XPath expressions passed as arguments to [ExtractValue()](#_bookmark215) and [UpdateXML()](#_bookmark216) may contain the colon character (:) in element selectors, which enables their use with markup employing XML namespaces notation. For example:

mysql> **SET** **@xml** **=** **'<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';**



Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **ExtractValue(@xml,** **'//e:f');**

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

| ExtractValue(@xml, '//e:f') |

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

| 444 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **UpdateXML(@xml,** **'//b:c',** **'<g:h>555</g:h>');**

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

| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |

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

| <a>111<g:h>555</g:h></a> |

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

1 row in set (0.00 sec)

This is similar in some respects to what is permitted by [Apache Xalan](http://xalan.apache.org/) and some other parsers, and is much simpler than requiring namespace declarations or the use of the namespace-uri() and local-name() functions.

**Error** **handling.** For both [ExtractValue()](#_bookmark215) and [UpdateXML()](#_bookmark216), the XPath locator used must be valid and the XML to be searched must consist of elements which are properly nested and closed. If the locator is invalid, an error is generated:

mysql> **SELECT** **ExtractValue('<a>c</a><b/>',** **'/&a');**

ERROR 1105 (HY000): XPATH syntax error: '&a'

If *xml\_frag* does not consist of elements which are properly nested and closed, NULL is returned and a warning is generated, as shown in this example:

mysql> **SELECT** **ExtractValue('<a>c</a><b',** **'//a');**

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

| ExtractValue('<a>c</a><b', '//a') |

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

| NULL |

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

1 row in set, 1 warning (0.00 sec)

mysql> **SHOW** **WARNINGS\G**

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

Level: Warning

Code: 1525

Message: Incorrect XML value: 'parse error at line 1 pos 11:

END-OF-INPUT unexpected ('>' wanted)'

1 row in set (0.00 sec)

mysql> **SELECT** **ExtractValue('<a>c</a><b/>',** **'//a');**

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

| ExtractValue('<a>c</a><b/>', '//a') |

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

| c |

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

1 row in set (0.00 sec)

**Important**

The replacement XML used as the third argument to [UpdateXML()](#_bookmark216) is *not* checked to determine whether it consists solely of elements which are properly nested and closed.

**XPath** **Injection.** *code* *injection* occurs when malicious code is introduced into the system to gain unauthorized access to privileges and data. It is based on exploiting assumptions made by developers about the type and content of data input from users. XPath is no exception in this regard.

A common scenario in which this can happen is the case of application which handles authorization by matching the combination of a login name and password with those found in an XML file, using an XPath expression like this one:

//user[login/text()='neapolitan' and password/text()='1c3cr34m']/attribute::id

This is the XPath equivalent of an SQL statement like this one:

SELECT id FROM users WHERE login='neapolitan' AND password='1c3cr34m';

A PHP application employing XPath might handle the login process like this:

<?php

$file = "users.xml";

$login = $POST["login"];

$password = $POST["password"];

$xpath = "//user[login/text()=$login and password/text()=$password]/attribute::id";

if( file\_exists($file) )

{

$xml = simplexml\_load\_file($file);

if($result = $xml->xpath($xpath))

echo "You are now logged in as user $result[0] .";

else

echo "Invalid login name or password .";

}

else

exit("Failed to open $file.");

?>

No checks are performed on the input. This means that a malevolent user can “short-circuit” the test by entering ' or 1=1 for both the login name and password, resulting in $xpath being evaluated as shown here:

//user[login/text()='' or 1=1 and password/text()='' or 1=1]/attribute::id

Since the expression inside the square brackets always evaluates as true, it is effectively the same as this one, which matches the id attribute of every user element in the XML document:

//user/attribute::id

One way in which this particular attack can be circumvented is simply by quoting the variable names to be interpolated in the definition of $xpath, forcing the values passed from a Web form to be converted to strings:

$xpath = "//user[login/text()='$login' and password/text()='$password']/attribute::id";

This is the same strategy that is often recommended for preventing SQL injection attacks. In general, the practices you should follow for preventing XPath injection attacks are the same as for preventing SQL injection:

• Never accepted untested data from users in your application.

• Check all user-submitted data for type; reject or convert data that is of the wrong type

• Test numeric data for out of range values; truncate, round, or reject values that are out of range. Test strings for illegal characters and either strip them out or reject input containing them.

• Do not output explicit error messages that might provide an unauthorized user with clues that could be used to compromise the system; log these to a file or database table instead.

Just as SQL injection attacks can be used to obtain information about database schemas, so can XPath injection be used to traverse XML files to uncover their structure, as discussed in Amit Klein's paper [Blind XPath Injection](http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf) (PDF file, 46KB).

It is also important to check the output being sent back to the client. Consider what can happen when we use the MySQL [ExtractValue()](#_bookmark215) function:

mysql> **SELECT** **ExtractValue(**

->

->

-> **)** **AS** **id;**

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

| id |

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

| 00327 13579 02403 42354 28570 |

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

1 row in set (0.01 sec)

**LOAD\_FILE('users.xml'),**

**'//user[login/text()=""** **or** **1=1** **and** **password/text()=""** **or** **1=1]/attribute::id'**

Because [ExtractValue()](#_bookmark215) returns multiple matches as a single space-delimited string, this injection attack provides every valid ID contained within users.xml to the user as a single row of output. As an extra safeguard, you should also test output before returning it to the user. Here is a simple example:

mysql> **SELECT** **@id** **=** **ExtractValue(**

-> **LOAD\_FILE('users** **.xml'),**

-> **'//user[login/text()=""** **or** **1=1** **and** **password/text()=""** **or** **1=1]/attribute::id'**

-> **);**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **IF(**

->

->

->

-> **AS** **singleID;**

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

| singleID |

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

| Unable to retrieve user ID |

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

1 row in set (0.00 sec)

**INSTR(@id,** **'** **')** **=** **0,**

**@id,**

**'Unable** **to** **retrieve** **user** **ID')**

In general, the guidelines for returning data to users securely are the same as for accepting user input. These can be summed up as:

• Always test outgoing data for type and permissible values.

• Never permit unauthorized users to view error messages that might provide information about the application that could be used to exploit it.

**12.13** **Bit** **Functions** **and** **Operators**

**Table** **12.17** **Bit** **Functions** **and** **Operators**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [&](#_bookmark217) | Bitwise AND |
| [>>](#_bookmark218) | Right shift |
| [<<](#_bookmark219) | Left shift |
| [^](#_bookmark220) | Bitwise XOR |
| [BIT\_COUNT()](#_bookmark221) | Return the number of bits that are set |
| [|](#_bookmark222) | Bitwise OR |
| [~](#_bookmark223) | Bitwise inversion |

Bit functions and operators comprise [BIT COUNT()](#_bookmark221)\_, BIT\_AND(), BIT\_OR(), BIT\_XOR(), [&](#_bookmark217), [|](#_bookmark222), [^](#_bookmark220), [~](#_bookmark223), [<<](#_bookmark219), and [>>](#_bookmark218). (The BIT\_AND(), BIT\_OR(), and BIT\_XOR() aggregate functions are described in Section 12.20.1, “Aggregate Function Descriptions” .) Prior to MySQL 8.0, bit functions and operators required BIGINT (64-bit integer) arguments and returned BIGINT values, so they had a maximum range of 64 bits. Non-BIGINT arguments were converted to BIGINT prior to performing the operation and truncation could occur.

In MySQL 8.0, bit functions and operators permit binary string type arguments (BINARY, VARBINARY, and the BLOB types) and return a value of like type, which enables them to take arguments and produce return values larger than 64 bits. Nonbinary string arguments are converted to BIGINT and processed as such, as before.

An implication of this change in behavior is that bit operations on binary string arguments might produce a different result in MySQL 8.0 than in 5.7. For information about how to prepare in MySQL 5.7 for potential incompatibilities between MySQL 5.7 and 8.0, see [Bit Functions Operators](https://dev.mysql.com/doc/refman/5.7/en/bit-functions.html)and, in [MySQL](https://dev.mysql.com/doc/refman/5.7/en/) [5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

• [Bit Operations Prior to MySQL 8.0](#_bookmark224)

• [Bit Operations in MySQL 8.0](#_bookmark225)

• [Binary String Bit-Operation Examples](#_bookmark226)

• [Bitwise AND, OR, and XOR Operations](#_bookmark227)

• [Bitwise Complement and Shift Operations](#_bookmark228)

• [BIT\_COUNT() Operations](#_bookmark229)

• [BIT\_AND(), BIT\_OR(), and BIT\_XOR() Operations](#_bookmark230)

• [Special Handling of Hexadecimal Literals, Bit Literals, and NULL Literals](#_bookmark231)

• [Bit-Operation Incompatibilities with MySQL 5.7](#_bookmark232)

**Bit** **Operations** **Prior** **to** **MySQL** **8.0**

Bit operations prior to MySQL 8.0 handle only unsigned 64-bit integer argument and result values (that is, unsigned BIGINT values). Conversion of arguments of other types to BIGINT occurs as necessary. Examples:

• This statement operates on numeric literals, treated as unsigned 64-bit integers:

mysql> **SELECT** **127** **|** **128,** **128** **<<** **2,** **BIT\_COUNT(15);**

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

| 127 | 128 | 128 << 2 | BIT\_COUNT(15) |

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

| 255 | 512 | 4 |

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

• This statement performs to-number conversions on the string arguments ( '127' to 127, and so forth) before performing the same operations as the first statement and producing the same results:

mysql> **SELECT** **'127'** **|** **'128',** **'128'** **<<** **2,** **BIT\_COUNT('15');**

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

| '127' | '128' | '128' << 2 | BIT\_COUNT('15') |

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

| 255 | 512 | 4 |

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

• This statement uses hexadecimal literals for the bit-operation arguments. MySQL by default treats hexadecimal literals as binary strings, but in numeric context evaluates them as numbers (see Section 9.1.4, “Hexadecimal Literals” ). Prior to MySQL 8.0, numeric context includes bit operations. Examples:

mysql> **SELECT** **X'7F'** **|** **X'80',** **X'80'** **<<** **2,** **BIT\_COUNT(X'0F');**

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

| X'7F' | X'80' | X'80' << 2 | BIT\_COUNT(X'0F') |

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

| 255 | 512 | 4 |

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

Handling of bit-value literals in bit operations is similar to hexadecimal literals (that is, as numbers).

**Bit** **Operations** **in** **MySQL** **8.0**

MySQL 8.0 extends bit operations to handle binary string arguments directly (without conversion) and produce binary string results. (Arguments that are not integers or binary strings are still converted to integers, as before.) This extension enhances bit operations in the following ways:

• Bit operations become possible on values longer than 64 bits.

• It is easier to perform bit operations on values that are more naturally represented as binary strings than as integers.

For example, consider UUID values and IPv6 addresses, which have human-readable text formats like this:

UUID: 6ccd780c-baba-1026-9564-5b8c656024db

IPv6: fe80::219:d1ff:fe91:1a72

It is cumbersome to operate on text strings in those formats. An alternative is convert them to fixed- length binary strings without delimiters. UUID\_TO\_BIN() and INET6\_ATON() each produce a value of data type BINARY(16), a binary string 16 bytes (128 bits) long. The following statements illustrate this (HEX() is used to produce displayable values):

mysql> **SELECT** **HEX(UUID\_TO\_BIN('6ccd780c-baba-1026-9564-5b8c656024db'));**

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

| HEX(UUID\_TO\_BIN('6ccd780c-baba-1026-9564-5b8c656024db')) |

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

| 6CCD780CBABA102695645B8C656024DB |

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

mysql> **SELECT** **HEX(INET6\_ATON('fe80::219:d1ff:fe91:1a72'));**

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

| HEX(INET6\_ATON('fe80::219:d1ff:fe91:1a72')) |

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

| FE800000000000000219D1FFFE911A72 |

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

Those binary values are easily manipulable with bit operations to perform actions such as extracting the timestamp from UUID values, or extracting the network and host parts of IPv6 addresses. (For examples, see later in this discussion.)

Arguments that count as binary strings include column values, routine parameters, local variables, and user-defined variables that have a binary string type: BINARY, VARBINARY, or one of the BLOB types.

What about hexadecimal literals and bit literals? Recall that those are binary strings by default in MySQL, but numbers in numeric context. How are they handled for bit operations in MySQL 8.0? Does MySQL continue to evaluate them in numeric context, as is done prior to MySQL 8.0? Or do bit operations evaluate them as binary strings, now that binary strings can be handled “natively” without conversion?

Answer: It has been common to specify arguments to bit operations using hexadecimal literals or bit literals with the intent that they represent numbers, so MySQL continues to evaluate bit operations in numeric context when all bit arguments are hexadecimal or bit literals, for backward compatbility. If you require evaluation as binary strings instead, that is easily accomplished: Use the \_binary introducer for at least one literal.

• These bit operations evaluate the hexadecimal literals and bit literals as integers:

mysql> **SELECT** **X'40'** **|** **X'01',** **b'11110001'** **&** **b'01001111';**

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

| X'40' | X'01' | b'11110001' & b'01001111' |

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

| 65 | 65 |

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

• These bit operations evaluate the hexadecimal literals and bit literals as binary strings, due to the \_binary introducer:

mysql> **SELECT** **\_binary** **X'40'** **|** **X'01',** **b'11110001'** **&** **\_binary** **b'01001111';**

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

| \_binary X'40' | X'01' | b'11110001' & \_binary b'01001111' |

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

| A | A |

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

Although the bit operations in both statements produce a result with a numeric value of 65, the second statement operates in binary-string context, for which 65 is ASCII A.

In numeric evaluation context, permitted values of hexadecimal literal and bit literal arguments have a maximum of 64 bits, as do results. By contrast, in binary-string evaluation context, permitted arguments (and results) can exceed 64 bits:

mysql> **SELECT** **\_binary** **X'4040404040404040'** **|** **X'0102030405060708';**

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

| \_binary X'4040404040404040' | X'0102030405060708' |

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

|  |  |
| --- | --- |
| | ABCDEFGH | | |

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

There are several ways to refer to a hexadecimal literal or bit literal in a bit operation to cause binary- string evaluation:

\_binary *literal*

BINARY *literal*

CAST(*literal* AS BINARY)

Another way to produce binary-string evaluation of hexadecimal literals or bit literals is to assign them to user-defined variables, which results in variables that have a binary string type:

mysql> **SET** **@v1** **=** **X'40',** **@v2** **=** **X'01',** **@v3** **=** **b'11110001',** **@v4** **=** **b'01001111';**

mysql> **SELECT** **@v1** **|** **@v2,** **@v3** **&** **@v4;**

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

| @v1 | @v2 | @v3 & @v4 |

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

| A | A |

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

In binary-string context, bitwise operation arguments must have the same length or an [ER\_INVALID\_BITWISE\_OPERANDS\_SIZE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_bitwise_operands_size) error occurs:

mysql> **SELECT** **\_binary** **X'40'** **|** **X'0001';**

ERROR 3513 (HY000): Binary operands of bitwise

operators must be of equal length

To satisfy the equal-length requirement, pad the shorter value with leading zero digits or, if the longer value begins with leading zero digits and a shorter result value is acceptable, strip them:

mysql> **SELECT** **\_binary** **X'0040'** **|** **X'0001';**

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

| \_binary X'0040' | X'0001' |

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

| A |

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

mysql> **SELECT** **\_binary** **X'40'** **|** **X'01';**

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

| \_binary X'40' | X'01' |

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

| A |

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

Padding or stripping can also be accomplished using functions such as [LPAD()](#_bookmark133), [RPAD()](#_bookmark154), [SUBSTR()](#_bookmark160), or [CAST()](#_bookmark109). In such cases, the expression arguments are no longer all literals and \_binary becomes unnecessary. Examples:

mysql> **SELECT** **LPAD(X'40',** **2,** **X'00')** **|** **X'0001';**

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

| LPAD(X'40', 2, X'00') | X'0001' |

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

| A |

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

mysql> **SELECT** **X'40'** **|** **SUBSTR(X'0001',** **2,** **1);**

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

| X'40' | SUBSTR(X'0001', 2, 1) |

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

| A |

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

**Binary** **String** **Bit-Operation** **Examples**

The following example illustrates use of bit operations to extract parts of a UUID value, in this case, the timestamp and IEEE 802 node number. This technique requires bitmasks for each extracted part.

Convert the text UUID to the corresponding 16-byte binary value so that it can be manipulated using bit operations in binary-string context:

mysql> **SET** **@uuid** **=** **UUID\_TO\_BIN('6ccd780c-baba-1026-9564-5b8c656024db');**

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

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

| HEX(@uuid) |

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

| 6CCD780CBABA102695645B8C656024DB |

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

Construct bitmasks for the timestamp and node number parts of the value. The timestamp comprises the first three parts (64 bits, bits 0 to 63) and the node number is the last part (48 bits, bits 80 to 127):

mysql> **SET** **@ts\_mask** **=** **CAST(X'FFFFFFFFFFFFFFFF'** **AS** **BINARY(16));**

mysql> **SET** **@node\_mask** **=** **CAST(X'FFFFFFFFFFFF'** **AS** **BINARY(16))** **>>** **80;**

mysql> **SELECT** **HEX(@ts\_mask);**

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

| HEX(@ts\_mask) |

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

| FFFFFFFFFFFFFFFF0000000000000000 |

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

mysql> **SELECT** **HEX(@node\_mask);**

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

| HEX(@node\_mask) |

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

| 00000000000000000000FFFFFFFFFFFF |

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

The CAST(... AS BINARY(16)) function is used here because the masks must be the same length as the UUID value against which they are applied. The same result can be produced using other functions to pad the masks to the required length:

SET @ts\_mask= RPAD(X'FFFFFFFFFFFFFFFF' , 16, X'00');

SET @node\_mask = LPAD(X'FFFFFFFFFFFF', 16, X'00') ;

Use the masks to extract the timestamp and node number parts:

mysql> **SELECT** **HEX(@uuid** **&** **@ts\_mask)** **AS** **'timestamp** **part';**

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

| timestamp part |

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

| 6CCD780CBABA10260000000000000000 |

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

mysql> **SELECT** **HEX(@uuid** **&** **@node\_mask)** **AS** **'node** **part';**

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

| node part |

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

| 000000000000000000005B8C656024DB |

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

The preceding example uses these bit operations: right shift ([>>](#_bookmark218)) and bitwise AND ([&](#_bookmark217)).



**Note**

UUID\_TO\_BIN() takes a flag that causes some bit rearrangement in the resulting binary UUID value. If you use that flag, modify the extraction masks accordingly.

The next example uses bit operations to extract the network and host parts of an IPv6 address. Suppose that the network part has a length of 80 bits. Then the host part has a length of 128 − 80 = 48 bits. To extract the network and host parts of the address, convert it to a binary string, then use bit operations in binary-string context.

Convert the text IPv6 address to the corresponding binary string:

mysql> **SET** **@ip** **=** **INET6\_ATON('fe80::219:d1ff:fe91:1a72');**

Define the network length in bits:

mysql> **SET** **@net\_len** **=** **80;**

Construct network and host masks by shifting the all-ones address left or right. To do this, begin with the address ::, which is shorthand for all zeros, as you can see by converting it to a binary string like this:

mysql> **SELECT** **HEX(INET6\_ATON('::'))** **AS** **'all** **zeros';**

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

| all zeros |

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

| 00000000000000000000000000000000 |

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

To produce the complementary value (all ones), use the [~](#_bookmark223) operator to invert the bits:

mysql> **SELECT** **HEX(~INET6\_ATON('::'))** **AS** **'all** **ones';**

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

| all ones |

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

| FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF |

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

Shift the all-ones value left or right to produce the network and host masks:

mysql> **SET** **@net\_mask** **=** **~INET6\_ATON('::')** **<<** **(128** **-** **@net\_len);**

mysql> **SET** **@host\_mask** **=** **~INET6\_ATON('::')** **>>** **@net\_len;**

Display the masks to verify that they cover the correct parts of the address:

mysql> **SELECT** **INET6\_NTOA(@net\_mask)** **AS** **'network** **mask';**

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

| network mask |

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

| ffff:ffff:ffff:ffff:ffff:: |

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

mysql> **SELECT** **INET6\_NTOA(@host\_mask)** **AS** **'host** **mask';**

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

| host mask |

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

| ::ffff:255 .255 .255 .255 |

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

Extract and display the network and host parts of the address:

mysql> **SET** **@net\_part** **=** **@ip** **&** **@net\_mask;**

mysql> **SET** **@host\_part** **=** **@ip** **&** **@host\_mask;**

mysql> **SELECT** **INET6\_NTOA(@net\_part)** **AS** **'network** **part';**

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

| network part |

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

| fe80::219:0:0:0 |

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

mysql> **SELECT** **INET6\_NTOA(@host\_part)** **AS** **'host** **part';**

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

| host part |

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

| ::d1ff:fe91:1a72 |

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

The preceding example uses these bit operations: Complement ([~](#_bookmark223)), left shift ([<<](#_bookmark219)), and bitwise AND ([&](#_bookmark217)).

The remaining discussion provides details on argument handling for each group of bit operations, more information about literal-value handling in bit operations, and potential incompatibilities between MySQL 8.0 and older MySQL versions.

**Bitwise** **AND,** **OR,** **and** **XOR** **Operations**

For [&](#_bookmark217), [|](#_bookmark222), and [^](#_bookmark220) bit operations, the result type depends on whether the arguments are evaluated as binary strings or numbers:

• Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.

• Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an [ER\_INVALID\_BITWISE\_OPERANDS\_SIZE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_bitwise_operands_size) error occurs. Numeric evaluation produces an unsigned 64-bit integer.

Examples of numeric evaluation:

mysql> **SELECT** **64** **|** **1,** **X'40'** **|** **X'01';**

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

| 64 | 1 | X'40' | X'01' |

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

| 65 | 65 |

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

Examples of binary-string evaluation:

mysql> **SELECT** **\_binary** **X'40'** **|** **X'01';**

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

| \_binary X'40' | X'01' |

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

| A |

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

mysql> **SET** **@var1** **=** **X'40',** **@var2** **=** **X'01';**

mysql> **SELECT** **@var1** **|** **@var2;**

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

| @var1 | @var2 |

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

| A |

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

**Bitwise** **Complement** **and** **Shift** **Operations**

For [~](#_bookmark223), [<<](#_bookmark219), and [>>](#_bookmark218) bit operations, the result type depends on whether the bit argument is evaluated as a binary string or number:

• Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.

• Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

For shift operations, bits shifted off the end of the value are lost without warning, regardless of the argument type. In particular, if the shift count is greater or equal to the number of bits in the bit argument, all bits in the result are 0.

Examples of numeric evaluation:

mysql> **SELECT** **~0,** **64** **<<** **2,** **X'40'** **<<** **2;**

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

| ~0 | 64 << 2 | X'40' << 2 |

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

| 18446744073709551615 | 256 | 256 |

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

Examples of binary-string evaluation:

mysql> **SELECT** **HEX(\_binary** **X'1111000022220000'** **>>** **16);**

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

| HEX(\_binary X'1111000022220000' >> 16) |

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

| 0000111100002222 |

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

mysql> **SELECT** **HEX(\_binary** **X'1111000022220000'** **<<** **16);**

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

| HEX(\_binary X'1111000022220000' << 16) |

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

| 0000222200000000 |

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

mysql> **SET** **@var1** **=** **X'F0F0F0F0';**

mysql> **SELECT** **HEX(~@var1);**

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

| HEX(~@var1) |

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

| 0F0F0F0F |

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

**BIT\_COUNT()** **Operations**

The [BIT\_COUNT()](#_bookmark221) function always returns an unsigned 64-bit integer, or NULL if the argument is

NULL.

mysql> **SELECT** **BIT\_COUNT(127);**

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

| BIT\_COUNT(127) |

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

| 7 |

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

mysql> **SELECT** **BIT\_COUNT(b'010101'),** **BIT\_COUNT(\_binary** **b'010101');**

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

| BIT\_COUNT(b'010101') | BIT\_COUNT(\_binary b'010101') |

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

| 3 | 3 |

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

**BIT\_AND(),** **BIT\_OR(),** **and** **BIT\_XOR()** **Operations**

For the BIT\_AND(), BIT\_OR(), and BIT\_XOR() bit functions, the result type depends on whether the function argument values are evaluated as binary strings or numbers:

• Binary-string evaluation occurs when the argument values have a binary string type, and the argument is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument value conversion to unsigned 64-bit integers as necessary.

• Binary-string evaluation produces a binary string of the same length as the argument values. If argument values have unequal lengths, an [ER\_INVALID\_BITWISE\_OPERANDS\_SIZE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_bitwise_operands_size) error occurs. If the argument size exceeds 511 bytes, an [ER\_INVALID\_BITWISE\_AGGREGATE\_OPERANDS\_SIZE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_bitwise_aggregate_operands_size) error occurs. Numeric evaluation produces an unsigned 64-bit integer.

NULL values do not affect the result unless all values are NULL. In that case, the result is a neutral value having the same length as the length of the argument values (all bits 1 for BIT\_AND(), all bits 0 for BIT\_OR(), and BIT\_XOR()).

**FROM** **t** **GROUP** **BY** **group\_id;**

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

| HEX(BIT\_AND(a)) | HEX(BIT\_OR(a)) | HEX(BIT\_XOR(a)) |

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

|

|

|

|

|

|

|

|

000000000000

FF34

000000000000

ED00

FFFFFFFFFFFF

1234

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

Example:

**TABLE** **t** **(group\_id** **INT,** **a** **VARBINARY(6));**

**INTO** **t** **VALUES** **(1,** **NULL);**

**INTO** **t** **VALUES** **(1,** **NULL);**

**INTO** **t** **VALUES** **(2,** **NULL);**

**INTO** **t** **VALUES** **(2,** **X'1234');**

**INTO** **t** **VALUES** **(2,** **X'FF34');**

**HEX(BIT\_AND(a)),** **HEX(BIT\_OR(a)),** **HEX(BIT\_XOR(a))**

mysql> **CREATE**

mysql> **INSERT**

mysql> **INSERT**

mysql> **INSERT**

mysql> **INSERT**

mysql> **INSERT**

mysql> **SELECT**

**Special** **Handling** **of** **Hexadecimal** **Literals,** **Bit** **Literals,** **and** **NULL** **Literals**

For backward compatibility, MySQL 8.0 evaluates bit operations in numeric context when all bit arguments are hexadecimal literals, bit literals, or NULL literals. That is, bit operations on binary-string bit arguments do not use binary-string evaluation if all bit arguments are unadorned hexadecimal literals, bit literals, or NULL literals. (This does not apply to such literals if they are written with a \_binary introducer, [BINARY](#_bookmark200) operator, or other way of specifying them explicitly as binary strings.)

The literal handling just described is the same as prior to MySQL 8.0. Examples:

• These bit operations evaluate the literals in numeric context and produce a BIGINT result:

b'0001' | b'0010'

X'0008' << 8

• These bit operations evaluate NULL in numeric context and produce a BIGINT result that has a NULL value:

NULL & NULL

NULL >> 4

In MySQL 8.0, you can cause those operations to evaluate the arguments in binary-string context by indicating explicitly that at least one argument is a binary string:

\_binary b'0001' | b'0010'

\_binary X'0008' << 8

BINARY NULL & NULL

BINARY NULL >> 4

The result of the last two expressions is NULL, just as without the BINARY operator, but the data type of the result is a binary string type rather than an integer type.

**Bit-Operation** **Incompatibilities** **with** **MySQL** **5.7**

Because bit operations can handle binary string arguments natively in MySQL 8.0, some expressions produce a different result in MySQL 8.0 than in 5.7. The five problematic expression types to watch out for are:

*nonliteral\_binary* { & | ^ } *binary*

*binary* { & | ^ } *nonliteral\_binary*

*nonliteral\_binary* { << >> } *anything*

~ *nonliteral\_binary*

*AGGR\_BIT\_FUNC* (*nonliteral\_binary*)

Those expressions return BIGINT in MySQL 5.7, binary string in 8.0.

Explanation of notation:

• { *op1* *op2* ... }: List of operators that apply to the given expression type.

• *binary*: Any kind of binary string argument, including a hexadecimal literal, bit literal, or NULL literal.

• *nonliteral\_binary*: An argument that is a binary string value other than a hexadecimal literal, bit literal, or NULL literal.

• *AGGR\_BIT\_FUNC*: An aggregate function that takes bit-value arguments: BIT\_AND(), BIT\_OR(), BIT\_XOR().

For information about how to prepare in MySQL 5.7 for potential incompatibilities between MySQL 5.7 and 8.0, see [Bit Functions and Operators](https://dev.mysql.com/doc/refman/5.7/en/bit-functions.html), in [MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

The following list describes available bit functions and operators:

• [|](#_bookmark222)

Bitwise OR.

The result type depends on whether the arguments are evaluated as binary strings or numbers:

• Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.

• Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an [ER\_INVALID\_BITWISE\_OPERANDS\_SIZE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_bitwise_operands_size) error occurs. Numeric evaluation produces an unsigned 64-bit integer.

For more information, see the introductory discussion in this section.

mysql> **SELECT** **29** **|** **15;**

-> 31

mysql> **SELECT** **\_binary** **X'40404040'** **|** **X'01020304';**

-> 'ABCD'

If bitwise OR is invoked from within the mysql client, binary string results display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [&](#_bookmark217)

Bitwise AND.

The result type depends on whether the arguments are evaluated as binary strings or numbers:

• Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.

• Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an [ER\_INVALID\_BITWISE\_OPERANDS\_SIZE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_bitwise_operands_size) error occurs. Numeric evaluation produces an unsigned 64-bit integer.

For more information, see the introductory discussion in this section.

mysql> **SELECT** **29** **&** **15;**

-> 13

mysql> **SELECT** **HEX(\_binary** **X'FF'** **&** **b'11110000');**

-> 'F0'

If bitwise AND is invoked from within the mysql client, binary string results display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [^](#_bookmark220)

Bitwise XOR.

The result type depends on whether the arguments are evaluated as binary strings or numbers:

• Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.

• Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an [ER\_INVALID\_BITWISE\_OPERANDS\_SIZE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_bitwise_operands_size) error occurs. Numeric evaluation produces an unsigned 64-bit integer.

For more information, see the introductory discussion in this section.

mysql> **SELECT** **1** **^** **1;**

-> 0

mysql> **SELECT** **1** **^** **0;**

-> 1

mysql> **SELECT** **11** **^** **3;**

-> 8

mysql> **SELECT** **HEX(\_binary** **X'FEDC'** **^** **X'1111');**

-> 'EFCD'

If bitwise XOR is invoked from within the mysql client, binary string results display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [<<](#_bookmark219)

Shifts a longlong (BIGINT) number or binary string to the left.

The result type depends on whether the bit argument is evaluated as a binary string or number:

• Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.

• Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

Bits shifted off the end of the value are lost without warning, regardless of the argument type. In particular, if the shift count is greater or equal to the number of bits in the bit argument, all bits in the result are 0.

For more information, see the introductory discussion in this section.

mysql> **SELECT** **1** **<<** **2;**

-> 4

mysql> **SELECT** **HEX(\_binary** **X'00FF00FF00FF'** **<<** **8);**

-> 'FF00FF00FF00'

If a bit shift is invoked from within the mysql client, binary string results display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [>>](#_bookmark218)

Shifts a longlong (BIGINT) number or binary string to the right.

The result type depends on whether the bit argument is evaluated as a binary string or number:

• Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.

• Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

Bits shifted off the end of the value are lost without warning, regardless of the argument type. In particular, if the shift count is greater or equal to the number of bits in the bit argument, all bits in the result are 0.

For more information, see the introductory discussion in this section.

mysql> **SELECT** **4** **>>** **2;**

-> 1

mysql> **SELECT** **HEX(\_binary** **X'00FF00FF00FF'** **>>** **8);**

-> '0000FF00FF00'

If a bit shift is invoked from within the mysql client, binary string results display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [~](#_bookmark223)

Invert all bits.

The result type depends on whether the bit argument is evaluated as a binary string or number:

• Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or NULL literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.

• Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

For more information, see the introductory discussion in this section.

mysql> **SELECT** **5** **&** **~1;**

-> 4

mysql> **SELECT** **HEX(~X'0000FFFF1111EEEE');**

-> 'FFFF0000EEEE1111'

If bitwise inversion is invoked from within the mysql client, binary string results display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [BIT\_COUNT(*N*)](#_bookmark221)

Returns the number of bits that are set in the argument *N*as an unsigned 64-bit integer, or NULL if the argument is NULL.

mysql> **SELECT** **BIT\_COUNT(64),** **BIT\_COUNT(BINARY** **64);**

-> 1, 7

mysql> **SELECT** **BIT\_COUNT('64'),** **BIT\_COUNT(\_binary** **'64');**

-> 1, 7

mysql> **SELECT** **BIT\_COUNT(X'40'),** **BIT\_COUNT(\_binary** **X'40');**

-> 1, 1

**12.14** **Encryption** **and** **Compression** **Functions**

**Table** **12.18** **Encryption** **Functions**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [AES\_DECRYPT()](#_bookmark233) | Decrypt using AES |
| [AES\_ENCRYPT()](#_bookmark234) | Encrypt using AES |
| [COMPRESS()](#_bookmark235) | Return result as a binary string |
| [MD5()](#_bookmark236) | Calculate MD5 checksum |
| [RANDOM\_BYTES()](#_bookmark237) | Return a random byte vector |
| [SHA1(), SHA()](#_bookmark238) | Calculate an SHA-1 160-bit checksum |
| [SHA2()](#_bookmark239) | Calculate an SHA-2 checksum |
| [STATEMENT\_DIGEST()](#_bookmark240) | Compute statement digest hash value |
| [STATEMENT\_DIGEST\_TEXT()](#_bookmark241) | Compute normalized statement digest |
| [UNCOMPRESS()](#_bookmark242) | Uncompress a string compressed |
| [UNCOMPRESSED\_LENGTH()](#_bookmark243) | Return the length of a string before compression |
| [VALIDATE\_PASSWORD\_STRENGTH()](#_bookmark244) | Determine strength of password |

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a VARBINARY or BLOB binary string data type. This avoids potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type (CHAR, VARCHAR, TEXT).

Some encryption functions return strings of ASCII characters: [MD5()](#_bookmark236), [SHA()](#_bookmark238), [SHA1()](#_bookmark238), [SHA2()](#_bookmark239), [STATEMENT\_DIGEST()](#_bookmark240), [STATEMENT\_DIGEST\_TEXT()](#_bookmark241). Their return value is a string that has a character set and collation determined by the character\_set\_connection and collation\_connection system variables. This is a nonbinary string unless the character set is binary.

If an application stores values from a function such as [MD5()](#_bookmark236) or [SHA1()](#_bookmark238) that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using [UNHEX()](#_bookmark166) and storing the result in a BINARY(*N*) column. Each pair of hexadecimal digits requires one byte in binary form, so the value of *N*depends on the length of the hex string. *N* is 16 for an [MD5()](#_bookmark236) value and 20 for a [SHA1()](#_bookmark238) value. For [SHA2()](#_bookmark239), *N* ranges from 28 to 32 depending on the argument specifying the desired bit length of the result.

The size penalty for storing the hex string in a CHAR column is at least two times, up to eight times if the value is stored in a column that uses the utf8mb4 character set (where each character uses 4 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores [MD5()](#_bookmark236) string values in a CHAR(32) column:

CREATE TABLE md5\_tbl (md5\_val CHAR(32), ...);

INSERT INTO md5\_tbl (md5\_val, ...) VALUES(MD5('abcdef'), ...);

To convert hex strings to more compact form, modify the application to use [UNHEX()](#_bookmark166) and BINARY(16) instead as follows:

CREATE TABLE md5\_tbl (md5\_val BINARY(16), ...);

INSERT INTO md5\_tbl (md5\_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);

Applications should be prepared to handle the very rare case that a hashing function produces the

same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.



**Note**

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using another one-way encryption function described in this section instead, such as [SHA2()](#_bookmark239).

**Caution**

Passwords or other sensitive values supplied as arguments to encryption functions are sent as cleartext to the MySQL server unless an SSL connection is used. Also, such values appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

• [AES\_DECRYPT(*crypt\_str*,*key\_str*[,*init\_vector*][,*kdf\_name*][,*salt*][,*info* *|*](#_bookmark233) [*iterations*])](#_bookmark233)

This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of [AES\_ENCRYPT()](#_bookmark234).

Statements that use [AES\_DECRYPT()](#_bookmark233) are unsafe for statement-based replication.

• [AES\_ENCRYPT(*str*,*key\_str*[,*init\_vector*][,*kdf\_name*][,*salt*][,*info* *|*](#_bookmark234) [*iterations*])](#_bookmark234)

[AES\_ENCRYPT()](#_bookmark234) and [AES\_DECRYPT()](#_bookmark233) implement encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” The AES standard permits various key lengths. By default these functions implement AES with a 128-bit key length. Key lengths of 196 or 256 bits can be used, as described later. The key length is a trade off between performance and security.

[AES\_ENCRYPT()](#_bookmark234) encrypts the string *str* using the key string *key\_str*, and returns a binary string containing the encrypted output. [AES\_DECRYPT()](#_bookmark233) decrypts the encrypted string *crypt\_str* using the key string *key\_str*, and returns the original plaintext string. If either function argument is NULL, the function returns NULL. If [AES\_DECRYPT()](#_bookmark233) detects invalid data or incorrect padding, it returns NULL. However, it is possible for [AES\_DECRYPT()](#_bookmark233) to return a non-NULL value (possibly garbage) if the input data or the key is invalid.

From MySQL 8.0.30, the functions support the use of a key derivation function (KDF) to create a cryptographically strong secret key from the information passed in *key\_str*. The derived key is used to encrypt and decrypt the data, and it remains in the MySQL Server instance and is not accessible to users. Using a KDF is highly recommended, as it provides better security than specifying your own premade key or deriving it by a simpler method as you use the function. The functions support HKDF (available from OpenSSL 1.1.0), for which you can specify an optional salt and context-specific information to include in the keying material, and PBKDF2 (available from OpenSSL 1.0.2), for which you can specify an optional salt and set the number of iterations used to produce the key.

[AES\_ENCRYPT()](#_bookmark234) and [AES\_DECRYPT()](#_bookmark233) permit control of the block encryption mode. The block\_encryption\_mode system variable controls the mode for block-based encryption algorithms. Its default value is aes-128-ecb, which signifies encryption using a key length of 128 bits and ECB mode. For a description of the permitted values of this variable, see Section 5.1.8, “Server System Variables” . The optional *init\_vector* argument is used to provide an initialization vector for block encryption modes that require it.

Statements that use [AES\_ENCRYPT()](#_bookmark234) or [AES\_DECRYPT()](#_bookmark233) are unsafe for statement-based replication.

Encryption and Compression Functions

INSERT INTO t

VALUES (1,AES\_ENCRYPT('text',UNHEX('F3229A0B371ED2D9441B830D21A390C3')))

INSERT INTO t

VALUES (1,AES\_ENCRYPT('text', UNHEX(SHA2('My secret passphrase',512))));

If [AES\_ENCRYPT()](#_bookmark234) is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

The arguments for the [AES\_ENCRYPT()](#_bookmark234) and [AES\_DECRYPT()](#_bookmark233) functions are as follows:

*str*

*crypt\_str*

*key\_str*

The string for [AES\_ENCRYPT()](#_bookmark234) to encrypt using the key string *key\_str*, or (from MySQL 8.0.30) the key derived from it by the specified KDF. The string can be any length. Padding is automatically added to *str* so it is a multiple of a block as required by block-based algorithms such as AES. This padding is automatically removed by the [AES\_DECRYPT()](#_bookmark233) function.

The encrypted string for [AES\_DECRYPT()](#_bookmark233) to decrypt using the key string *key\_str*, or (from MySQL 8.0.30) the key derived from it by the specified KDF. The string can be any length. The length of *crypt\_str* can be calculated from the length of the original string using this formula:

16 \* (trunc(*string\_length* / 16) + 1)

The encryption key, or the input keying material that is used

as the basis for deriving a key using a key derivation function

(KDF). For the same instance of data, use the same value of

*key\_str* for encryption with [AES\_ENCRYPT()](#_bookmark234) and decryption

with [AES\_DECRYPT()](#_bookmark233).

If you are using a KDF, which you can from MySQL 8.0.30, *key\_str* can be any arbitrary information such as a password or passphrase. In the further arguments for the function, you specify the KDF name, then add further options to increase the security as appropriate for the KDF.

When you use a KDF, the function creates a cryptographically strong secret key from the information passed in *key\_str* and any salt or additional information that you provide in the other arguments. The derived key is used to encrypt and decrypt the data, and it remains in the MySQL Server instance and is not accessible to users. Using a KDF is highly recommended, as it provides better security than specifying your own premade key or deriving it by a simpler method as you use the function.

If you are not using a KDF, for a key length of 128 bits, the most secure way to pass a key to the *key\_str* argument is to create a truly random 128-bit value and pass it as a binary value. For example:

A passphrase can be used to generate an AES key by hashing the passphrase. For example:

If you exceed the maximum key length of 128 bits, a warning is returned. If you are not using a KDF, do not pass a password or passphrase directly to *key\_str*, hash it first. Previous versions of this documentation suggested the former approach, but it is



*init\_vector*

*kdf\_name*

no longer recommended as the examples shown here are more secure.

An initialization vector, for block encryption modes that require it. The block\_encryption\_mode system variable controls the mode. For the same instance of data, use the same value of *init\_vector* for encryption with [AES\_ENCRYPT()](#_bookmark234) and decryption with [AES\_DECRYPT()](#_bookmark233).

**Note**

If you are using a KDF, you must specify an initialization vector or a null string for this argument, in order to access the later arguments to define the KDF.

For modes that require an initialization vector, it must be 16 bytes or longer (bytes in excess of 16 are ignored). An error occurs if *init\_vector* is missing. For modes that do not require an initialization vector, it is ignored and a warning is generated if *init\_vector* is specified, unless you are using a KDF.

The default value for the block\_encryption\_mode system variable is aes-128-ecb, or ECB mode, which does not require an initialization vector. The alternative permitted block encryption modes CBC, CFB1, CFB8, CFB128, and OFB all require an initialization vector.

A random string of bytes to use for the initialization vector can be produced by calling [RANDOM\_BYTES(16)](#_bookmark237).

The name of the key derivation function (KDF) to create a key from the input keying material passed in *key\_str*, and other arguments as appropriate for the KDF. This optional argument is available from MySQL 8.0.30.

For the same instance of data, use the same value of *kdf\_name* for encryption with [AES\_ENCRYPT()](#_bookmark234) and decryption with [AES\_DECRYPT()](#_bookmark233). When you specify *kdf\_name*, you must specify *init\_vector*, using either a valid initialization vector, or a null string if the encryption mode does not require an initialization vector.

The following values are supported:

HKDF, which is available from OpenSSL 1.1.0. HKDF extracts a pseudorandom key from the keying material then expands it into additional keys. With HKDF, you can specify an optional salt (*salt*) and context-specific information such as application details (*info*) to include in the keying material.

hkdf

PBKDF2, which is available from OpenSSL 1.0.2. PBKDF2

pbkdf2\_hmac

Encryption and Compression Functions

*salt*

*info*

*iterations*

applies a pseudorandom function to the keying material, and repeats this process a large number of times to produce the key. With PBKDF2, you can specify an optional salt (*salt*) to include in the keying material, and set the number of iterations used to produce the key (*iterations*).

In this example, HKDF is specified as the key derivation function, and a salt and context information are provided. The argument for the initialization vector is included but is the empty string:

SELECT AES\_ENCRYPT('mytext','mykeystring', '', 'hkdf', 'salt', 'info');

In this example, PBKDF2 is specified as the key derivation

function, a salt is provided, and the number of iterations is

doubled from the recommended minimum:

SELECT AES\_ENCRYPT('mytext','mykeystring', '', 'pbkdf2\_hmac','salt', '20

A salt to be passed to the key derivation function (KDF). This optional argument is available from MySQL 8.0.30. Both HKDF and PBKDF2 can use salts, and their use is recommended to help prevent attacks based on dictionaries of common passwords or rainbow tables.

A salt consists of random data, which for security must be different for each encryption operation. A random string of bytes to use for the salt can be produced by calling [RANDOM\_BYTES()](#_bookmark237).

This example produces a 64-bit salt: SET @salt = RANDOM\_BYTES(8);

For the same instance of data, use the same value of *salt*

for encryption with [AES\_ENCRYPT()](#_bookmark234) and decryption with

[AES\_DECRYPT()](#_bookmark233). The salt can safely be stored along with the

encrypted data.

Context-specific information for HKDF to include in the keying material, such as information about the application. This optional argument is available from MySQL 8.0.30 when you specify hkdf as the KDF name. HKDF adds this information to the keying material specified in *key\_str* and the salt specified in *salt* to produce the key.

For the same instance of data, use the same value of *info*

for encryption with [AES\_ENCRYPT()](#_bookmark234) and decryption with

[AES\_DECRYPT()](#_bookmark233).

The iteration count for PBKDF2 to use when producing the key. This optional argument is available from MySQL 8.0.30 when you specify pbkdf2\_hmac as the KDF name. A higher count gives greater resistance to brute-force attacks because it has a greater computational cost for the attacker, but the same is necessarily true for the key derivation process. The default if

you do not specify this argument is 1000, which is the minimum recommended by the OpenSSL standard.

For the same instance of data, use the same value of *iterations* for encryption with [AES\_ENCRYPT()](#_bookmark234) and decryption with [AES\_DECRYPT()](#_bookmark233).

mysql> **SET** **block\_encryption\_mode** **=** **'aes-256-cbc';**

mysql> **SET** **@key\_str** **=** **SHA2('My** **secret** **passphrase',512);**

mysql> **SET** **@init\_vector** **=** **RANDOM\_BYTES(16);**

mysql> **SET** **@crypt\_str** **=** **AES\_ENCRYPT('text',@key\_str,@init\_vector);**

mysql> **SELECT** **AES\_DECRYPT(@crypt\_str,@key\_str,@init\_vector);**

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

| AES\_DECRYPT(@crypt\_str,@key\_str,@init\_vector) |

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

| text |

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

• [COMPRESS(*string\_to\_compress*)](#_bookmark235)

Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as zlib. Otherwise, the return value is always NULL. The return value is also NULL if *string\_to\_compress* is NULL. The compressed string can be uncompressed with [UNCOMPRESS()](#_bookmark242).

mysql> **SELECT**

**LENGTH(COMPRESS(REPEAT('a',1000)));** **LENGTH(COMPRESS(''));** **LENGTH(COMPRESS('a'));**

**LENGTH(COMPRESS(REPEAT('a',16)));**

-> 21

mysql> **SELECT**

-> 0

mysql> **SELECT**

-> 13

mysql> **SELECT**

-> 15

The compressed string contents are stored the following way:

• Empty strings are stored as empty strings.

• Nonempty strings are stored as a 4-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra . character is added to avoid problems with endspace trimming should the result be stored in a CHAR or VARCHAR column. (However, use of nonbinary string data types such as CHAR or VARCHAR to store compressed strings is not recommended anyway because character set conversion may occur. Use a VARBINARY or BLOB binary string column instead.)

If [COMPRESS()](#_bookmark235) is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [MD5(*str*)](#_bookmark236)

Calculates an MD5 128-bit checksum for the string. The value is returned as a string of 32 hexadecimal digits, or NULL if the argument was NULL. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

The return value is a string in the connection character set.

If FIPS mode is enabled, MD5() returns NULL. See Section 6.8, “FIPS Support” .

mysql> **SELECT** **MD5('testing');**

-> 'ae2b1fca515949e5d54fb22b8ed95575'

This is the “RSA Data Security, Inc. MD5 Message-Digest Algorithm.”

See the note regarding the MD5 algorithm at the beginning this section.

• [RANDOM\_BYTES(*len*)](#_bookmark237)

This function returns a binary string of *len* random bytes generated using the random number generator of the SSL library. Permitted values of *len* range from 1 to 1024. For values outside that range, an error occurs. Returns NULL if *len* is NULL.

[RANDOM\_BYTES()](#_bookmark237) can be used to provide the initialization vector for the [AES\_DECRYPT()](#_bookmark233) and [AES\_ENCRYPT()](#_bookmark234) functions. For use in that context, *len* must be at least 16. Larger values are permitted, but bytes in excess of 16 are ignored.

[RANDOM\_BYTES()](#_bookmark237) generates a random value, which makes its result nondeterministic. Consequently, statements that use this function are unsafe for statement-based replication.

If [RANDOM\_BYTES()](#_bookmark237) is invoked from within the mysql client, binary strings display using hexadecimal notation, depending on the value of the --binary-as-hex. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client” .

• [SHA1(*str*)](#_bookmark238), [SHA(*str*)](#_bookmark238)

Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hexadecimal digits, or NULL if the argument is NULL. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. [SHA()](#_bookmark238) is synonymous with [SHA1()](#_bookmark238).

The return value is a string in the connection character set.

mysql> **SELECT** **SHA1('abc');**

-> 'a9993e364706816aba3e25717850c26c9cd0d89d'

[SHA1()](#_bookmark238) can be considered a cryptographically more secure equivalent of [MD5()](#_bookmark236). However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

• [SHA2(*str*, *hash\_length*)](#_bookmark239)

Calculates the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). The first argument is the plaintext string to be hashed. The second argument indicates the desired bit length of the result, which must have a value of 224, 256, 384, 512, or 0 (which is equivalent to 256). If either argument is NULL or the hash length is not one of the permitted values, the return value is NULL. Otherwise, the function result is a hash value containing the desired number of bits. See the notes at the beginning of this section about storing hash values efficiently.

The return value is a string in the connection character set.

mysql> **SELECT** **SHA2('abc',** **224);**

-> '23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7'

This function works only if MySQL has been configured with SSL support. See Section 6.3, “Using Encrypted Connections” .

[SHA2()](#_bookmark239) can be considered cryptographically more secure than [MD5()](#_bookmark236) or [SHA1()](#_bookmark238).

• [STATEMENT\_DIGEST(*statement*)](#_bookmark240)

Given an SQL statement as a string, returns the statement digest hash value as a string in the connection character set, or NULL if the argument is NULL. The related

about statement digesting, see Section 27.10, “Performance Schema Statement Digests and Sampling” .

Both functions use the MySQL parser to parse the statement. If parsing fails, an error occurs. The error message includes the parse error only if the statement is provided as a literal string.

The max\_digest\_length system variable determines the maximum number of bytes available to these functions for computing normalized statement digests.

mysql> **SET** **@stmt** **=** **'SELECT** **\*** **FROM** **mytable** **WHERE** **cola** **=** **10** **AND** **colb** **=** **20';**

mysql> **SELECT** **STATEMENT\_DIGEST(@stmt);**

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

| STATEMENT\_DIGEST(@stmt) |

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

| 3bb95eeade896657c4526e74ff2a2862039d0a0fe8a9e7155b5fe492cbd78387 |

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

mysql> **SELECT** **STATEMENT\_DIGEST\_TEXT(@stmt);**

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

| STATEMENT\_DIGEST\_TEXT(@stmt) |

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

| SELECT \* FROM `mytable` WHERE `cola` = ? AND `colb` = ? |

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

• [STATEMENT\_DIGEST\_TEXT(*statement*)](#_bookmark241)

Given an SQL statement as a string, returns the normalized statement digest as a string in the connection character set, or NULL if the argument is NULL. For additional discussion and examples, see the description of the related [STATEMENT\_DIGEST()](#_bookmark240) function.

• [UNCOMPRESS(*string\_to\_uncompress*)](#_bookmark242)

Uncompresses a string compressed by the [COMPRESS()](#_bookmark235) function. If the argument is not a compressed value, the result is NULL; if *string\_to\_uncompress* is NULL, the result is also NULL. This function requires MySQL to have been compiled with a compression library such as zlib. Otherwise, the return value is always NULL.

mysql> **SELECT** **UNCOMPRESS(COMPRESS('any** **string'));**

-> 'any string'

mysql> **SELECT** **UNCOMPRESS('any** **string');**

-> NULL

• [UNCOMPRESSED\_LENGTH(*compressed\_string*)](#_bookmark243)

Returns the length that the compressed string had before being compressed. Returns NULL if *compressed\_string* is NULL.

mysql> **SELECT** **UNCOMPRESSED\_LENGTH(COMPRESS(REPEAT('a',30)));**

-> 30

• [VALIDATE\_PASSWORD\_STRENGTH(*str*)](#_bookmark244)

Given an argument representing a plaintext password, this function returns an integer to indicate how strong the password is, or NULL if the argument is NULL. The return value ranges from 0 (weak) to 100 (strong).

Password assessment by [VALIDATE\_PASSWORD\_STRENGTH()](#_bookmark244) is done by the validate\_password component. If that component is not installed, the function always returns 0. For information about installing validate\_password, see Section 6.4.3, “The Password Validation Component” . To examine or configure the parameters that affect password testing, check or set the system variables implemented by validate\_password. See Section 6.4.3.2, “Password Validation Options and Variables” .

The password is subjected to increasingly strict tests and the return value reflects

which tests were satisfied, as shown in the following table. In addition, if the validate\_password.check\_user\_name system variable is enabled and the password

matches the user name, [VALIDATE\_PASSWORD\_STRENGTH()](#_bookmark244) returns 0 regardless of how other validate\_password system variables are set.

|  |  |
| --- | --- |
| **Password** **Test** | **Return** **Value** |
| Length < 4 | 0 |
| Length ≥ 4 and <  validate\_password.length | 25 |
| Satisfies policy 1 (LOW) | 50 |
| Satisfies policy 2 (MEDIUM) | 75 |
| Satisfies policy 3 (STRONG) | 100 |

**12.15** **Locking** **Functions**

This section describes functions used to manipulate user-level locks.

**Table** **12.19** **Locking** **Functions**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [GET\_LOCK()](#_bookmark245) | Get a named lock |
| [IS\_FREE\_LOCK()](#_bookmark246) | Whether the named lock is free |
| [IS\_USED\_LOCK()](#_bookmark247) | Whether the named lock is in use; return  connection identifier if true |
| [RELEASE\_ALL\_LOCKS()](#_bookmark248) | Release all current named locks |
| [RELEASE\_LOCK()](#_bookmark249) | Release the named lock |

• [GET\_LOCK(*str*,*timeout*)](#_bookmark245)

Tries to obtain a lock with a name given by the string *str*, using a timeout of *timeout* seconds. A negative *timeout* value means infinite timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

Returns 1 if the lock was obtained successfully, 0 if the attempt timed out (for example, because another client has previously locked the name), or NULL if an error occurred (such as running out of memory or the thread was killed with mysqladmin kill).

A lock obtained with [GET\_LOCK()](#_bookmark245) is released explicitly by executing [RELEASE\_LOCK()](#_bookmark249) or implicitly when your session terminates (either normally or abnormally). Locks obtained with [GET\_LOCK()](#_bookmark245) are not released when transactions commit or roll back.

[GET\_LOCK()](#_bookmark245) is implemented using the metadata locking (MDL) subsystem. Multiple simultaneous locks can be acquired and [GET\_LOCK()](#_bookmark245) does not release any existing locks. For example, suppose that you execute these statements:

|  |  |
| --- | --- |
| SELECT  SELECT  SELECT  SELECT | GET\_LOCK('lock1',10); GET\_LOCK('lock2',10); RELEASE\_LOCK('lock2'); RELEASE\_LOCK('lock1'); |

The second [GET\_LOCK()](#_bookmark245) acquires a second lock and both [RELEASE\_LOCK()](#_bookmark249) calls return 1 (success).

It is even possible for a given session to acquire multiple locks for the same name. Other sessions cannot acquire a lock with that name until the acquiring session releases all its locks for the name.

Uniquely named locks acquired with [GET\_LOCK()](#_bookmark245) appear in the Performance Schema metadata\_locks table. The OBJECT\_TYPE column says USER LEVEL LOCK and the



OBJECT\_NAME column indicates the lock name. In the case that multiple locks are acquired for the *same* name, only the first lock for the name registers a row in the metadata\_locks table. Subsequent locks for the name increment a counter in the lock but do not acquire additional metadata locks. The metadata\_locks row for the lock is deleted when the last lock instance on the name is released.

The capability of acquiring multiple locks means there is the possibility of deadlock among clients. When this happens, the server chooses a caller and terminates its lock-acquisition request with an [ER\_USER\_LOCK\_DEADLOCK](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_user_lock_deadlock) error. This error does not cause transactions to roll back.

MySQL enforces a maximum length on lock names of 64 characters.

[GET\_LOCK()](#_bookmark245) can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked within one session, [GET\_LOCK()](#_bookmark245) blocks any request by another session for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *db\_name.str* or *app\_name.str*.

If multiple clients are waiting for a lock, the order in which they acquire it is undefined. Applications should not assume that clients acquire the lock in the same order that they issued the lock requests.

[GET\_LOCK()](#_bookmark245) is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

Since GET\_LOCK() establishes a lock only on a single mysqld, it is not suitable for use with NDB Cluster, which has no way of enforcing an SQL lock across multiple MySQL servers. See Section 23.2.7.10, “Limitations Relating to Multiple NDB Cluster Nodes” , for more information.

**Caution**

With the capability of acquiring multiple named locks, it is possible for a single statement to acquire a large number of locks. For example:

INSERT INTO ... SELECT GET\_LOCK(t1.col\_name) FROM t1;

These types of statements may have certain adverse effects. For example, if the statement fails part way through and rolls back, locks acquired up to the point of failure still exist. If the intent is for there to be a correspondence between rows inserted and locks acquired, that intent is not satisfied. Also, if it is important that locks are granted in a certain order, be aware that result set order may differ depending on which execution plan the optimizer chooses. For these reasons, it may be best to limit applications to a single lock-acquisition call per statement.

A different locking interface is available as either a plugin service or a set of loadable functions. This interface provides lock namespaces and distinct read and write locks, unlike the interface provided by [GET\_LOCK()](#_bookmark245) and related functions. For details, see Section 5.6.9.1, “The Locking Service” .

• [IS\_FREE\_LOCK(*str*)](#_bookmark246)

Checks whether the lock named *str* is free to use (that is, not locked). Returns 1 if the lock is free (no one is using the lock), 0 if the lock is in use, and NULL if an error occurs (such as an incorrect argument).

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

• [IS\_USED\_LOCK(*str*)](#_bookmark247)

Checks whether the lock named *str* is in use (that is, locked). If so, it returns the connection identifier of the client session that holds the lock. Otherwise, it returns NULL.

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

• [RELEASE\_ALL\_LOCKS()](#_bookmark248)

Releases all named locks held by the current session and returns the number of locks released (0 if there were none)

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

• [RELEASE\_LOCK(*str*)](#_bookmark249)

Releases the lock named by the string *str* that was obtained with [GET\_LOCK()](#_bookmark245). Returns 1 if the lock was released, 0 if the lock was not established by this thread (in which case the lock is not released), and NULL if the named lock did not exist. The lock does not exist if it was never obtained by a call to [GET\_LOCK()](#_bookmark245) or if it has previously been released.

The DO statement is convenient to use with [RELEASE\_LOCK()](#_bookmark249). See Section 13.2.3, “DO Statement” .

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

**12.16** **Information** **Functions**

**Table** **12.20** **Information** **Functions**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | |
| [BENCHMARK()](#_bookmark250) | Repeatedly execute an expression | |
| [CHARSET()](#_bookmark175) | Return the character set of the argument | |
| [COERCIBILITY()](#_bookmark251) | Return the collation coercibility value of the string argument | |
| [COLLATION()](#_bookmark176) | Return the collation of the string argument | |
| [CONNECTION\_ID()](#_bookmark252) | Return the connection ID (thread ID) for the connection | |
| [CURRENT\_ROLE()](#_bookmark253) | Return the current active roles | |
| [CURRENT\_USER(), CURRENT\_USER](#_bookmark254) | The authenticated user name and host name | |
| [DATABASE()](#_bookmark255) | Return the default (current) database name | |
| [FOUND\_ROWS()](#_bookmark256) | For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause | |
| [ICU\_VERSION()](#_bookmark257) | ICU library version | |
| [LAST\_INSERT\_ID()](#_bookmark258) | Value of the AUTOINCREMENT column for the last INSERT | |
| [ROLES\_GRAPHML()](#_bookmark259) | Return a GraphML document memory role subgraphs | representing |
| [ROW\_COUNT()](#_bookmark260) | The number of rows updated | |
| [SCHEMA()](#_bookmark261) | Synonym for DATABASE() | |
| [SESSION\_USER()](#_bookmark262) | Synonym for USER() | |
| [SYSTEM\_USER()](#_bookmark263) | Synonym for USER() | |



|  |  |
| --- | --- |
| **Name** | **Description** |
| [USER()](#_bookmark264) | The user name and host name provided by the client |
| [VERSION()](#_bookmark265) | Return a string that indicates the MySQL server version |

• [BENCHMARK(*count*,*expr*)](#_bookmark250)

The [BENCHMARK()](#_bookmark250) function executes the expression *expr* repeatedly *count* times. It may be used to time how quickly MySQL processes the expression. The result value is 0, or NULL for inappropriate arguments such as a NULL or negative repeat count.

The intended use is from within the mysql client, which reports query execution times:

mysql> **SELECT** **BENCHMARK(1000000,AES\_ENCRYPT('hello','goodbye'));**

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

| BENCHMARK(1000000,AES\_ENCRYPT('hello','goodbye')) |

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

| 0 |

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

1 row in set (4.74 sec)

The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute [BENCHMARK()](#_bookmark250) several times, and to interpret the result with regard to how heavily loaded the server machine is.

[BENCHMARK()](#_bookmark250) is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

• Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, [BENCHMARK(10, (SELECT \* FROM t))](#_bookmark250) fails if the table t has more than one column or more than one row.

• Executing a SELECT *expr* statement *N*times differs from executing SELECT BENCHMARK(*N*, *expr*) in terms of the amount of overhead involved. The two have very different execution profiles and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation *N*times each. The latter involves only runtime evaluation *N*times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of [BENCHMARK()](#_bookmark250) thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.

• [CHARSET(*str*)](#_bookmark175)

Returns the character set of the string argument, or NULL if the argument is NULL.

mysql> **SELECT** **CHARSET('abc');**

-> 'utf8mb3'

mysql> **SELECT** **CHARSET(CONVERT('abc'** **USING** **latin1));**

-> 'latin1'

mysql> **SELECT** **CHARSET(USER());**

-> 'utf8mb3'

• [COERCIBILITY(*str*)](#_bookmark251)

Returns the collation coercibility value of the string argument.

mysql> **SELECT** **COERCIBILITY('abc'** **COLLATE** **utf8mb4\_swedish\_ci);**

-> 0

mysql> **SELECT** **COERCIBILITY(USER());**

-> 3

mysql> **SELECT** **COERCIBILITY('abc');**



-> 4

mysql> **SELECT** **COERCIBILITY(1000);**

-> 5

The return values have the meanings shown in the following table. Lower values have higher precedence.

|  |  |  |
| --- | --- | --- |
| **Coercibility** | **Meaning** | **Example** |
| 0 | Explicit collation | Value with COLLATE clause |
| 1 | No collation | Concatenation of strings with different collations |
| 2 | Implicit collation | Column value, stored routine parameter or local variable |
| 3 | System constant | [USER()](#_bookmark264) return value |
| 4 | Coercible | Literal string |
| 5 | Numeric | Numeric or temporal value |
| 6 | Ignorable | NULL or an expression derived from NULL |

For more information, see Section 10.8.4, “Collation Coercibility in Expressions” . • [COLLATION(*str*)](#_bookmark176)

Returns the collation of the string argument.

mysql> **SELECT** **COLLATION('abc');**

-> 'utf8mb4\_0900\_ai\_ci'

mysql> **SELECT** **COLLATION(\_utf8mb4'abc');**

-> 'utf8mb4\_0900\_ai\_ci'

mysql> **SELECT** **COLLATION(\_latin1'abc');**

-> 'latin1\_swedish\_ci'

• [CONNECTION\_ID()](#_bookmark252)

Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

The value returned by [CONNECTION\_ID()](#_bookmark252) is the same type of value as displayed in the ID column of the Information Schema PROCESSLIST table, the Id column of SHOW PROCESSLIST output, and the PROCESSLIST\_ID column of the Performance Schema threads table.

mysql> **SELECT** **CONNECTION\_ID();**

-> 23786

**Warning**

Changing the session value of the pseudo\_thread\_id system variable changes the value returned by the [CONNECTION\_ID()](#_bookmark252) function.

• [CURRENT\_ROLE()](#_bookmark253)

Returns a utf8mb3 string containing the current active roles for the current session, separated by commas, or NONE if there are none. The value reflects the setting of the sql\_quote\_show\_create system variable.

Suppose that an account is granted roles as follows:

GRANT 'r1', 'r2' TO 'u1'@'localhost';

SET DEFAULT ROLE ALL TO 'u1'@'localhost';

In sessions for u1, the initial [CURRENT\_ROLE()](#_bookmark253) value names the default account roles. Using SET ROLE changes that:

mysql> **SELECT** **CURRENT\_ROLE();**

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

| CURRENT\_ROLE() |

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

| `r1`@`%`,`r2`@`%` |

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

mysql> **SET** **ROLE** **'r1';** **SELECT** **CURRENT\_ROLE();**

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

| CURRENT\_ROLE() |

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

| `r1`@`%` |

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

• [CURRENT\_USER](#_bookmark254), [CURRENT\_USER()](#_bookmark254)

Returns the user name and host name combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the utf8mb3 character set.

The value of [CURRENT USER()](#_bookmark254)\_ can differ from the value of [USER()](#_bookmark264).

mysql> **SELECT** **USER();**

-> 'davida@localhost'

mysql> **SELECT** **\*** **FROM** **mysql** **.user;**

ERROR 1044: Access denied for user ''@'localhost' to

database 'mysql'

mysql> **SELECT** **CURRENT\_USER();**

-> '@localhost'

The example illustrates that although the client specified a user name of davida (as indicated by the value of the [USER()](#_bookmark264) function), the server authenticated the client using an anonymous user account

(as seen by the empty user name part of the [CURRENT\_USER()](#_bookmark254) value). One way this might occur is that there is no account listed in the grant tables for davida.

Within a stored program or view, [CURRENT\_USER()](#_bookmark254) returns the account for the user who defined the object (as given by its DEFINER value) unless defined with the SQL SECURITY INVOKER characteristic. In the latter case, [CURRENT\_USER()](#_bookmark254) returns the object's invoker.

Triggers and events have no option to define the SQL SECURITY characteristic, so for these objects, [CURRENT\_USER()](#_bookmark254) returns the account for the user who defined the object. To return the invoker, use [USER()](#_bookmark264) or [SESSION USER()](#_bookmark262)\_.

The following statements support use of the [CURRENT\_USER()](#_bookmark254) function to take the place of the name of (and, possibly, a host for) an affected user or a definer; in such cases, [CURRENT\_USER()](#_bookmark254) is expanded where and as needed:

• DROP USER

• RENAME USER

• GRANT

• REVOKE

• CREATE FUNCTION

• CREATE PROCEDURE

• CREATE TRIGGER

• CREATE EVENT

• CREATE VIEW

• ALTER EVENT

• ALTER VIEW

• SET PASSWORD

For information about the implications that this expansion of [CURRENT\_USER()](#_bookmark254) has for replication, see Section 17.5.1.8, “Replication of CURRENT\_USER()” .

Beginning with MySQL 8.0.34, this function can be used for the default value of a VARCHAR or TEXT column, as shown in the following CREATE TABLE statement:

CREATE TABLE t (c VARCHAR(288) DEFAULT (CURRENT\_USER()));

• [DATABASE()](#_bookmark255)

Returns the default (current) database name as a string in the utf8mb3 character set. If there is no default database, [DATABASE()](#_bookmark255) returns NULL. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

mysql> **SELECT** **DATABASE();**

-> 'test'

If there is no default database, [DATABASE()](#_bookmark255) returns NULL.



• [FOUND\_ROWS()](#_bookmark256)

**Note**

The SQL\_CALC\_FOUND\_ROWS query modifier and accompanying [FOUND\_ROWS()](#_bookmark256) function are deprecated as of MySQL 8.0.17; expect them to be removed in a future version of MySQL. As a replacement, considering executing your query with LIMIT, and then a second query with COUNT(\*) and without LIMIT to determine whether there are additional rows. For example, instead of these queries:

SELECT SQL\_CALC\_FOUND\_ROWS \* FROM *tbl\_name* WHERE id > 100 LIMIT 10; SELECT FOUND\_ROWS();

Use these queries instead:

SELECT \* FROM *tbl\_name* WHERE id > 100 LIMIT 10;

SELECT COUNT(\*) FROM *tbl\_name* WHERE id > 100;

COUNT(\*) is subject to certain optimizations. SQL\_CALC\_FOUND\_ROWS causes some optimizations to be disabled.

A SELECT statement may include a LIMIT clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the LIMIT, but without running the statement again. To obtain this row count, include an SQL\_CALC\_FOUND\_ROWS option in the SELECT statement, and then invoke [FOUND\_ROWS()](#_bookmark256) afterward:

mysql> **SELECT** **SQL\_CALC\_FOUND\_ROWS** **\*** **FROM** ***tb1\_name***

-> **WHERE** **id** **>** **100** **LIMIT** **10;**

mysql> **SELECT** **FOUND\_ROWS();**

The second SELECT returns a number indicating how many rows the first SELECT would have returned had it been written without the LIMIT clause.

In the absence of the SQL\_CALC\_FOUND\_ROWS option in the most recent successful SELECT statement, [FOUND\_ROWS()](#_bookmark256) returns the number of rows in the result set returned by that statement. If the statement includes a LIMIT clause, [FOUND\_ROWS()](#_bookmark256) returns the number of rows up to the limit. For example, [FOUND\_ROWS()](#_bookmark256) returns 10 or 60, respectively, if the statement includes LIMIT 10 or

LIMIT 50, 10.

The row count available through [FOUND\_ROWS()](#_bookmark256) is transient and not intended to be available past the statement following the SELECT SQL\_CALC\_FOUND\_ROWS statement. If you need to refer to the value later, save it:

mysql> **SELECT** **SQL\_CALC\_FOUND\_ROWS** **\*** **FROM** **...** **;**

mysql> **SET** **@rows** **=** **FOUND\_ROWS();**

If you are using SELECT SQL\_CALC\_FOUND\_ROWS, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without LIMIT, because the result set need not be sent to the client.

SQL\_CALC\_FOUND\_ROWS and [FOUND\_ROWS()](#_bookmark256) can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display

containing links to the pages that show other sections of a search result. Using [FOUND\_ROWS()](#_bookmark256) enables you to determine how many other pages are needed for the rest of the result.

The use of SQL\_CALC\_FOUND\_ROWS and [FOUND\_ROWS()](#_bookmark256) is more complex for UNION statements than for simple SELECT statements, because LIMIT may occur at multiple places in a UNION. It may be applied to individual SELECT statements in the UNION, or global to the UNION result as a whole.

The intent of SQL\_CALC\_FOUND\_ROWS for UNION is that it should return the row count that would be returned without a global LIMIT. The conditions for use of SQL\_CALC\_FOUND\_ROWS with UNION are:

• The SQL\_CALC\_FOUND\_ROWS keyword must appear in the first SELECT of the UNION.

• The value of [FOUND\_ROWS()](#_bookmark256) is exact only if UNION ALL is used. If UNION without ALL is used, duplicate removal occurs and the value of [FOUND\_ROWS()](#_bookmark256) is only approximate.

• If no LIMIT is present in the UNION, SQL\_CALC\_FOUND\_ROWS is ignored and returns the number of rows in the temporary table that is created to process the UNION.

Beyond the cases described here, the behavior of [FOUND\_ROWS()](#_bookmark256) is undefined (for example, its value following a SELECT statement that fails with an error).

**Important**

[FOUND\_ROWS()](#_bookmark256) is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

• [ICU\_VERSION()](#_bookmark257)

The version of the International Components for Unicode (ICU) library used to support regular expression operations (see [Section 12.8.2, “Regular Expressions”](#_bookmark170)). This function is primarily intended for use in test cases.

• [LAST\_INSERT\_ID()](#_bookmark258), [LAST\_INSERT\_ID(*expr*)](#_bookmark258)

With no argument, [LAST\_INSERT\_ID()](#_bookmark258) returns a BIGINT UNSIGNED (64-bit) value representing the first automatically generated value successfully inserted for an AUTO\_INCREMENT column as a result of the most recently executed INSERT statement. The value of [LAST\_INSERT\_ID()](#_bookmark258) remains unchanged if no rows are successfully inserted.

With an argument, [LAST\_INSERT\_ID()](#_bookmark258) returns an unsigned integer, or NULL if the argument is

NULL.

For example, after inserting a row that generates an AUTO\_INCREMENT value, you can get the value like this:

mysql> **SELECT** **LAST\_INSERT\_ID();**

-> 195

The currently executing statement does not affect the value of [LAST\_INSERT\_ID()](#_bookmark258).

Suppose that you generate an AUTO\_INCREMENT value with one statement, and then refer to [LAST\_INSERT\_ID()](#_bookmark258) in a multiple-row INSERT statement that inserts rows into a table with its own AUTO\_INCREMENT column. The value of [LAST\_INSERT\_ID()](#_bookmark258) remains stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (You should be aware that, if you mix references to [LAST\_INSERT\_ID()](#_bookmark258) and [LAST\_INSERT\_ID(*expr*)](#_bookmark258), the effect is undefined.)

If the previous statement returned an error, the value of [LAST\_INSERT\_ID()](#_bookmark258) is undefined. For transactional tables, if the statement is rolled back due to an error, the value of [LAST\_INSERT\_ID()](#_bookmark258)

is left undefined. For manual ROLLBACK, the value of [LAST\_INSERT\_ID()](#_bookmark258) is not restored to that before the transaction; it remains as it was at the point of the ROLLBACK.

Within the body of a stored routine (procedure or function) or a trigger, the value of [LAST\_INSERT\_ID()](#_bookmark258) changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of [LAST\_INSERT\_ID()](#_bookmark258) that is seen by following statements depends on the kind of routine:

• If a stored procedure executes statements that change the value of [LAST\_INSERT\_ID()](#_bookmark258), the changed value is seen by statements that follow the procedure call.

• For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so statements coming after it do not see a changed value.

The ID that was generated is maintained in the server on a *per-connection* *basis*. This means that the value returned by the function to a given client is the first AUTO\_INCREMENT value generated for most recent statement affecting an AUTO\_INCREMENT column *by* *that* *client*. This value cannot be affected by other clients, even if they generate AUTO\_INCREMENT values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of [LAST\_INSERT\_ID()](#_bookmark258) is not changed if you set the AUTO\_INCREMENT column of a row to a non-“magic” value (that is, a value that is not NULL and not 0).

|  |  |  |
| --- | --- | --- |
| For example: |  | **Important**  If you insert multiple rows using a single INSERT statement, [LAST\_INSERT\_ID()](#_bookmark258) returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same INSERT statement against some other server. |

mysql> **USE** **test;**

mysql> **CREATE** **TABLE** **t** **(**

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

**name** **VARCHAR(10)** **NOT** **NULL**

**);**

mysql> **INSERT** **INTO** **t** **VALUES** **(NULL,** **'Bob');**

mysql> **SELECT** **\*** **FROM** **t;**

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

| id | name |

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

| 1 | Bob |

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

mysql> **SELECT** **LAST\_INSERT\_ID();**

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

| LAST\_INSERT\_ID() |

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

| 1 |

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

mysql> **INSERT** **INTO** **t** **VALUES**

**(NULL,** **'Mary'),** **(NULL,** **'Jane'),** **(NULL,** **'Lisa');**

mysql> **SELECT** **\*** **FROM** **t;**

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

| id | name |

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

| 1 | Bob |

| 2 | Mary |

| 3 | Jane |

| 4 | Lisa |

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

mysql> **SELECT** **LAST\_INSERT\_ID();**

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

| LAST\_INSERT\_ID() |

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

| 2 |

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

Although the second INSERT statement inserted three new rows into t, the ID generated for the first of these rows was 2, and it is this value that is returned by [LAST\_INSERT\_ID()](#_bookmark258) for the following SELECT statement.

If you use INSERT IGNORE and the row is ignored, the [LAST\_INSERT\_ID()](#_bookmark258) remains unchanged from the current value (or 0 is returned if the connection has not yet performed a successful INSERT) and, for non-transactional tables, the AUTO\_INCREMENT counter is not incremented. For InnoDB tables, the AUTO\_INCREMENT counter is incremented if innodb\_autoinc\_lock\_mode is set to 1 or 2, as demonstrated in the following example:

mysql> **USE** **test;**

mysql> **SELECT** **@@innodb\_autoinc\_lock\_mode;**

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

| @@innodb\_autoinc\_lock\_mode |

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

| 1 |

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

mysql> **CREATE** **TABLE** **`t`** **(**

**`id`** **INT(11)** **NOT** **NULL** **AUTO\_INCREMENT,**

**`val`** **INT(11)** **DEFAULT** **NULL,**

**PRIMARY** **KEY** **(`id`),**

**UNIQUE** **KEY** **`i1`** **(`val`)**

**)** **ENGINE=InnoDB** **DEFAULT** **CHARSET=latin1;**

# Insert two rows

mysql> **INSERT** **INTO** **t** **(val)** **VALUES** **(1),(2);**

# With auto\_increment\_offset=1, the inserted rows

# result in an AUTO\_INCREMENT value of 3

mysql> **SHOW** **CREATE** **TABLE** **t\G**

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

Table: t

Create Table: CREATE TABLE `t` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`val` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `i1` (`val`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=latin1

# LAST\_INSERT\_ID() returns the first automatically generated

# value that is successfully inserted for the AUTO\_INCREMENT column

mysql> **SELECT** **LAST\_INSERT\_ID();**

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

| LAST\_INSERT\_ID() |

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

| 1 |

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

# The attempted insertion of duplicate rows fail but errors are ignored

mysql> **INSERT** **IGNORE** **INTO** **t** **(val)** **VALUES** **(1),(2);**

Query OK, 0 rows affected (0.00 sec)

Records: 2 Duplicates: 2 Warnings: [0](#_bookmark266)

# With innodb\_autoinc\_lock\_mode=1, the AUTO\_INCREMENT counter

# is incremented for the ignored rows

mysql> **SHOW** **CREATE** **TABLE** **t\G**

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

Table: t

Create Table: CREATE TABLE `t` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`val` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `i1` (`val`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=latin1

# The LAST\_INSERT\_ID is unchanged because the previous insert was unsuccessful

mysql> **SELECT** **LAST\_INSERT\_ID();**

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

| LAST\_INSERT\_ID() |

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

| 1 |

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

For more information, see Section 15.6.1.6, “AUTO\_INCREMENT Handling in InnoDB” .

If *expr* is given as an argument to [LAST\_INSERT\_ID()](#_bookmark258), the value of the argument is returned by the function and is remembered as the next value to be returned by [LAST\_INSERT\_ID()](#_bookmark258). This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

mysql> **CREATE** **TABLE** **sequence** **(id** **INT** **NOT** **NULL);**

mysql> **INSERT** **INTO** **sequence** **VALUES** **(0);**

2. Use the table to generate sequence numbers like this:

mysql> **UPDATE** **sequence** **SET** **id=LAST\_INSERT\_ID(id+1);**

mysql> **SELECT** **LAST\_INSERT\_ID();**

The UPDATE statement increments the sequence counter and causes the next call to [LAST\_INSERT\_ID()](#_bookmark258) to return the updated value. The SELECT statement retrieves that value. The [mysql\_insert\_id()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-insert-id.html) C API function can also be used to get the value. See [mysql\_insert\_id()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-insert-id.html).

You can generate sequences without calling [LAST\_INSERT\_ID()](#_bookmark258), but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the UPDATE statement and get their own sequence value with the SELECT statement (or [mysql\_insert\_id()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-insert-id.html)), without affecting or being affected by other clients that generate their own sequence values.

Note that [mysql\_insert\_id()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-insert-id.html) is only updated after INSERT and UPDATE statements, so you cannot use the C API function to retrieve the value for [LAST\_INSERT\_ID(*expr*)](#_bookmark258) after executing other SQL statements like SELECT or SET.

• [ROLES\_GRAPHML()](#_bookmark259)

Returns a utf8mb3 string containing a GraphML document representing memory role subgraphs. The ROLE\_ADMIN privilege (or the deprecated SUPER privilege) is required to see content in the <graphml> element. Otherwise, the result shows only an empty element:

mysql> **SELECT** **ROLES\_GRAPHML();**

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

| ROLES\_GRAPHML() |

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

| <?xml version="1 .0" encoding="UTF-8"?><graphml /> |

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

• [ROW\_COUNT()](#_bookmark260)

ROW\_COUNT() returns a value as follows:

• DDL statements: 0. This applies to statements such as CREATE TABLE or DROP TABLE.

• DML statements other than SELECT: The number of affected rows. This applies to statements such as UPDATE, INSERT, or DELETE (as before), but now also to statements such as ALTER TABLE and LOAD DATA.

• SELECT: -1 if the statement returns a result set, or the number of rows “affected” if it does not. For example, for SELECT \* FROM t1, [ROW\_COUNT()](#_bookmark260) returns -1. For SELECT \* FROM t1 INTO OUTFILE '*file\_name*', [ROW\_COUNT()](#_bookmark260) returns the number of rows written to the file.

• SIGNAL statements: 0.

For UPDATE statements, the affected-rows value by default is the number of rows actually changed. If you specify the CLIENT\_FOUND\_ROWS flag to [mysql\_real\_connect()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-connect.html) when connecting to mysqld, the affected-rows value is the number of rows “found”; that is, matched by the WHERE clause.

For REPLACE statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For INSERT ... ON DUPLICATE KEY UPDATE statements, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the CLIENT\_FOUND\_ROWS flag, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

The [ROW\_COUNT()](#_bookmark260) value is similar to the value from the [mysql\_affected\_rows()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-affected-rows.html) C API function and the row count that the mysql client displays following statement execution.

mysql> **INSERT** **INTO** **t** **VALUES(1),(2),(3);**

Query OK, 3 rows affected (0.00 sec)

Records: 3 Duplicates: 0 Warnings: [0](#_bookmark267)

mysql> **SELECT** **ROW\_COUNT();**

+ +

| ROW\_COUNT() |

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

| 3 |

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

1 row in set (0.00 sec)

mysql> **DELETE** **FROM** **t** **WHERE** **i** **IN(1,2);**

Query OK, 2 rows affected (0.00 sec)

mysql> **SELECT** **ROW\_COUNT();**

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

| ROW\_COUNT() |

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

| 2 |

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

1 row in set (0.00 sec)

**Important**

[ROW\_COUNT()](#_bookmark260) is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

• [SCHEMA()](#_bookmark261)

This function is a synonym for [DATABASE()](#_bookmark255).



• [SESSION\_USER()](#_bookmark262)

[SESSION USER()](#_bookmark262)\_ is a synonym for [USER()](#_bookmark264).

Beginning with MySQL 8.0.34, like [USER()](#_bookmark264), this function can be used for the default value of a VARCHAR or TEXT column, as shown in the following CREATE TABLE statement:

CREATE TABLE t (c VARCHAR(288) DEFAULT (SESSION\_USER()));

• [SYSTEM\_USER()](#_bookmark263)

[SYSTEM USER()](#_bookmark263)\_ is a synonym for [USER()](#_bookmark264).

**Note**

The [SYSTEM\_USER()](#_bookmark263) function is distinct from the SYSTEM\_USER privilege. The former returns the current MySQL account name. The latter distinguishes the system user and regular user account categories (see Section 6.2.11, “Account Categories”).

Beginning with MySQL 8.0.34, like [USER()](#_bookmark264), this function can be used for the default value of a VARCHAR or TEXT column, as shown in the following CREATE TABLE statement:

CREATE TABLE t (c VARCHAR(288) DEFAULT (SYSTEM\_USER()));

• [USER()](#_bookmark264)

Returns the current MySQL user name and host name as a string in the utf8mb3 character set.

mysql> **SELECT** **USER();**

-> 'davida@localhost'

The value indicates the user name you specified when connecting to the server, and the client host from which you connected. The value can be different from that of [CURRENT\_USER()](#_bookmark254).

Beginning with MySQL 8.0.34, this function can be used for the default value of a VARCHAR or TEXT column, as shown in the following CREATE TABLE statement:

CREATE TABLE t (c VARCHAR(288) DEFAULT (USER()));

• [VERSION()](#_bookmark265)

Returns a string that indicates the MySQL server version. The string uses the utf8mb3 character set. The value might have a suffix in addition to the version number. See the description of the version system variable in Section 5.1.8, “Server System Variables” .

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog\_format is set to STATEMENT.

mysql> **SELECT** **VERSION();**

-> '8.0.32-standard'

**12.17** **Spatial** **Analysis** **Functions**

MySQL provides functions to perform various operations on spatial data. These functions can be grouped into several major categories according to the type of operation they perform:

• Functions that create geometries in various formats (WKT, WKB, internal)

• Functions that convert geometries between formats

• Functions that access qualitative or quantitative properties of a geometry

• Functions that describe relations between two geometries

• Functions that create new geometries from existing ones

For general background about MySQL support for using spatial data, see Section 11.4, “Spatial Data Types” .

**12.17.1** **Spatial** **Function** **Reference**

The following table lists each spatial function and provides a short description of each one.

**Table** **12.21** **Spatial** **Functions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | | **Introduced** |
| [GeomCollection()](#_bookmark268) | Construct geometry collection from geometries | |  |
| [GeometryCollection()](#_bookmark269) | Construct geometry collection from geometries | |  |
| [LineString()](#_bookmark270) | Construct LineString from Point values | |  |
| [MBRContains()](#_bookmark271) | Whether MBR of one geometry contains MBR of another | |  |
| [MBRCoveredBy()](#_bookmark272) | Whether one MBR is covered by another | |  |
| [MBRCovers()](#_bookmark273) | Whether one MBR covers  another | |  |
| [MBRDisjoint()](#_bookmark274) | Whether MBRs of two  geometries are disjoint | |  |
| [MBREquals()](#_bookmark275) | Whether MBRs of two  geometries are equal | |  |
| [MBRIntersects()](#_bookmark276) | Whether MBRs of two  geometries intersect | |  |
| [MBROverlaps()](#_bookmark277) | Whether MBRs of two  geometries overlap | |  |
| [MBRTouches()](#_bookmark278) | Whether MBRs of two  geometries touch | |  |
| [MBRWithin()](#_bookmark279) | Whether MBR of one geometry is within MBR of another | |  |
| [MultiLineString()](#_bookmark280) | Contruct MultiLineString from LineString values | |  |
| [MultiPoint()](#_bookmark281) | Construct MultiPoint from Point values | |  |
| [MultiPolygon()](#_bookmark282) | Construct MultiPolygon from Polygon values | |  |
| [Point()](#_bookmark283) | Construct Point from coordinates | |  |
| [Polygon()](#_bookmark284) | Construct Polygon from  LineString arguments | |  |
| [ST\_Area()](#_bookmark285) | Return Polygon or MultiPolygon area | |  |
| [ST\_AsBinary(), ST\_AsWKB()](#_bookmark286) | Convert from internal format to WKB | geometry |  |
| [ST\_AsGeoJSON()](#_bookmark287) | Generate GeoJSON object from geometry | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | | **Introduced** |
| [ST\_AsText(), ST\_AsWKT()](#_bookmark288) | Convert from internal format to WKT | geometry |  |
| [ST\_Buffer()](#_bookmark289) | Return geometry of points within given distance from geometry | |  |
| [ST\_Buffer\_Strategy()](#_bookmark290) | Produce strategy option for ST\_Buffer() | |  |
| [ST\_Centroid()](#_bookmark291) | Return centroid as a point | |  |
| [ST\_Collect()](#_bookmark292) | Aggregate spatial values into collection | | 8.0.24 |
| [ST\_Contains()](#_bookmark293) | Whether one geometry contains another | |  |
| [ST\_ConvexHull()](#_bookmark294) | Return convex hull of geometry | |  |
| [ST\_Crosses()](#_bookmark295) | Whether one geometry crosses another | |  |
| [ST\_Difference()](#_bookmark296) | Return point set difference of two geometries | |  |
| [ST\_Dimension()](#_bookmark297) | Dimension of geometry | |  |
| [ST\_Disjoint()](#_bookmark298) | Whether one geometry is disjoint from another | |  |
| [ST\_Distance()](#_bookmark299) | The distance of one geometry from another | |  |
| [ST\_Distance\_Sphere()](#_bookmark300) | Minimum distance on earth between two geometries | |  |
| [ST\_EndPoint()](#_bookmark301) | End Point of LineString | |  |
| [ST\_Envelope()](#_bookmark302) | Return MBR of geometry | |  |
| [ST\_Equals()](#_bookmark303) | Whether one geometry is equal to another | |  |
| [ST\_ExteriorRing()](#_bookmark304) | Return exterior ring of Polygon | |  |
| [ST\_FrechetDistance()](#_bookmark305) | The discrete Fréchet distance of one geometry from another | | 8.0.23 |
| [ST\_GeoHash()](#_bookmark306) | Produce a geohash value | |  |
| [ST\_GeomCollFromText(),](#_bookmark307) ST\_GeometryCollectionFrom [ST\_GeomCollFromTxt()](#_bookmark307) | Return geometry collection from T (), | |  |
| [ST\_GeomCollFromWKB(),](#_bookmark308) ST\_GeometryCollectionFrom | Return geometry collection from **W**KB() | |  |
| [ST\_GeometryN()](#_bookmark309) | Return N-th geometry from  geometry collection | |  |
| [ST\_GeometryType()](#_bookmark310) | Return name of geometry type | |  |
| [ST\_GeomFromGeoJSON()](#_bookmark311) | Generate geometry from  GeoJSON object | |  |
| [ST\_GeomFromText(),](#_bookmark312) [ST\_GeometryFromText()](#_bookmark312) | Return geometry from WKT | |  |
| [ST\_GeomFromWKB(),](#_bookmark313) [ST\_GeometryFromWKB()](#_bookmark313) | Return geometry from WKB | |  |

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Introduced** |
| [ST\_HausdorffDistance()](#_bookmark314) | The discrete Hausdorff distance of one geometry from another | 8.0.23 |
| [ST\_InteriorRingN()](#_bookmark315) | Return N-th interior ring of  Polygon |  |
| [ST\_Intersection()](#_bookmark316) | Return point set intersection of two geometries |  |
| [ST\_Intersects()](#_bookmark317) | Whether one geometry intersects another |  |
| [ST\_IsClosed()](#_bookmark318) | Whether a geometry is closed and simple |  |
| [ST\_IsEmpty()](#_bookmark319) | Whether a geometry is empty |  |
| [ST\_IsSimple()](#_bookmark320) | Whether a geometry is simple |  |
| [ST\_IsValid()](#_bookmark321) | Whether a geometry is valid |  |
| [ST\_LatFromGeoHash()](#_bookmark322) | Return latitude from geohash value |  |
| [ST\_Latitude()](#_bookmark323) | Return latitude of Point | 8.0.12 |
| [ST\_Length()](#_bookmark169) | Return length of LineString |  |
| [ST\_LineFromText(),](#_bookmark324) [ST\_LineStringFromText()](#_bookmark324) | Construct LineString from WKT |  |
| [ST\_LineFromWKB(),](#_bookmark325) [ST\_LineStringFromWKB()](#_bookmark325) | Construct LineString from WKB |  |
| [ST\_LineInterpolatePoint()](#_bookmark326) | The point a given percentage along a LineString | 8.0.24 |
| [ST\_LineInterpolatePoints(](#_bookmark327) | )The points a given percentage along a LineString | 8.0.24 |
| [ST\_LongFromGeoHash()](#_bookmark328) | Return longitude from geohash value |  |
| [ST\_Longitude()](#_bookmark329) | Return longitude of Point | 8.0.12 |
| [ST\_MakeEnvelope()](#_bookmark330) | Rectangle around two points |  |
| [ST\_MLineFromText(),](#_bookmark331) ST\_MultiLineStringFromTex | Construct MultiLineString from tW(K)T |  |
| [ST\_MLineFromWKB(),](#_bookmark332) ST\_MultiLineStringFromWKB | Construct MultiLineString from W()KB |  |
| [ST\_MPointFromText(),](#_bookmark333) [ST\_MultiPointFromText()](#_bookmark333) | Construct MultiPoint from WKT |  |
| [ST\_MPointFromWKB(),](#_bookmark334) [ST\_MultiPointFromWKB()](#_bookmark334) | Construct MultiPoint from WKB |  |
| [ST\_MPolyFromText(),](#_bookmark335) [ST\_MultiPolygonFromText()](#_bookmark335) | Construct MultiPolygon from  WKT |  |
| [ST\_MPolyFromWKB(),](#_bookmark336) [ST\_MultiPolygonFromWKB()](#_bookmark336) | Construct MultiPolygon from  WKB |  |
| [ST\_NumGeometries()](#_bookmark337) | Return number of geometries in geometry collection |  |
| [ST\_NumInteriorRing(),](#_bookmark338) [ST\_NumInteriorRings()](#_bookmark338) | Return number of interior rings in Polygon |  |

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Introduced** |
| [ST\_NumPoints()](#_bookmark339) | Return number of points in  LineString |  |
| [ST\_Overlaps()](#_bookmark340) | Whether one geometry overlaps another |  |
| [ST\_PointAtDistance()](#_bookmark341) | The point a given distance along a LineString | 8.0.24 |
| [ST\_PointFromGeoHash()](#_bookmark342) | Convert geohash value to POINT value |  |
| [ST\_PointFromText()](#_bookmark343) | Construct Point from WKT |  |
| [ST\_PointFromWKB()](#_bookmark344) | Construct Point from WKB |  |
| [ST\_PointN()](#_bookmark345) | Return N-th point from LineString |  |
| [ST\_PolyFromText(),](#_bookmark346) [ST\_PolygonFromText()](#_bookmark346) | Construct Polygon from WKT |  |
| [ST\_PolyFromWKB(),](#_bookmark347)  [ST\_PolygonFromWKB()](#_bookmark347) | Construct Polygon from WKB |  |
| [ST\_Simplify()](#_bookmark348) | Return simplified geometry |  |
| [ST\_SRID()](#_bookmark349) | Return spatial reference system ID for geometry |  |
| [ST\_StartPoint()](#_bookmark350) | Start Point of LineString |  |
| [ST\_SwapXY()](#_bookmark351) | Return argument with X/Y  coordinates swapped |  |
| [ST\_SymDifference()](#_bookmark352) | Return point set symmetric difference of two geometries |  |
| [ST\_Touches()](#_bookmark353) | Whether one geometry touches another |  |
| [ST\_Transform()](#_bookmark354) | Transform coordinates of  geometry | 8.0.13 |
| [ST\_Union()](#_bookmark355) | Return point set union of two geometries |  |
| [ST\_Validate()](#_bookmark356) | Return validated geometry |  |
| [ST\_Within()](#_bookmark357) | Whether one geometry is within another |  |
| [ST\_X()](#_bookmark358) | Return X coordinate of Point |  |
| [ST\_Y()](#_bookmark359) | Return Y coordinate of Point |  |

**12.17.2** **Argument** **Handling** **by** **Spatial** **Functions**

Spatial values, or geometries, have the properties described in Section 11.4.2.2, “Geometry Class” . The following discussion lists general spatial function argument-handling characteristics. Specific functions or groups of functions may have additional or different argument-handling characteristics, as discussed in the sections where those function descriptions occur. Where that is true, those descriptions take precedence over the general discussion here.

Spatial functions are defined only for valid geometry values. See Section 11.4.4, “Geometry Well- Formedness and Validity” .

Each geometry value is associated with a spatial reference system (SRS), which is a coordinate-based system for geographic locations. See Section 11.4.5, “Spatial Reference System Support” .

The spatial reference identifier (SRID) of a geometry identifies the SRS in which the geometry is defined. In MySQL, the SRID value is an integer associated with the geometry value. The maximum usable SRID value is 232 − 1. If a larger value is given, only the lower 32 bits are used.

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

For computations on multiple geometry values, all values must be in the same SRS or an error occurs. Thus, spatial functions that take multiple geometry arguments require those arguments to be in the same SRS. If a spatial function returns [ER\_GIS\_DIFFERENT\_SRIDS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_different_srids), it means that the geometry arguments were not all in the same SRS. You must modify them to have the same SRS.

A geometry returned by a spatial function is in the SRS of the geometry arguments because geometry values produced by any spatial function inherit the SRID of the geometry arguments.

The [Open Geospatial Consortium](http://www.opengeospatial.org) guidelines require that input polygons already be closed, so unclosed polygons are rejected as invalid rather than being closed.

In MySQL, the only valid empty geometry is represented in the form of an empty geometry collection. Empty geometry collection handling is as follows: An empty WKT input geometry collection may be specified as 'GEOMETRYCOLLECTION()'. This is also the output WKT resulting from a spatial operation that produces an empty geometry collection.

During parsing of a nested geometry collection, the collection is flattened and its basic components are used in various GIS operations to compute results. This provides additional flexibility to users because it is unnecessary to be concerned about the uniqueness of geometry data. Nested geometry collections may be produced from nested GIS function calls without having to be explicitly flattened first.

**12.17.3** **Functions** **That** **Create** **Geometry** **Values** **from** **WKT** **Values**

These functions take as arguments a Well-Known Text (WKT) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry. For a description of WKT format, see Well-Known Text (WKT) Format.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

[ST\_GeomFromText()](#_bookmark312) accepts a WKT value of any geometry type as its first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

Functions such as [ST\_MPointFromText()](#_bookmark333) and [ST\_GeomFromText()](#_bookmark312) that accept WKT-format representations of MultiPoint values permit individual points within values to be surrounded by parentheses. For example, both of the following function calls are valid:

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

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

Functions such as [ST\_GeomFromText()](#_bookmark312) that accept WKT geometry collection arguments understand both OpenGIS 'GEOMETRYCOLLECTION EMPTY' standard syntax and MySQL 'GEOMETRYCOLLECTION()' nonstandard syntax. Functions such as [ST\_AsWKT()](#_bookmark288) that produce WKT values produce 'GEOMETRYCOLLECTION EMPTY' standard syntax:

mysql> **SET** **@s1** **=** **ST\_GeomFromText('GEOMETRYCOLLECTION()');**

mysql> **SET** **@s2** **=** **ST\_GeomFromText('GEOMETRYCOLLECTION** **EMPTY');**

mysql> **SELECT** **ST\_AsWKT(@s1),** **ST\_AsWKT(@s2);**

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

| ST\_AsWKT(@s1) | ST\_AsWKT(@s2) |

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

| GEOMETRYCOLLECTION EMPTY | GEOMETRYCOLLECTION EMPTY |

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

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any geometry argument is NULL or is not a syntactically well-formed geometry, or if the SRID argument is NULL, the return value is NULL.

• By default, geographic coordinates (latitude, longitude) are interpreted as in the order specified by the spatial reference system of geometry arguments. An optional *options* argument may be given to override the default axis order. options consists of a list of comma-separated *key*=*value*. The only permitted *key* value is axis-order, with permitted values of lat-long, long-lat and srid-defined (the default).

If the *options* argument is NULL, the return value is NULL. If the *options* argument is invalid, an error occurs to indicate why.

• If an SRID argument refers to an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) error occurs.

• If a latitude value is not in the range [−90, 90], an [ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values

in its unit. The exact range limits deviate slightly due to floating-point arithmetic. These functions are available for creating geometries from WKT values:

• [ST\_GeomCollFromText(*wkt* [, *srid* [, *options*]])](#_bookmark307), [ST\_GeometryCollectionFromText(*wkt* [, *srid* [, *options*]])](#_bookmark307), [ST\_GeomCollFromTxt(*wkt* [, *srid* [, *options*]])](#_bookmark307)

Constructs a GeometryCollection value using its WKT representation and SRID. These functions handle their arguments as described in the introduction to this section.

mysql> **SET** **@g** **=** **"MULTILINESTRING((10** **10,** **11** **11),** **(9** **9,** **10** **10))";**

mysql> **SELECT** **ST\_AsText(ST\_GeomCollFromText(@g));**

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

| ST\_AsText(ST\_GeomCollFromText(@g)) |

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

| MULTILINESTRING((10 10,11 11),(9 9,10 10)) |

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

• [ST\_GeomFromText(*wkt* [, *srid* [, *options*]])](#_bookmark312), [ST\_GeometryFromText(*wkt* [, *srid*](#_bookmark312) [[, *options*]])](#_bookmark312)

Constructs a geometry value of any type using its WKT representation and SRID. These functions handle their arguments as described in the introduction to this section.

• [ST\_LineFromText(*wkt* [, *srid* [, *options*]])](#_bookmark324), [ST\_LineStringFromText(*wkt* [,](#_bookmark324) [*srid* [, *options*]])](#_bookmark324)

Constructs a LineString value using its WKT representation and SRID.

These functions handle their arguments as described in the introduction to this section.

• [ST\_MLineFromText(*wkt* [, *srid* [, *options*]])](#_bookmark331), [ST\_MultiLineStringFromText(*wkt*](#_bookmark331) [[, *srid* [, *options*]])](#_bookmark331)

Constructs a MultiLineString value using its WKT representation and SRID.

These functions handle their arguments as described in the introduction to this section.

• [ST\_MPointFromText(*wkt* [, *srid* [, *options*]])](#_bookmark333), [ST\_MultiPointFromText(*wkt* [,](#_bookmark333) [*srid* [, *options*]])](#_bookmark333)

Constructs a MultiPoint value using its WKT representation and SRID.

These functions handle their arguments as described in the introduction to this section.

• [ST\_MPolyFromText(*wkt* [, *srid* [, *options*]])](#_bookmark335), [ST\_MultiPolygonFromText(*wkt* [,](#_bookmark335) [*srid* [, *options*]])](#_bookmark335)

Constructs a MultiPolygon value using its WKT representation and SRID.

These functions handle their arguments as described in the introduction to this section.

• [ST\_PointFromText(*wkt* [, *srid* [, *options*]])](#_bookmark343) Constructs a Point value using its WKT representation and SRID.

[ST\_PointFromText()](#_bookmark343) handles its arguments as described in the introduction to this section.

• [ST\_PolyFromText(*wkt* [, *srid* [, *options*]])](#_bookmark346), [ST\_PolygonFromText(*wkt* [, *srid*](#_bookmark346) [[, *options*]])](#_bookmark346)

Constructs a Polygon value using its WKT representation and SRID.

These functions handle their arguments as described in the introduction to this section.

**12.17.4** **Functions** **That** **Create** **Geometry** **Values** **from** **WKB** **Values**

These functions take as arguments a BLOB containing a Well-Known Binary (WKB) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry. For a description of WKB format, see Well-Known Binary (WKB) Format.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

[ST\_GeomFromWKB()](#_bookmark313) accepts a WKB value of any geometry type as its first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

Prior to MySQL 8.0, these functions also accepted geometry objects as returned by the functions in [Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”](#_bookmark362) . Geometry arguments are no longer permitted and produce an error. To migrate calls from using geometry arguments to using WKB arguments, follow these guidelines:

• Rewrite constructs such as ST\_GeomFromWKB(Point(0, 0)) as Point(0, 0).

• Rewrite constructs such as ST\_GeomFromWKB(Point(0, 0), 4326) as ST\_SRID(Point(0, 0), 4326) or ST\_GeomFromWKB(ST\_AsWKB(Point(0, 0)), 4326).

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If the WKB or SRID argument is NULL, the return value is NULL.

• By default, geographic coordinates (latitude, longitude) are interpreted as in the order specified by the spatial reference system of geometry arguments. An optional *options* argument may be given to override the default axis order. options consists of a list of comma-separated *key*=*value*.

The only permitted *key* value is axis-order, with permitted values of lat-long, long-lat and srid-defined (the default).

If the *options* argument is NULL, the return value is NULL. If the *options* argument is invalid, an error occurs to indicate why.

• If an SRID argument refers to an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) error occurs.

• If a latitude value is not in the range [−90, 90], an [ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values

in its unit. The exact range limits deviate slightly due to floating-point arithmetic. These functions are available for creating geometries from WKB values:

• [ST\_GeomCollFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark308), [ST\_GeometryCollectionFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark308)

Constructs a GeometryCollection value using its WKB representation and SRID. These functions handle their arguments as described in the introduction to this section.

• [ST\_GeomFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark313), [ST\_GeometryFromWKB(*wkb* [, *srid* [,](#_bookmark313) [*options*]])](#_bookmark313)

Constructs a geometry value of any type using its WKB representation and SRID. These functions handle their arguments as described in the introduction to this section.

• [ST\_LineFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark325), [ST\_LineStringFromWKB(*wkb* [, *srid*](#_bookmark325) [[, *options*]])](#_bookmark325)

Constructs a LineString value using its WKB representation and SRID.

These functions handle their arguments as described in the introduction to this section.

• [ST\_MLineFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark332), [ST\_MultiLineStringFromWKB(*wkb* [,](#_bookmark332) [*srid* [, *options*]])](#_bookmark332)

Constructs a MultiLineString value using its WKB representation and SRID. These functions handle their arguments as described in the introduction to this section.

• [ST\_MPointFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark334), [ST\_MultiPointFromWKB(*wkb* [,](#_bookmark334) [*srid* [, *options*]])](#_bookmark334)

Constructs a MultiPoint value using its WKB representation and SRID.

These functions handle their arguments as described in the introduction to this section.

• [ST\_MPolyFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark336), [ST\_MultiPolygonFromWKB(*wkb* [,](#_bookmark336) [*srid* [, *options*]])](#_bookmark336)

Constructs a MultiPolygon value using its WKB representation and SRID.

These functions handle their arguments as described in the introduction to this section.

• [ST\_PointFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark344)

Constructs a Point value using its WKB representation and SRID.

[ST\_PointFromWKB()](#_bookmark344) handles its arguments as described in the introduction to this section.

• [ST\_PolyFromWKB(*wkb* [, *srid* [, *options*]])](#_bookmark347), [ST\_PolygonFromWKB(*wkb* [, *srid* [,](#_bookmark347) [*options*]])](#_bookmark347)

Constructs a Polygon value using its WKB representation and SRID.

These functions handle their arguments as described in the introduction to this section.

**12.17.5** **MySQL-Specific** **Functions** **That** **Create** **Geometry** **Values**

MySQL provides a set of useful nonstandard functions for creating geometry values. The functions described in this section are MySQL extensions to the OpenGIS specification.

These functions produce geometry objects from either WKB values or geometry objects as arguments. If any argument is not a proper WKB or geometry representation of the proper object type, the return value is NULL.

For example, you can insert the geometry return value from [Point()](#_bookmark283) directly into a POINT column:

INSERT INTO t1 (pt\_col) VALUES(Point(1,2));

• [GeomCollection(*g* [, *g*] ...)](#_bookmark268)

Constructs a GeomCollection value from the geometry arguments.

[GeomCollection()](#_bookmark268) returns all the proper geometries contained in the arguments even if a nonsupported geometry is present.

[GeomCollection()](#_bookmark268) with no arguments is permitted as a way to create an empty geometry. Also, functions such as [ST\_GeomFromText()](#_bookmark312) that accept WKT geometry collection arguments understand both OpenGIS 'GEOMETRYCOLLECTION EMPTY' standard syntax and MySQL 'GEOMETRYCOLLECTION()' nonstandard syntax.

[GeomCollection()](#_bookmark268) and [GeometryCollection()](#_bookmark269) are synonymous, with [GeomCollection()](#_bookmark268) the preferred function.

• [GeometryCollection(*g* [, *g*] ...)](#_bookmark269)

Constructs a GeomCollection value from the geometry arguments.

[GeometryCollection()](#_bookmark269) returns all the proper geometries contained in the arguments even if a nonsupported geometry is present.

[GeometryCollection()](#_bookmark269) with no arguments is permitted as a way to create an empty geometry. Also, functions such as [ST\_GeomFromText()](#_bookmark312) that accept WKT geometry collection arguments understand both OpenGIS 'GEOMETRYCOLLECTION EMPTY' standard syntax and MySQL 'GEOMETRYCOLLECTION()' nonstandard syntax.

[GeomCollection()](#_bookmark268) and [GeometryCollection()](#_bookmark269) are synonymous, with [GeomCollection()](#_bookmark268) the preferred function.

• [LineString(*pt* [, *pt*] ...)](#_bookmark270)

Constructs a LineString value from a number of Point or WKB Point arguments. If the number of arguments is less than two, the return value is NULL.

• [MultiLineString(*ls* [, *ls*] ...)](#_bookmark280)

Constructs a MultiLineString value using LineString or WKB LineString arguments. • [MultiPoint(*pt* [, *pt2*] ...)](#_bookmark281)

Constructs a MultiPoint value using Point or WKB Point arguments.

• [MultiPolygon(*poly* [, *poly*] ...)](#_bookmark282)

Constructs a MultiPolygon value from a set of Polygon or WKB Polygon arguments.

• [Point(*x*, *y*)](#_bookmark283)

Constructs a Point using its coordinates.

• [Polygon(*ls* [, *ls*] ...)](#_bookmark284)

Constructs a Polygon value from a number of LineString or WKB LineString arguments. If any argument does not represent a LinearRing (that is, not a closed and simple LineString), the return value is NULL.

**12.17.6** **Geometry** **Format** **Conversion** **Functions**

MySQL supports the functions listed in this section for converting geometry values from internal geometry format to WKT or WKB format, or for swapping the order of X and Y coordinates.

There are also functions to convert a string from WKT or WKB format to internal geometry format. See [Section 12.17.3, “Functions That Create Geometry Values from WKT Values”](#_bookmark360) , and [Section 12.17.4,](#_bookmark361) [“Functions That Create Geometry Values from WKB Values”](#_bookmark361) .

Functions such as [ST\_GeomFromText()](#_bookmark312) that accept WKT geometry collection arguments understand both OpenGIS 'GEOMETRYCOLLECTION EMPTY' standard syntax and MySQL 'GEOMETRYCOLLECTION()' nonstandard syntax. Another way to produce an empty geometry collection is by calling [GeometryCollection()](#_bookmark269) with no arguments. Functions such as [ST AsWKT()](#_bookmark288)\_ that produce WKT values produce 'GEOMETRYCOLLECTION EMPTY' standard syntax:

mysql> **SET** **@s1** **=** **ST\_GeomFromText('GEOMETRYCOLLECTION()');**

mysql> **SET** **@s2** **=** **ST\_GeomFromText('GEOMETRYCOLLECTION** **EMPTY');**

mysql> **SELECT** **ST\_AsWKT(@s1),** **ST\_AsWKT(@s2);**

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

| ST\_AsWKT(@s1) | ST\_AsWKT(@s2) |

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

| GEOMETRYCOLLECTION EMPTY | GEOMETRYCOLLECTION EMPTY |

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

mysql> **SELECT** **ST\_AsWKT(GeomCollection());**

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

| ST\_AsWKT(GeomCollection()) |

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

| GEOMETRYCOLLECTION EMPTY |

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

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is in an undefined spatial reference system, the axes are output in the order they appear in the geometry and an [ER\_WARN\_SRS\_NOT\_FOUND\_AXIS\_ORDER](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_warn_srs_not_found_axis_order) warning occurs.

• By default, geographic coordinates (latitude, longitude) are interpreted as in the order specified by the spatial reference system of geometry arguments. An optional *options* argument may be given to override the default axis order. options consists of a list of comma-separated *key*=*value*.

The only permitted *key* value is axis-order, with permitted values of lat-long, long-lat and srid-defined (the default).

If the *options* argument is NULL, the return value is NULL. If the *options* argument is invalid, an error occurs to indicate why.

• Otherwise, the return value is non-NULL.

These functions are available for format conversions or coordinate swapping:

• [ST\_AsBinary(*g* [, *options*])](#_bookmark286), [ST\_AsWKB(*g* [, *options*])](#_bookmark286)

Converts a value in internal geometry format to its WKB representation and returns the binary result.

The function return value has geographic coordinates (latitude, longitude) in the order specified by the spatial reference system that applies to the geometry argument. An optional *options* argument may be given to override the default axis order.

[ST\_AsBinary()](#_bookmark286) and [ST\_AsWKB()](#_bookmark286) handle their arguments as described in the introduction to this section.

mysql> **SET** **@g** **=** **ST\_LineFromText('LINESTRING(0** **5,5** **10,10** **15)',** **4326);**

mysql> **SELECT** **ST\_AsText(ST\_GeomFromWKB(ST\_AsWKB(@g)));**

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

| ST\_AsText(ST\_GeomFromWKB(ST\_AsWKB(@g))) |

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

| LINESTRING(5 0,10 5,15 10) |

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

mysql> **SELECT** **ST\_AsText(ST\_GeomFromWKB(ST\_AsWKB(@g,** **'axis-order=long-lat')));**

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

| ST\_AsText(ST\_GeomFromWKB(ST\_AsWKB(@g, 'axis-order=long-lat'))) |

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

| LINESTRING(0 5,5 10,10 15) |

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

mysql> **SELECT** **ST\_AsText(ST\_GeomFromWKB(ST\_AsWKB(@g,** **'axis-order=lat-long')));**

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

| ST\_AsText(ST\_GeomFromWKB(ST\_AsWKB(@g, 'axis-order=lat-long'))) |

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

| LINESTRING(5 0,10 5,15 10) |

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

• [ST\_AsText(*g* [, *options*])](#_bookmark288), [ST\_AsWKT(*g* [, *options*])](#_bookmark288)

Converts a value in internal geometry format to its WKT representation and returns the string result.

The function return value has geographic coordinates (latitude, longitude) in the order specified by the spatial reference system that applies to the geometry argument. An optional *options* argument may be given to override the default axis order.

[ST\_AsText()](#_bookmark288) and [ST\_AsWKT()](#_bookmark288) handle their arguments as described in the introduction to this section.

mysql> **SET** **@g** **=** **'LineString(1** **1,2** **2,3** **3)';**

mysql> **SELECT** **ST\_AsText(ST\_GeomFromText(@g));**

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

| ST\_AsText(ST\_GeomFromText(@g)) |

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

| LINESTRING(1 1,2 2,3 3) |

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

Output for MultiPoint values includes parentheses around each point. For example:

mysql> **SELECT** **ST\_AsText(ST\_GeomFromText(@mp));**

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

| ST\_AsText(ST\_GeomFromText(@mp)) |

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

| MULTIPOINT((1 1),(2 2),(3 3)) |

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

• [ST\_SwapXY(*g*)](#_bookmark351)

Accepts an argument in internal geometry format, swaps the X and Y values of each coordinate pair within the geometry, and returns the result.

[ST\_SwapXY()](#_bookmark351) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g** **=** **ST\_LineFromText('LINESTRING(0** **5,5** **10,10** **15)');**

mysql> **SELECT** **ST\_AsText(@g);**

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

| ST\_AsText(@g) |

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

| LINESTRING(0 5,5 10,10 15) |

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

mysql> **SELECT** **ST\_AsText(ST\_SwapXY(@g));**

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

| ST\_AsText(ST\_SwapXY(@g)) |

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

| LINESTRING(5 0,10 5,15 10) |

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

**12.17.7** **Geometry** **Property** **Functions**

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return NULL if the argument is of an incorrect geometry type. For example, the [ST\_Area()](#_bookmark285) polygon function returns NULL if the object type is neither Polygon nor MultiPolygon.

**12.17.7.1** **General** **Geometry** **Property** **Functions**

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• If any SRID argument is not within the range of a 32-bit unsigned integer, an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

• If any SRID argument refers to an undefined SRS, an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• Otherwise, the return value is non-NULL.

These functions are available for obtaining geometry properties:

• [ST\_Dimension(*g*)](#_bookmark297)

Returns the inherent dimension of the geometry value *g*. The dimension can be − 1, 0, 1, or 2. The meaning of these values is given in Section 11.4.2.2, “Geometry Class” .

[ST\_Dimension()](#_bookmark297) handles its arguments as described in the introduction to this section.

mysql> **SELECT** **ST\_Dimension(ST\_GeomFromText('LineString(1** **1,2** **2)'));**

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

| ST\_Dimension(ST\_GeomFromText('LineString(1 1,2 2)')) |

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

| 1 |

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

• [ST\_Envelope(*g*)](#_bookmark302)

Returns the minimum bounding rectangle (MBR) for the geometry value *g*. The result is returned as a Polygon value that is defined by the corner points of the bounding box:

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

mysql> **SELECT** **ST\_AsText(ST\_Envelope(ST\_GeomFromText('LineString(1** **1,2** **2)')));**

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

| ST\_AsText(ST\_Envelope(ST\_GeomFromText('LineString(1 1,2 2)'))) |

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

| POLYGON((1 1,2 1,2 2,1 2,1 1)) |

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

If the argument is a point or a vertical or horizontal line segment, [ST\_Envelope()](#_bookmark302) returns the point or the line segment as its MBR rather than returning an invalid polygon:

mysql> **SELECT** **ST\_AsText(ST\_Envelope(ST\_GeomFromText('LineString(1** **1,1** **2)')));**

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

| ST\_AsText(ST\_Envelope(ST\_GeomFromText('LineString(1 1,1 2)'))) |

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

| LINESTRING(1 1,1 2) |

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

[ST\_Envelope()](#_bookmark302) handles its arguments as described in the introduction to this section, with this exception:

• If the geometry has an SRID value for a geographic spatial reference system (SRS), an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

• [ST\_GeometryType(*g*)](#_bookmark310)

Returns a binary string indicating the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable Geometry subclasses.

[ST\_GeometryType()](#_bookmark310) handles its arguments as described in the introduction to this section.

mysql> **SELECT** **ST\_GeometryType(ST\_GeomFromText('POINT(1** **1)'));**

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

| ST\_GeometryType(ST\_GeomFromText('POINT(1 1)')) |

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

| POINT |

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

• [ST\_IsEmpty(*g*)](#_bookmark319)

This function is a placeholder that returns 1 for an empty geometry collection value or 0 otherwise.

The only valid empty geometry is represented in the form of an empty geometry collection value.

MySQL does not support GIS EMPTY values such as POINT EMPTY. [ST\_IsEmpty()](#_bookmark319) handles its arguments as described in the introduction to this section.

• [ST\_IsSimple(*g*)](#_bookmark320)

Returns 1 if the geometry value *g* is simple according to the ISO *SQL/MM* *Part* *3:* *Spatial* standard. [ST\_IsSimple()](#_bookmark320) returns 0 if the argument is not simple.

The descriptions of the instantiable geometric classes given under Section 11.4.2, “The OpenGIS Geometry Model” include the specific conditions that cause class instances to be classified as not simple.

[ST\_IsSimple()](#_bookmark320) handles its arguments as described in the introduction to this section, with this exception:

• If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

• [ST\_SRID(*g* [, *srid*])](#_bookmark349)

With a single argument representing a valid geometry object *g*, [ST\_SRID()](#_bookmark349) returns an integer indicating the ID of the spatial reference system (SRS) associated with *g*.

With the optional second argument representing a valid SRID value, [ST\_SRID()](#_bookmark349) returns an object with the same type as its first argument with an SRID value equal to the second argument. This only sets the SRID value of the object; it does not perform any transformation of coordinate values.

[ST\_SRID()](#_bookmark349) handles its arguments as described in the introduction to this section, with this exception:

• For the single-argument syntax, [ST\_SRID()](#_bookmark349) returns the geometry SRID even if it refers to an undefined SRS. An [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error does not occur.

[ST\_SRID(*g*, *target\_srid*)](#_bookmark349) and [ST\_Transform(*g*, *target\_srid*)](#_bookmark354) differ as follows:

• [ST\_SRID()](#_bookmark349) changes the geometry SRID value without transforming its coordinates.

• [ST\_Transform()](#_bookmark354) transforms the geometry coordinates in addition to changing its SRID value.

mysql> **SET** **@g** **=** **ST\_GeomFromText('LineString(1** **1,2** **2)',** **0);**

mysql> **SELECT** **ST\_SRID(@g);**

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

| ST\_SRID(@g) |

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

| 0 |

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

mysql> **SET** **@g** **=** **ST\_SRID(@g,** **4326);**

mysql> **SELECT** **ST\_SRID(@g);**

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

| ST\_SRID(@g) |

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

| 4326 |

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

It is possible to create a geometry in a particular SRID by passing to [ST\_SRID()](#_bookmark349) the result of one of the MySQL-specific functions for creating spatial values, along with an SRID value. For example:

SET @g1 = ST\_SRID(Point(1, 1), 4326);

However, that method creates the geometry in SRID 0, then casts it to SRID 4326 (WGS 84). A preferable alternative is to create the geometry with the correct spatial reference system to begin with. For example:

SET @g1 = ST\_PointFromText('POINT(1 1)', 4326);

SET @g1 = ST\_GeomFromText('POINT(1 1)', 4326);

The two-argument form of [ST\_SRID()](#_bookmark349) is useful for tasks such as correcting or changing the SRS of geometries that have an incorrect SRID.

**12.17.7.2** **Point** **Property** **Functions**

A Point consists of X and Y coordinates, which may be obtained using the [ST\_X()](#_bookmark358) and [ST\_Y()](#_bookmark359) functions, respectively. These functions also permit an optional second argument that specifies an X or Y coordinate value, in which case the function result is the Point object from the first argument with the appropriate coordinate modified to be equal to the second argument.

For Point objects that have a geographic spatial reference system (SRS), the longitude and latitude may be obtained using the [ST\_Longitude()](#_bookmark329) and [ST\_Latitude()](#_bookmark323) functions, respectively. These functions also permit an optional second argument that specifies a longitude or latitude value, in which case the function result is the Point object from the first argument with the longitude or latitude modified to be equal to the second argument.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL, the return value is NULL.

• If any geometry argument is a valid geometry but not a Point object, an [ER\_UNEXPECTED\_GEOMETRY\_TYPE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unexpected_geometry_type) error occurs.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• If an X or Y coordinate argument is provided and the value is -inf, +inf, or NaN, an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

• If a longitude or latitude value is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) error occurs.

• If a latitude value is not in the range [−90, 90], an [ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) error occurs. Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

• Otherwise, the return value is non-NULL.

These functions are available for obtaining point properties:

• [ST\_Latitude(*p* [, *new\_latitude\_val*])](#_bookmark323)

With a single argument representing a valid Point object *p* that has a geographic spatial reference system (SRS), [ST\_Latitude()](#_bookmark323) returns the latitude value of *p* as a double-precision number.

With the optional second argument representing a valid latitude value, [ST\_Latitude()](#_bookmark323) returns a Point object like the first argument with its latitude equal to the second argument.

[ST\_Latitude()](#_bookmark323) handles its arguments as described in the introduction to this section, with the addition that if the Point object is valid but does not have a geographic SRS, an [ER\_SRS\_NOT\_GEOGRAPHIC](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_geographic) error occurs.

mysql> **SET** **@pt** **=** **ST\_GeomFromText('POINT(45** **90)',** **4326);**

mysql> **SELECT** **ST\_Latitude(@pt);**

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

| ST\_Latitude(@pt) |

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

| 45 |

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

mysql> **SELECT** **ST\_AsText(ST\_Latitude(@pt,** **10));**

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

| ST\_AsText(ST\_Latitude(@pt, 10)) |

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

| POINT(10 90) |

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

This function was added in MySQL 8.0.12.

• [ST\_Longitude(*p* [, *new\_longitude\_val*])](#_bookmark329)

With a single argument representing a valid Point object *p* that has a geographic spatial reference system (SRS), [ST\_Longitude()](#_bookmark329) returns the longitude value of *p* as a double-precision number.

With the optional second argument representing a valid longitude value, [ST\_Longitude()](#_bookmark329) returns a Point object like the first argument with its longitude equal to the second argument.

[ST\_Longitude()](#_bookmark329) handles its arguments as described in the introduction to this section, with the addition that if the Point object is valid but does not have a geographic SRS, an [ER\_SRS\_NOT\_GEOGRAPHIC](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_geographic) error occurs.

mysql> **SET** **@pt** **=** **ST\_GeomFromText('POINT(45** **90)',** **4326);**

mysql> **SELECT** **ST\_Longitude(@pt);**

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

| ST\_Longitude(@pt) |

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

| 90 |

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

mysql> **SELECT** **ST\_AsText(ST\_Longitude(@pt,** **10));**

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

| ST\_AsText(ST\_Longitude(@pt, 10)) |

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

| POINT(45 10) |

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

This function was added in MySQL 8.0.12.

• [ST\_X(*p* [, *new\_x\_val*])](#_bookmark358)

With a single argument representing a valid Point object *p*, [ST\_X()](#_bookmark358) returns the X-coordinate value of *p* as a double-precision number. As of MySQL 8.0.12, the X coordinate is considered to refer to the axis that appears first in the Point spatial reference system (SRS) definition.

With the optional second argument, [ST\_X()](#_bookmark358) returns a Point object like the first argument with its X coordinate equal to the second argument. As of MySQL 8.0.12, if the Point object has a geographic SRS, the second argument must be in the proper range for longitude or latitude values.

[ST\_X()](#_bookmark358) handles its arguments as described in the introduction to this section.

mysql> **SELECT** **ST\_X(Point(56** **.7,** **53** **.34));**

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

| ST\_X(Point(56 .7, 53 .34)) |

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

| 56.7 |

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

mysql> **SELECT** **ST\_AsText(ST\_X(Point(56** **.7,** **53** **.34),** **10** **.5));**

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

| ST\_AsText(ST\_X(Point(56 .7, 53 .34), 10 .5)) |

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

| POINT(10 .5 53 .34) |

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

• [ST\_Y(*p* [, *new\_y\_val*])](#_bookmark359)

With a single argument representing a valid Point object *p*, [ST\_Y()](#_bookmark359) returns the Y-coordinate value of *p* as a double-precision number. As of MySQL 8.0.12, the Y coordinate is considered to refer to the axis that appears second in the Point spatial reference system (SRS) definition.

With the optional second argument, [ST\_Y()](#_bookmark359) returns a Point object like the first argument with its Y coordinate equal to the second argument. As of MySQL 8.0.12, if the Point object has a geographic SRS, the second argument must be in the proper range for longitude or latitude values.

[ST\_Y()](#_bookmark359) handles its arguments as described in the introduction to this section.

mysql> **SELECT** **ST\_Y(Point(56** **.7,** **53** **.34));**

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

| ST\_Y(Point(56 .7, 53 .34)) |

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

| 53.34 |

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

mysql> **SELECT** **ST\_AsText(ST\_Y(Point(56** **.7,** **53** **.34),** **10** **.5));**

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

| ST\_AsText(ST\_Y(Point(56 .7, 53 .34), 10 .5)) |

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

| POINT(56 .7 10 .5) |

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

**12.17.7.3** **LineString** **and** **MultiLineString** **Property** **Functions**

A LineString consists of Point values. You can extract particular points of a LineString, count the number of points that it contains, or obtain its length.

Some functions in this section also work for MultiLineString values.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL or any geometry argument is an empty geometry, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• Otherwise, the return value is non-NULL.

These functions are available for obtaining linestring properties:

• [ST\_EndPoint(*ls*)](#_bookmark301)

Returns the Point that is the endpoint of the LineString value *ls*.

[ST\_EndPoint()](#_bookmark301) handles its arguments as described in the introduction to this section.

mysql> **SET** **@ls** **=** **'LineString(1** **1,2** **2,3** **3)';**

mysql> **SELECT** **ST\_AsText(ST\_EndPoint(ST\_GeomFromText(@ls)));**

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

| ST\_AsText(ST\_EndPoint(ST\_GeomFromText(@ls))) |

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

| POINT(3 3) |

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

• [ST\_IsClosed(*ls*)](#_bookmark318)

For a LineString value *ls*, [ST\_IsClosed()](#_bookmark318) returns 1 if *ls* is closed (that is, its [ST\_StartPoint()](#_bookmark350) and [ST\_EndPoint()](#_bookmark301) values are the same).

For a MultiLineString value *ls*, [ST\_IsClosed()](#_bookmark318) returns 1 if *ls* is closed (that is, the

[ST\_StartPoint()](#_bookmark350) and [ST\_EndPoint()](#_bookmark301) values are the same for each LineString in *ls*). [ST\_IsClosed()](#_bookmark318) returns 0 if *ls* is not closed, and NULL if *ls* is NULL.

[ST\_IsClosed()](#_bookmark318) handles its arguments as described in the introduction to this section, with this exception:

• If the geometry has an SRID value for a geographic spatial reference system (SRS), an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

mysql> **SET** **@ls1** **=** **'LineString(1** **1,2** **2,3** **3,2** **2)';**

mysql> **SET** **@ls2** **=** **'LineString(1** **1,2** **2,3** **3,1** **1)';**

mysql> **SELECT** **ST\_IsClosed(ST\_GeomFromText(@ls1));**

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

| ST\_IsClosed(ST\_GeomFromText(@ls1)) |

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

| 0 |

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

mysql> **SELECT** **ST\_IsClosed(ST\_GeomFromText(@ls2));**

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

| ST\_IsClosed(ST\_GeomFromText(@ls2)) |

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

| 1 |

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

mysql> **SET** **@ls3** **=** **'MultiLineString((1** **1,2** **2,3** **3),(4** **4,5** **5))';**

mysql> **SELECT** **ST\_IsClosed(ST\_GeomFromText(@ls3));**

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

| ST\_IsClosed(ST\_GeomFromText(@ls3)) |

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

| 0 |

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

• [ST\_Length(*ls* [, *unit*])](#_bookmark169)

Returns a double-precision number indicating the length of the LineString or MultiLineString value *ls* in its associated spatial reference system. The length of a MultiLineString value is equal to the sum of the lengths of its elements.

[ST\_Length()](#_bookmark169) computes a result as follows:

• If the geometry is a valid LineString in a Cartesian SRS, the return value is the Cartesian length of the geometry.

• If the geometry is a valid MultiLineString in a Cartesian SRS, the return value is the sum of the Cartesian lengths of its elements.

• If the geometry is a valid LineString in a geographic SRS, the return value is the geodetic length of the geometry in that SRS, in meters.

• If the geometry is a valid MultiLineString in a geographic SRS, the return value is the sum of the geodetic lengths of its elements in that SRS, in meters.

[ST\_Length()](#_bookmark169) handles its arguments as described in the introduction to this section, with these exceptions:

• If the geometry is not a LineString or MultiLineString, the return value is NULL.

• If the geometry is geometrically invalid, either the result is an undefined length (that is, it can be any number), or an error occurs.

• If the length computation result is +inf, an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

• If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

As of MySQL 8.0.16, [ST\_Length()](#_bookmark169) permits an optional *unit* argument that specifies the linear unit for the returned length value. These rules apply:

• If a unit is specified but not supported by MySQL, an [ER\_UNIT\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unit_not_found) error occurs.

• If a supported linear unit is specified and the SRID is 0, an [ER\_GEOMETRY\_IN\_UNKNOWN\_LENGTH\_UNIT](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_in_unknown_length_unit) error occurs.

• If a supported linear unit is specified and the SRID is not 0, the result is in that unit.

• If a unit is not specified, the result is in the unit of the SRS of the geometries, whether Cartesian or geographic. Currently, all MySQL SRSs are expressed in meters.

A unit is supported if it is found in the INFORMATION\_SCHEMA ST\_UNITS\_OF\_MEASURE table. See Section 26.3.37, “The INFORMATION\_SCHEMA ST\_UNITS\_OF\_MEASURE Table” .

mysql> **SET** **@ls** **=** **ST\_GeomFromText('LineString(1** **1,2** **2,3** **3)');**

mysql> **SELECT** **ST\_Length(@ls);**

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

| ST\_Length(@ls) |

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

| 2.8284271247461903 |

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

mysql> **SET** **@mls** **=** **ST\_GeomFromText('MultiLineString((1** **1,2** **2,3** **3),(4** **4,5** **5))');**

mysql> **SELECT** **ST\_Length(@mls);**

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

| ST\_Length(@mls) |

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

| 4.242640687119286 |

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

mysql> **SET** **@ls** **=** **ST\_GeomFromText('LineString(1** **1,2** **2,3** **3)',** **4326);**

mysql> **SELECT** **ST\_Length(@ls);**

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

| ST\_Length(@ls) |

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

| 313701.9623204328 |

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

mysql> **SELECT** **ST\_Length(@ls,** **'metre');**

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

| ST\_Length(@ls, 'metre') |

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

| 313701.9623204328 |

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

mysql> **SELECT** **ST\_Length(@ls,** **'foot');**

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

| ST\_Length(@ls, 'foot') |

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

| 1029205.9131247795 |

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

• [ST\_NumPoints(*ls*)](#_bookmark339)

Returns the number of Point objects in the LineString value *ls*.

[ST\_NumPoints()](#_bookmark339) handles its arguments as described in the introduction to this section.

mysql> **SET** **@ls** **=** **'LineString(1** **1,2** **2,3** **3)';**

mysql> **SELECT** **ST\_NumPoints(ST\_GeomFromText(@ls));**

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

| ST\_NumPoints(ST\_GeomFromText(@ls)) |

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

| 3 |

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

• [ST\_PointN(*ls*, *N*)](#_bookmark345)

Returns the *N*-th Point in the Linestring value *ls*. Points are numbered beginning with 1.

[ST\_PointN()](#_bookmark345) handles its arguments as described in the introduction to this section.

mysql> **SET** **@ls** **=** **'LineString(1** **1,2** **2,3** **3)';**

mysql> **SELECT** **ST\_AsText(ST\_PointN(ST\_GeomFromText(@ls),2));**

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

| ST\_AsText(ST\_PointN(ST\_GeomFromText(@ls),2)) |

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

| POINT(2 2) |

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

• [ST\_StartPoint(*ls*)](#_bookmark350)

Returns the Point that is the start point of the LineString value *ls*. [ST\_StartPoint()](#_bookmark350) handles its arguments as described in the introduction to this section.

mysql> **SET** **@ls** **=** **'LineString(1** **1,2** **2,3** **3)';**

mysql> **SELECT** **ST\_AsText(ST\_StartPoint(ST\_GeomFromText(@ls)));**

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

| ST\_AsText(ST\_StartPoint(ST\_GeomFromText(@ls))) |

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

|  |  |
| --- | --- |
| | POINT(1 1) | | |

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

**12.17.7.4** **Polygon** **and** **MultiPolygon** **Property** **Functions**

Functions in this section return properties of Polygon or MultiPolygon values.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL or any geometry argument is an empty geometry, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• For functions that take multiple geometry arguments, if those arguments are not in the same SRS, an [ER\_GIS\_DIFFERENT\_SRIDS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_different_srids) error occurs.

• Otherwise, the return value is non-NULL.

These functions are available for obtaining polygon properties:

• [ST\_Area({*poly*|*mpoly*})](#_bookmark285)

Returns a double-precision number indicating the area of the Polygon or MultiPolygon argument, as measured in its spatial reference system.

As of MySQL 8.0.13, [ST\_Area()](#_bookmark285) handles its arguments as described in the introduction to this section, with these exceptions:

• If the geometry is geometrically invalid, either the result is an undefined area (that is, it can be any number), or an error occurs.

• If the geometry is valid but is not a Polygon or MultiPolygon object, an [ER\_UNEXPECTED\_GEOMETRY\_TYPE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unexpected_geometry_type) error occurs.

• If the geometry is a valid Polygon in a Cartesian SRS, the result is the Cartesian area of the polygon.

• If the geometry is a valid MultiPolygon in a Cartesian SRS, the result is the sum of the Cartesian area of the polygons.

• If the geometry is a valid Polygon in a geographic SRS, the result is the geodetic area of the polygon in that SRS, in square meters.

• If the geometry is a valid MultiPolygon in a geographic SRS, the result is the sum of geodetic area of the polygons in that SRS, in square meters.

• If an area computation results in +inf, an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

• If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

**'MultiPolygon(((0** **0,0** **3,3** **3,3** **0,0** **0),(1** **1,1** **2,2** **2,2** **1,1** **1)))';** mysql> **SELECT** **ST\_Area(ST\_GeomFromText(@mpoly));**

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

| ST\_Area(ST\_GeomFromText(@mpoly)) |

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

| 8 |

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

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

Prior to MySQL 8.0.13, [ST\_Area()](#_bookmark285) handles its arguments as described in the introduction to this section, with these exceptions:

• For arguments of dimension 0 or 1, the result is 0.

• If a geometry is empty, the return value is 0 rather than NULL.

• For a geometry collection, the result is the sum of the area values of all components. If the geometry collection is empty, its area is returned as 0.

• If the geometry has an SRID value for a geographic spatial reference system (SRS), an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

mysql> **SET** **@poly** **=**

**'Polygon((0** **0,0** **3,3** **0,0** **0),(1** **1,1** **2,2** **1,1** **1))';**

mysql> **SELECT** **ST\_Area(ST\_GeomFromText(@poly));**

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

| ST\_Area(ST\_GeomFromText(@poly)) |

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

| 4 |

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

mysql> **SET** **@mpoly** **=**

• [ST\_Centroid({*poly*|*mpoly*})](#_bookmark291)

Returns the mathematical centroid for the Polygon or MultiPolygon argument as a Point. The result is not guaranteed to be on the MultiPolygon.

This function processes geometry collections by computing the centroid point for components of highest dimension in the collection. Such components are extracted and made into a single MultiPolygon, MultiLineString, or MultiPoint for centroid computation.

[ST\_Centroid()](#_bookmark291) handles its arguments as described in the introduction to this section, with these exceptions:

• The return value is NULL for the additional condition that the argument is an empty geometry collection.

• If the geometry has an SRID value for a geographic spatial reference system (SRS), an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

mysql> **SET** **@poly** **=**

**ST\_GeomFromText('POLYGON((0** **0,10** **0,10** **10,0** **10,0** **0),(5** **5,7** **5,7** **7,5** **7,5** **5))');**

mysql> **SELECT** **ST\_GeometryType(@poly),ST\_AsText(ST\_Centroid(@poly));**

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

| ST\_GeometryType(@poly) | ST\_AsText(ST\_Centroid(@poly)) |

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

| POLYGON | POINT(4 .958333333333333 4 .958333333333333) |

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

• [ST\_ExteriorRing(*poly*)](#_bookmark304)

**'Polygon((0** **0,0** **3,3** **3,3** **0,0** **0),(1** **1,1** **2,2** **2,2** **1,1** **1))';** mysql> **SELECT** **ST\_AsText(ST\_ExteriorRing(ST\_GeomFromText(@poly)));**

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

| ST\_AsText(ST\_ExteriorRing(ST\_GeomFromText(@poly))) |

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

| LINESTRING(0 0,0 3,3 3,3 0,0 0) |

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

**'Polygon((0** **0,0** **3,3** **3,3** **0,0** **0),(1** **1,1** **2,2** **2,2** **1,1** **1))';** mysql> **SELECT** **ST\_AsText(ST\_InteriorRingN(ST\_GeomFromText(@poly),1));**

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

| ST\_AsText(ST\_InteriorRingN(ST\_GeomFromText(@poly),1)) |

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

| LINESTRING(1 1,1 2,2 2,2 1,1 1) |

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

**'Polygon((0** **0,0** **3,3** **3,3** **0,0** **0),(1** **1,1** **2,2** **2,2** **1,1** **1))';** mysql> **SELECT** **ST\_NumInteriorRings(ST\_GeomFromText(@poly));**

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

| ST\_NumInteriorRings(ST\_GeomFromText(@poly)) |

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

| 1 |

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

Returns the exterior ring of the Polygon value *poly* as a LineString. [ST\_ExteriorRing()](#_bookmark304) handles its arguments as described in the introduction to this section.

mysql> **SET** **@poly** **=**

• [ST\_InteriorRingN(*poly*, *N*)](#_bookmark315)

Returns the *N*-th interior ring for the Polygon value *poly* as a LineString. Rings are numbered beginning with 1.

[ST\_InteriorRingN()](#_bookmark315) handles its arguments as described in the introduction to this section.

mysql> **SET** **@poly** **=**

• [ST\_NumInteriorRing(*poly*)](#_bookmark338), [ST\_NumInteriorRings(*poly*)](#_bookmark338) Returns the number of interior rings in the Polygon value *poly*.

[ST\_NumInteriorRing()](#_bookmark338) and [ST\_NuminteriorRings()](#_bookmark338) handle their arguments as described in the introduction to this section.

mysql> **SET** **@poly** **=**

**12.17.7.5** **GeometryCollection** **Property** **Functions**

These functions return properties of GeometryCollection values.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL or any geometry argument is an empty geometry, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• Otherwise, the return value is non-NULL.

These functions are available for obtaining geometry collection properties:

• [ST\_GeometryN(*gc*, *N*)](#_bookmark309)

Returns the *N*-th geometry in the GeometryCollection value *gc*. Geometries are numbered beginning with 1.

[ST\_GeometryN()](#_bookmark309) handles its arguments as described in the introduction to this section.

mysql> **SET** **@gc** **=** **'GeometryCollection(Point(1** **1),LineString(2** **2,** **3** **3))';**

mysql> **SELECT** **ST\_AsText(ST\_GeometryN(ST\_GeomFromText(@gc),1));**

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

| ST\_AsText(ST\_GeometryN(ST\_GeomFromText(@gc),1)) |

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

| POINT(1 1) |

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

• [ST\_NumGeometries(*gc*)](#_bookmark337)

Returns the number of geometries in the GeometryCollection value *gc*.

[ST\_NumGeometries()](#_bookmark337) handles its arguments as described in the introduction to this section.

mysql> **SET** **@gc** **=** **'GeometryCollection(Point(1** **1),LineString(2** **2,** **3** **3))';**

mysql> **SELECT** **ST\_NumGeometries(ST\_GeomFromText(@gc));**

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

| ST\_NumGeometries(ST\_GeomFromText(@gc)) |

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

| 2 |

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

**12.17.8** **Spatial** **Operator** **Functions**

OpenGIS proposes a number of functions that can produce geometries. They are designed to implement spatial operators. These functions support all argument type combinations except those that are inapplicable according to the [Open Geospatial Consortium](http://www.opengeospatial.org) specification.

MySQL also implements certain functions that are extensions to OpenGIS, as noted in the function descriptions. In addition, [Section 12.17.7, “Geometry Property Functions”](#_bookmark363) , discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

• [ST\_Envelope(*g*)](#_bookmark302)

• [ST\_StartPoint(*ls*)](#_bookmark350)

• [ST\_EndPoint(*ls*)](#_bookmark301)

• [ST\_PointN(*ls*, *N*)](#_bookmark345)

• [ST\_ExteriorRing(*poly*)](#_bookmark304)

• [ST\_InteriorRingN(*poly*, *N*)](#_bookmark315)

• [ST\_GeometryN(*gc*, *N*)](#_bookmark309)

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• For functions that take multiple geometry arguments, if those arguments are not in the same SRS, an [ER\_GIS\_DIFFERENT\_SRIDS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_different_srids) error occurs.

• If any geometry argument has an SRID value for a geographic SRS and the function does not handle geographic geometries, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

• For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

• Otherwise, the return value is non-NULL. These spatial operator functions are available:

• [ST\_Buffer(*g*, *d* [, *strategy1* [, *strategy2* [, *strategy3*]]])](#_bookmark289)

Returns a geometry that represents all points whose distance from the geometry value *g* is less than or equal to a distance of *d*. The result is in the same SRS as the geometry argument.

If the geometry argument is empty, [ST\_Buffer()](#_bookmark289) returns an empty geometry.

If the distance is 0, [ST\_Buffer()](#_bookmark289) returns the geometry argument unchanged:

mysql> **SET** **@pt** **=** **ST\_GeomFromText('POINT(0** **0)');**

mysql> **SELECT** **ST\_AsText(ST\_Buffer(@pt,** **0));**

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

| ST\_AsText(ST\_Buffer(@pt, 0)) |

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

|  |  |
| --- | --- |
| | POINT(0 0) | | |

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

If the geometry argument is in a Cartesian SRS:

• [ST\_Buffer()](#_bookmark289) supports negative distances for Polygon and MultiPolygon values, and for geometry collections containing Polygon or MultiPolygon values.

• If the result is reduced so much that it disappears, the result is an empty geometry.

• An [ER\_WRONG\_ARGUMENTS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_arguments) error occurs for [ST\_Buffer()](#_bookmark289) with a negative distance for Point, MultiPoint, LineString, and MultiLineString values, and for geometry collections not containing any Polygon or MultiPolygon values.

If the geometry argument is in a geographic SRS:

• Prior to MySQL 8.0.26, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

• As of MySQL 8.0.26, Point geometries in a geographic SRS are permitted. For non-Point geometries, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error still occurs.

For MySQL versions that permit geographic Point geometries:

• If the distance is not negative and no strategies are specified, the function returns the geographic buffer of the Point in its SRS. The distance argument must be in the SRS distance unit (currently always meters).

• If the distance is negative or any strategy (except NULL) is specified, an [ER\_WRONG\_ARGUMENTS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_arguments) error occurs.

[ST\_Buffer()](#_bookmark289) permits up to three optional strategy arguments following the distance argument. Strategies influence buffer computation. These arguments are byte string values produced by the [ST\_Buffer\_Strategy()](#_bookmark290) function, to be used for point, join, and end strategies:

• Point strategies apply to Point and MultiPoint geometries. If no point strategy is specified, the default is [ST\_Buffer\_Strategy('point\_circle', 32)](#_bookmark290).

• Join strategies apply to LineString, MultiLineString, Polygon, and MultiPolygon geometries. If no join strategy is specified, the default is [ST\_Buffer\_Strategy('join\_round', 32)](#_bookmark290).

• End strategies apply to LineString and MultiLineString geometries. If no end strategy is specified, the default is [ST\_Buffer\_Strategy('end\_round', 32)](#_bookmark290).

Up to one strategy of each type may be specified, and they may be given in any order.

If the buffer strategies are invalid, an [ER\_WRONG\_ARGUMENTS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_arguments) error occurs. Strategies are invalid under any of these circumstances:

• Multiple strategies of a given type (point, join, or end) are specified.

• A value that is not a strategy (such as an arbitrary binary string or a number) is passed as a strategy.

• A Point strategy is passed and the geometry contains no Point or MultiPoint values.

• An end or join strategy is passed and the geometry contains no LineString, Polygon, MultiLinestring or MultiPolygon values.

mysql> **SET** **@pt** **=** **ST\_GeomFromText('POINT(0** **0)');**

mysql> **SET** **@pt\_strategy** **=** **ST\_Buffer\_Strategy('point\_square');**

mysql> **SELECT** **ST\_AsText(ST\_Buffer(@pt,** **2,** **@pt\_strategy));**

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

| ST\_AsText(ST\_Buffer(@pt, 2, @pt\_strategy)) |

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

| POLYGON((-2 -2,2 -2,2 2,-2 2,-2 -2)) |

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

mysql> **SET** **@ls** **=** **ST\_GeomFromText('LINESTRING(0** **0,0** **5,5** **5)');**

mysql> **SET** **@end\_strategy** **=** **ST\_Buffer\_Strategy('end\_flat');**

mysql> **SET** **@join\_strategy** **=** **ST\_Buffer\_Strategy('join\_round',** **10);**

mysql> **SELECT** **ST\_AsText(ST\_Buffer(@ls,** **5,** **@end\_strategy,** **@join\_strategy))**

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

| ST\_AsText(ST\_Buffer(@ls, 5, @end\_strategy, @join\_strategy)) |

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

| POLYGON((5 5,5 10,0 10,-3 .5355339059327373 8 .535533905932738, |

| -5 5,-5 0,0 0,5 0,5 5)) |

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

• [ST\_Buffer\_Strategy(*strategy* [, *points\_per\_circle*])](#_bookmark290)

This function returns a strategy byte string for use with [ST\_Buffer()](#_bookmark289) to influence buffer computation.

Information about strategies is available at [Boost.org](http://www.boost.org).

The first argument must be a string indicating a strategy option:

• For point strategies, permitted values are 'point\_circle' and 'point\_square'.

• For join strategies, permitted values are 'join\_round' and 'join\_miter'.

• For end strategies, permitted values are 'end\_round' and 'end\_flat'.

If the first argument is 'point\_circle', 'join\_round', 'join\_miter', or 'end\_round', the *points\_per\_circle* argument must be given as a positive numeric value. The maximum *points\_per\_circle* value is the value of the max\_points\_in\_geometry system variable.

For examples, see the description of [ST\_Buffer()](#_bookmark289).

[ST\_Buffer\_Strategy()](#_bookmark290) handles its arguments as described in the introduction to this section, with these exceptions:

• If any argument is invalid, an [ER\_WRONG\_ARGUMENTS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_arguments) error occurs.

• If the first argument is 'point\_square' or 'end\_flat', the *points\_per\_circle* argument must not be given or an [ER\_WRONG\_ARGUMENTS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_arguments) error occurs.

• [ST\_ConvexHull(*g*)](#_bookmark294)

Returns a geometry that represents the convex hull of the geometry value *g*.

This function computes a geometry's convex hull by first checking whether its vertex points are colinear. The function returns a linear hull if so, a polygon hull otherwise. This function processes geometry collections by extracting all vertex points of all components of the collection, creating a MultiPoint value from them, and computing its convex hull.

[ST\_ConvexHull()](#_bookmark294) handles its arguments as described in the introduction to this section, with this exception:

• The return value is NULL for the additional condition that the argument is an empty geometry collection.

mysql> **SET** **@g** **=** **'MULTIPOINT(5** **0,25** **0,15** **10,15** **25)';**

mysql> **SELECT** **ST\_AsText(ST\_ConvexHull(ST\_GeomFromText(@g)));**

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

| ST\_AsText(ST\_ConvexHull(ST\_GeomFromText(@g))) |

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

|  |  |
| --- | --- |
| | POLYGON((5 0,25 0,15 25,5 0)) | | |

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

• [ST\_Difference(*g1*, *g2*)](#_bookmark296)

Returns a geometry that represents the point set difference of the geometry values *g1* and *g2*. The result is in the same SRS as the geometry arguments.

As of MySQL 8.0.26, [ST\_Difference()](#_bookmark296) permits arguments in either a Cartesian or a geographic SRS. Prior to MySQL 8.0.26, [ST\_Difference()](#_bookmark296) permits arguments in a Cartesian SRS only; for arguments in a geographic SRS, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

[ST\_Difference()](#_bookmark296) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g1** **=** **Point(1,1),** **@g2** **=** **Point(2,2);**

mysql> **SELECT** **ST\_AsText(ST\_Difference(@g1,** **@g2));**

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

| ST\_AsText(ST\_Difference(@g1, @g2)) |

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

| POINT(1 1) |

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

• [ST\_Intersection(*g1*, *g2*)](#_bookmark316)

Returns a geometry that represents the point set intersection of the geometry values *g1* and *g2*. The result is in the same SRS as the geometry arguments.

As of MySQL 8.0.27, [ST\_Intersection()](#_bookmark316) permits arguments in either a Cartesian or a geographic SRS. Prior to MySQL 8.0.27, [ST\_Intersection()](#_bookmark316) permits arguments in a Cartesian SRS only; for

arguments in a geographic SRS, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs. [ST\_Intersection()](#_bookmark316) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g1** **=** **ST\_GeomFromText('LineString(1** **1,** **3** **3)');**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('LineString(1** **3,** **3** **1)');**

mysql> **SELECT** **ST\_AsText(ST\_Intersection(@g1,** **@g2));**

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

| ST\_AsText(ST\_Intersection(@g1, @g2)) |

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

|  |  |
| --- | --- |
| | POINT(2 2) | | |

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

• [ST\_LineInterpolatePoint(*ls*, *fractional\_distance*)](#_bookmark326)

This function takes a LineString geometry and a fractional distance in the range [0.0, 1.0] and returns the Point along the LineString at the given fraction of the distance from its start point to its endpoint. It can be used to answer questions such as which Point lies halfway along the road described by the geometry argument.

The function is implemented for LineString geometries in all spatial reference systems, both Cartesian and geographic.

If the *fractional\_distance* argument is 1.0, the result may not be exactly the last point of the LineString argument but a point close to it due to numerical inaccuracies in approximate-value computations.

A related function, [ST\_LineInterpolatePoints()](#_bookmark327), takes similar arguments but returns a MultiPoint consisting of Point values along the LineString at each fraction of the distance from its start point to its endpoint. For examples of both functions, see the [ST\_LineInterpolatePoints()](#_bookmark327) description.

[ST\_LineInterpolatePoint()](#_bookmark326) handles its arguments as described in the introduction to this section, with these exceptions:

• If the geometry argument is not a LineString, an [ER\_UNEXPECTED\_GEOMETRY\_TYPE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unexpected_geometry_type) error occurs.

• If the fractional distance argument is outside the range [0.0, 1.0], an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

[ST\_LineInterpolatePoint()](#_bookmark326) is a MySQL extension to OpenGIS. This function was added in MySQL 8.0.24.

• [ST\_LineInterpolatePoints(*ls*, *fractional\_distance*)](#_bookmark327)

This function takes a LineString geometry and a fractional distance in the range (0.0, 1.0] and returns the MultiPoint consisting of the LineString start point, plus Point values along the LineString at each fraction of the distance from its start point to its endpoint. It can be used to

answer questions such as which Point values lie every 10% of the way along the road described by the geometry argument.

The function is implemented for LineString geometries in all spatial reference systems, both Cartesian and geographic.

If the *fractional\_distance* argument divides 1.0 with zero remainder the result may not contain the last point of the LineString argument but a point close to it due to numerical inaccuracies in approximate-value computations.

A related function, [ST\_LineInterpolatePoint()](#_bookmark326), takes similar arguments but returns the Point along the LineString at the given fraction of the distance from its start point to its endpoint.

[ST\_LineInterpolatePoints()](#_bookmark327) handles its arguments as described in the introduction to this section, with these exceptions:

• If the geometry argument is not a LineString, an [ER\_UNEXPECTED\_GEOMETRY\_TYPE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unexpected_geometry_type) error occurs.

• If the fractional distance argument is outside the range [0.0, 1.0], an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

mysql> **SET** **@ls1** **=** **ST\_GeomFromText('LINESTRING(0** **0,0** **5,5** **5)');**

mysql> **SELECT** **ST\_AsText(ST\_LineInterpolatePoint(@ls1,** **.5));**

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

| ST\_AsText(ST\_LineInterpolatePoint(@ls1, .5)) |

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

| POINT(0 5) |

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

mysql> **SELECT** **ST\_AsText(ST\_LineInterpolatePoint(@ls1,** **.75));**

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

| ST\_AsText(ST\_LineInterpolatePoint(@ls1, .75)) |

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

| POINT(2 .5 5) |

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

mysql> **SELECT** **ST\_AsText(ST\_LineInterpolatePoint(@ls1,** **1));**

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

| ST\_AsText(ST\_LineInterpolatePoint(@ls1, 1)) |

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

| POINT(5 5) |

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

mysql> **SELECT** **ST\_AsText(ST\_LineInterpolatePoints(@ls1,** **.25));**

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

| ST\_AsText(ST\_LineInterpolatePoints(@ls1, .25)) |

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

| MULTIPOINT((0 2 .5),(0 5),(2 .5 5),(5 5)) |

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

[ST\_LineInterpolatePoints()](#_bookmark327) is a MySQL extension to OpenGIS. This function was added in MySQL 8.0.24.

• [ST\_PointAtDistance(*ls*, *distance*)](#_bookmark341)

This function takes a LineString geometry and a distance in the range [0.0, [ST\_Length(*ls*)](#_bookmark169)] measured in the unit of the spatial reference system (SRS) of the LineString, and returns the Point along the LineString at that distance from its start point. It can be used to answer

questions such as which Point value is 400 meters from the start of the road described by the geometry argument.

The function is implemented for LineString geometries in all spatial reference systems, both Cartesian and geographic.

[ST\_PointAtDistance()](#_bookmark341) handles its arguments as described in the introduction to this section, with these exceptions:

• If the geometry argument is not a LineString, an [ER\_UNEXPECTED\_GEOMETRY\_TYPE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unexpected_geometry_type) error occurs.

• If the fractional distance argument is outside the range [0.0, [ST\_Length(*ls*)](#_bookmark169)], an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

[ST\_PointAtDistance()](#_bookmark341) is a MySQL extension to OpenGIS. This function was added in MySQL

8.0.24.

• [ST\_SymDifference(*g1*, *g2*)](#_bookmark352)

Returns a geometry that represents the point set symmetric difference of the geometry values *g1* and *g2*, which is defined as:

*g1* symdifference *g2* := (*g1* union *g2*) difference (*g1* intersection *g2*)

Or, in function call notation:

ST\_SymDifference(*g1*, *g2*) = ST\_Difference(ST\_Union(*g1*, *g2*), ST\_Intersection(*g1*, *g2*))

The result is in the same SRS as the geometry arguments.

As of MySQL 8.0.27, [ST\_SymDifference()](#_bookmark352) permits arguments in either a Cartesian or a

geographic SRS. Prior to MySQL 8.0.27, [ST\_SymDifference()](#_bookmark352) permits arguments in a Cartesian SRS only; for arguments in a geographic SRS, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

[ST\_SymDifference()](#_bookmark352) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g1** **=** **ST\_GeomFromText('MULTIPOINT(5** **0,15** **10,15** **25)');**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('MULTIPOINT(1** **1,15** **10,15** **25)');**

mysql> **SELECT** **ST\_AsText(ST\_SymDifference(@g1,** **@g2));**

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

| ST\_AsText(ST\_SymDifference(@g1, @g2)) |

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

| MULTIPOINT((1 1),(5 0)) |

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

• [ST\_Transform(*g*, *target\_srid*)](#_bookmark354)

Transforms a geometry from one spatial reference system (SRS) to another. The return value is a geometry of the same type as the input geometry with all coordinates transformed to the target SRID, *target\_srid*. Prior to MySQL 8.0.30, transformation support was limited to geographic SRSs (unless the SRID of the geometry argument was the same as the target SRID value, in which case the return value was the input geometry for any valid SRS), and this function did not support Cartesian SRSs. Beginning with MySQL 8.0.30, support is provided for the Popular Visualisation Pseudo Mercator (EPSG 1024) projection method, used for WGS 84 Pseudo-Mercator (SRID 3857). In MySQL 8.0.32 and later, support is extended to all SRSs defined by EPSG except for those listed here:

• EPSG 1042 Krovak Modified

• EPSG 1043 Krovak Modified (North Orientated)

• EPSG 9816 Tunisia Mining Grid

• EPSG 9826 Lambert Conic Conformal (West Orientated)

[ST\_Transform()](#_bookmark354) handles its arguments as described in the introduction to this section, with these exceptions:

• Geometry arguments that have an SRID value for a geographic SRS do not produce an error.

• If the geometry or target SRID argument has an SRID value that refers to an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• If the geometry is in an SRS that [ST\_Transform()](#_bookmark354) cannot transform from, an [ER\_TRANSFORM\_SOURCE\_SRS\_NOT\_SUPPORTED](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_transform_source_srs_not_supported) error occurs.

• If the target SRID is in an SRS that [ST\_Transform()](#_bookmark354) cannot transform to, an [ER\_TRANSFORM\_TARGET\_SRS\_NOT\_SUPPORTED](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_transform_target_srs_not_supported) error occurs.

• If the geometry is in an SRS that is not WGS 84 and has no TOWGS84 clause, an [ER\_TRANSFORM\_SOURCE\_SRS\_MISSING\_TOWGS84](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_transform_source_srs_missing_towgs84) error occurs.

• If the target SRID is in an SRS that is not WGS 84 and has no TOWGS84 clause, an [ER\_TRANSFORM\_TARGET\_SRS\_MISSING\_TOWGS84](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_transform_target_srs_missing_towgs84) error occurs.

[ST\_SRID(*g*, *target\_srid*)](#_bookmark349) and [ST\_Transform(*g*, *target\_srid*)](#_bookmark354) differ as follows:

• [ST\_SRID()](#_bookmark349) changes the geometry SRID value without transforming its coordinates.

• [ST\_Transform()](#_bookmark354) transforms the geometry coordinates in addition to changing its SRID value.

mysql> **SET** **@p** **=** **ST\_GeomFromText('POINT(52** **.381389** **13** **.064444)',** **4326);**

mysql> **SELECT** **ST\_AsText(@p);**

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

| ST\_AsText(@p) |

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

| POINT(52 .381389 13 .064444) |

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

mysql> **SET** **@p** **=** **ST\_Transform(@p,** **4230);**

mysql> **SELECT** **ST\_AsText(@p);**

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

| ST\_AsText(@p) |

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

| POINT(52 .38208611407426 13 .065520672345304) |

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

• [ST\_Union(*g1*, *g2*)](#_bookmark355)

Returns a geometry that represents the point set union of the geometry values *g1* and *g2*. The result is in the same SRS as the geometry arguments.

As of MySQL 8.0.26, [ST\_Union()](#_bookmark355) permits arguments in either a Cartesian or a geographic SRS. Prior to MySQL 8.0.26, [ST\_Union()](#_bookmark355) permits arguments in a Cartesian SRS only; for arguments in a geographic SRS, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

[ST\_Union()](#_bookmark355) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g1** **=** **ST\_GeomFromText('LineString(1** **1,** **3** **3)');**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('LineString(1** **3,** **3** **1)');**

mysql> **SELECT** **ST\_AsText(ST\_Union(@g1,** **@g2));**

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

| ST\_AsText(ST\_Union(@g1, @g2)) |

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

| MULTILINESTRING((1 1,3 3),(1 3,3 1)) |

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

**12.17.9** **Functions** **That** **Test** **Spatial** **Relations** **Between** **Geometry** **Objects**

The functions described in this section take two geometries as arguments and return a qualitative or quantitative relation between them.

MySQL implements two sets of functions using function names defined by the OpenGIS specification. One set tests the relationship between two geometry values using precise object shapes, the other set uses object minimum bounding rectangles (MBRs).

**12.17.9.1** **Spatial** **Relation** **Functions** **That** **Use** **Object** **Shapes**

The OpenGIS specification defines the following functions to test the relationship between two geometry values *g1* and *g2*, using precise object shapes. The return values 1 and 0 indicate true and false, respectively, except that distance functions return distance values.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL or any geometry argument is an empty geometry, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• For functions that take multiple geometry arguments, if those arguments are not in the same SRS, an [ER\_GIS\_DIFFERENT\_SRIDS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_different_srids) error occurs.

• If any geometry argument is geometrically invalid, either the result is true or false (it is undefined which), or an error occurs.

• For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

• Otherwise, the return value is non-NULL.

Some functions in this section permit a unit argument that specifies the length unit for the return value. Unless otherwise specified, functions handle their unit argument as follows:

• A unit is supported if it is found in the INFORMATION\_SCHEMA ST\_UNITS\_OF\_MEASURE table. See Section 26.3.37, “The INFORMATION\_SCHEMA ST\_UNITS\_OF\_MEASURE Table” .

• If a unit is specified but not supported by MySQL, an [ER\_UNIT\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_unit_not_found) error occurs.

• If a supported linear unit is specified and the SRID is 0, an [ER\_GEOMETRY\_IN\_UNKNOWN\_LENGTH\_UNIT](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_in_unknown_length_unit) error occurs.

• If a supported linear unit is specified and the SRID is not 0, the result is in that unit.

• If a unit is not specified, the result is in the unit of the SRS of the geometries, whether Cartesian or geographic. Currently, all MySQL SRSs are expressed in meters.

These object-shape functions are available for testing geometry relationships:

• [ST\_Contains(*g1*, *g2*)](#_bookmark293)

Returns 1 or 0 to indicate whether *g1* completely contains *g2*. This tests the opposite relationship as [ST\_Within()](#_bookmark357).

[ST\_Contains()](#_bookmark293) handles its arguments as described in the introduction to this section. • [ST\_Crosses(*g1*, *g2*)](#_bookmark295)

Two geometries *spatially* *cross* if their spatial relation has the following properties:

• Unless *g1* and *g2* are both of dimension 1: *g1* crosses *g2* if the interior of *g2* has points in common with the interior of *g1*, but *g2* does not cover the entire interior of *g1*.

• If both *g1* and *g2* are of dimension 1: If the lines cross each other in a finite number of points (that is, no common line segments, only single points in common).

This function returns 1 or 0 to indicate whether *g1* spatially crosses *g2*.

[ST\_Crosses()](#_bookmark295) handles its arguments as described in the introduction to this section except that the return value is NULL for these additional conditions:

• *g1* is of dimension 2 (Polygon or MultiPolygon).

• *g2* is of dimension 1 (Point or MultiPoint).

• [ST\_Disjoint(*g1*, *g2*)](#_bookmark298)

Returns 1 or 0 to indicate whether *g1* is spatially disjoint from (does not intersect) *g2*.

[ST\_Disjoint()](#_bookmark298) handles its arguments as described in the introduction to this section.

• [ST\_Distance(*g1*, *g2* [, *unit*])](#_bookmark299)

Returns the distance between *g1* and *g2*, measured in the length unit of the spatial reference system (SRS) of the geometry arguments, or in the unit of the optional *unit* argument if that is specified.

This function processes geometry collections by returning the shortest distance among all combinations of the components of the two geometry arguments.

[ST\_Distance()](#_bookmark299) handles its geometry arguments as described in the introduction to this section, with these exceptions:

• [ST\_Distance()](#_bookmark299) detects arguments in a geographic (ellipsoidal) spatial reference system and returns the geodetic distance on the ellipsoid. As of MySQL 8.0.18, [ST\_Distance()](#_bookmark299) supports distance calculations for geographic SRS arguments of all geometry types. Prior to MySQL 8.0.18, the only permitted geographic argument types are Point and Point, or Point and MultiPoint (in any argument order). If called with other geometry type argument combinations in a geographic SRS, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

• If any argument is geometrically invalid, either the result is an undefined distance (that is, it can be any number), or an error occurs.

• If an intermediate or final result produces NaN or a negative number, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

As of MySQL 8.0.14, [ST\_Distance()](#_bookmark299) permits an optional *unit* argument that specifies the linear unit for the returned distance value. [ST\_Distance()](#_bookmark299) handles its *unit* argument as described in the introduction to this section.

mysql> **SET** **@g1** **=** **ST\_GeomFromText('POINT(1** **1)');**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('POINT(2** **2)');**

mysql> **SELECT** **ST\_Distance(@g1,** **@g2);**

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

| ST\_Distance(@g1, @g2) |

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

| 1.4142135623730951 |

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

mysql> **SET** **@g1** **=** **ST\_GeomFromText('POINT(1** **1)',** **4326);**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('POINT(2** **2)',** **4326);**

mysql> **SELECT** **ST\_Distance(@g1,** **@g2);**

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

| ST\_Distance(@g1, @g2) |

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

| 156874.3859490455 |

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

mysql> **SELECT** **ST\_Distance(@g1,** **@g2,** **'metre');**

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

| ST\_Distance(@g1, @g2, 'metre') |

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

| 156874.3859490455 |

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

mysql> **SELECT** **ST\_Distance(@g1,** **@g2,** **'foot');**

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

| ST\_Distance(@g1, @g2, 'foot') |

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

| 514679.7439273146 |

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

For the special case of distance calculations on a sphere, see the [ST\_Distance\_Sphere()](#_bookmark300) function.

• [ST\_Equals(*g1*, *g2*)](#_bookmark303)

Returns 1 or 0 to indicate whether *g1* is spatially equal to *g2*.

[ST\_Equals()](#_bookmark303) handles its arguments as described in the introduction to this section, except that it does not return NULL for empty geometry arguments.

mysql> **SET** **@g1** **=** **Point(1,1),** **@g2** **=** **Point(2,2);**

mysql> **SELECT** **ST\_Equals(@g1,** **@g1),** **ST\_Equals(@g1,** **@g2);**

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

| ST\_Equals(@g1, @g1) | ST\_Equals(@g1, @g2) |

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

| 1 | 0 |

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

• [ST\_FrechetDistance(*g1*, *g2* [, *unit*])](#_bookmark305)

Returns the discrete Fréchet distance between two geometries, reflecting how similar the geometries are. The result is a double-precision number measured in the length unit of the spatial reference system (SRS) of the geometry arguments, or in the length unit of the *unit* argument if that argument is given.

This function implements the discrete Fréchet distance, which means it is restricted to distances between the points of the geometries. For example, given two LineString arguments, only the

points explicitly mentioned in the geometries are considered. Points on the line segments between these points are not considered.

[ST\_FrechetDistance()](#_bookmark305) handles its geometry arguments as described in the introduction to this section, with these exceptions:

• The geometries may have a Cartesian or geographic SRS, but only LineString values are supported. If the arguments are in the same Cartesian or geographic SRS, but

either is not a LineString, an [ER\_NOT\_IMPLEMENTED\_FOR\_CARTESIAN\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_cartesian_srs) or [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs, depending on the SRS type.

[ST\_FrechetDistance()](#_bookmark305) handles its optional *unit* argument as described in the introduction to this section.

|  |  |
| --- | --- |
| mysql> **SET** **@ls1** **=** **ST\_GeomFromText('LINESTRING(0**  mysql> **SET** **@ls2** **=** **ST\_GeomFromText('LINESTRING(0**  mysql> **SELECT** **ST\_FrechetDistance(@ls1,** **@ls2);**  +--------------------------------+  | ST\_FrechetDistance(@ls1, @ls2) |  +--------------------------------+ | **0,0** **5,5** **5)');**  **1,0** **6,3** **3,5** **6)');** |

| 2.8284271247461903 |

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

mysql> **SET** **@ls1** **=** **ST\_GeomFromText('LINESTRING(0** **0,0** **5,5** **5)',** **4326);** mysql> **SET** **@ls2** **=** **ST\_GeomFromText('LINESTRING(0** **1,0** **6,3** **3,5** **6)',** **4326);** mysql> **SELECT** **ST\_FrechetDistance(@ls1,** **@ls2);**

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

| ST\_FrechetDistance(@ls1, @ls2) |

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

| 313421.1999416798 |

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

mysql> **SELECT** **ST\_FrechetDistance(@ls1,** **@ls2,** **'foot');**

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

| ST\_FrechetDistance(@ls1, @ls2, 'foot') |

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

| 1028284.7767115477 |

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

This function was added in MySQL 8.0.23.

• [ST\_HausdorffDistance(*g1*, *g2* [, *unit*])](#_bookmark314)

Returns the discrete Hausdorff distance between two geometries, reflecting how similar the geometries are. The result is a double-precision number measured in the length unit of the spatial reference system (SRS) of the geometry arguments, or in the length unit of the *unit* argument if that argument is given.

This function implements the discrete Hausdorff distance, which means it is restricted to distances between the points of the geometries. For example, given two LineString arguments, only the points explicitly mentioned in the geometries are considered. Points on the line segments between these points are not considered.

[ST\_HausdorffDistance()](#_bookmark314) handles its geometry arguments as described in the introduction to this section, with these exceptions:

• If the geometry arguments are in the same Cartesian or geographic SRS, but are not in a supported combination, an [ER\_NOT\_IMPLEMENTED\_FOR\_CARTESIAN\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_cartesian_srs) or

[ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs, depending on the SRS type. These combinations are supported:

• LineString and LineString

• Point and MultiPoint

• LineString and MultiLineString

• MultiPoint and MultiPoint

• MultiLineString and MultiLineString

[ST\_HausdorffDistance()](#_bookmark314) handles its optional *unit* argument as described in the introduction to this section.

|  |  |
| --- | --- |
| mysql> **SET** **@ls1** **=** **ST\_GeomFromText('LINESTRING(0**  mysql> **SET** **@ls2** **=** **ST\_GeomFromText('LINESTRING(0**  mysql> **SELECT** **ST\_HausdorffDistance(@ls1,** **@ls2);**  +----------------------------------+  | ST\_HausdorffDistance(@ls1, @ls2) |  +----------------------------------+ | **0,0** **5,5** **5)');**  **1,0** **6,3** **3,5** **6)');** |

| 1 |

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

mysql> **SET** **@ls1** **=** **ST\_GeomFromText('LINESTRING(0** **0,0** **5,5** **5)',** **4326);** mysql> **SET** **@ls2** **=** **ST\_GeomFromText('LINESTRING(0** **1,0** **6,3** **3,5** **6)',** **4326);** mysql> **SELECT** **ST\_HausdorffDistance(@ls1,** **@ls2);**

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

| ST\_HausdorffDistance(@ls1, @ls2) |

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

| 111319.49079326246 |

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

mysql> **SELECT** **ST\_HausdorffDistance(@ls1,** **@ls2,** **'foot');**

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

| ST\_HausdorffDistance(@ls1, @ls2, 'foot') |

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

| 365221.4264870815 |

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

This function was added in MySQL 8.0.23.

• [ST\_Intersects(*g1*, *g2*)](#_bookmark317)

Returns 1 or 0 to indicate whether *g1* spatially intersects *g2*.

[ST\_Intersects()](#_bookmark317) handles its arguments as described in the introduction to this section.

• [ST\_Overlaps(*g1*, *g2*)](#_bookmark340)

Two geometries *spatially* *overlap* if they intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

This function returns 1 or 0 to indicate whether *g1* spatially overlaps *g2*.

[ST\_Overlaps()](#_bookmark340) handles its arguments as described in the introduction to this section except that the return value is NULL for the additional condition that the dimensions of the two geometries are not equal.

• [ST\_Touches(*g1*, *g2*)](#_bookmark353)

Two geometries *spatially* *touch* if their interiors do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

This function returns 1 or 0 to indicate whether *g1* spatially touches *g2*.

[ST\_Touches()](#_bookmark353) handles its arguments as described in the introduction to this section except that the return value is NULL for the additional condition that both geometries are of dimension 0 (Point or MultiPoint).

• [ST\_Within(*g1*, *g2*)](#_bookmark357)

Returns 1 or 0 to indicate whether *g1* is spatially within *g2*. This tests the opposite relationship as [ST\_Contains()](#_bookmark293).

[ST\_Within()](#_bookmark357) handles its arguments as described in the introduction to this section.

**12.17.9.2** **Spatial** **Relation** **Functions** **That** **Use** **Minimum** **Bounding** **Rectangles**

MySQL provides several MySQL-specific functions that test the relationship between minimum bounding rectangles (MBRs) of two geometries *g1* and *g2*. The return values 1 and 0 indicate true and false, respectively.

The bounding box of a point is interpreted as a point that is both boundary and interior.

The bounding box of a straight horizontal or vertical line is interpreted as a line where the interior of the line is also boundary. The endpoints are boundary points.

If any of the parameters are geometry collections, the interior, boundary, and exterior of those parameters are those of the union of all elements in the collection.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL or an empty geometry, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• For functions that take multiple geometry arguments, if those arguments are not in the same SRS, an [ER\_GIS\_DIFFERENT\_SRIDS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_different_srids) error occurs.

• If any argument is geometrically invalid, either the result is true or false (it is undefined which), or an error occurs.

• For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

• Otherwise, the return value is non-NULL.

These MBR functions are available for testing geometry relationships:

• [MBRContains(*g1*, *g2*)](#_bookmark271)

Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* contains the minimum bounding rectangle of *g2*. This tests the opposite relationship as [MBRWithin()](#_bookmark279).

[MBRContains()](#_bookmark271) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g1** **=** **ST\_GeomFromText('Polygon((0** **0,0** **3,3** **3,3** **0,0** **0))');**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('Point(1** **1)');**

mysql> **SELECT** **MBRContains(@g1,@g2),** **MBRWithin(@g2,@g1);**

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

| MBRContains(@g1,@g2) | MBRWithin(@g2,@g1) |

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

| 1 | 1 |

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

• [MBRCoveredBy(*g1*, *g2*)](#_bookmark272)

Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* is covered by the minimum bounding rectangle of *g2*. This tests the opposite relationship as [MBRCovers()](#_bookmark273).

[MBRCoveredBy()](#_bookmark272) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g1** **=** **ST\_GeomFromText('Polygon((0** **0,0** **3,3** **3,3** **0,0** **0))');**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('Point(1** **1)');**

mysql> **SELECT** **MBRCovers(@g1,@g2),** **MBRCoveredby(@g1,@g2);**

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

| MBRCovers(@g1,@g2) | MBRCoveredby(@g1,@g2) |

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

| 1 | 0 |

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

mysql> **SELECT** **MBRCovers(@g2,@g1),** **MBRCoveredby(@g2,@g1);**

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

| MBRCovers(@g2,@g1) | MBRCoveredby(@g2,@g1) |

+--------------------+-----------------------+

| 0 | 1 |

+--------------------+-----------------------+

• [MBRCovers(*g1*, *g2*)](#_bookmark273)

Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* covers the minimum bounding rectangle of *g2*. This tests the opposite relationship as [MBRCoveredBy()](#_bookmark272). See the description of [MBRCoveredBy()](#_bookmark272) for examples.

[MBRCovers()](#_bookmark273) handles its arguments as described in the introduction to this section.

• [MBRDisjoint(*g1*, *g2*)](#_bookmark274)

Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

[MBRDisjoint()](#_bookmark274) handles its arguments as described in the introduction to this section.

• [MBREquals(*g1*, *g2*)](#_bookmark275)

Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are the same.

[MBREquals()](#_bookmark275) handles its arguments as described in the introduction to this section, except that it does not return NULL for empty geometry arguments.

• [MBRIntersects(*g1*, *g2*)](#_bookmark276)

Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* intersect.

[MBRIntersects()](#_bookmark276) handles its arguments as described in the introduction to this section.

• [MBROverlaps(*g1*, *g2*)](#_bookmark277)

Two geometries *spatially* *overlap* if they intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

This function returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* overlap.

[MBROverlaps()](#_bookmark277) handles its arguments as described in the introduction to this section.

• [MBRTouches(*g1*, *g2*)](#_bookmark278)

Two geometries *spatially* *touch* if their interiors do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

This function returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* touch.

[MBRTouches()](#_bookmark278) handles its arguments as described in the introduction to this section.

• [MBRWithin(*g1*, *g2*)](#_bookmark279)

Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* is within the minimum bounding rectangle of *g2*. This tests the opposite relationship as [MBRContains()](#_bookmark271).

[MBRWithin()](#_bookmark279) handles its arguments as described in the introduction to this section.

mysql> **SET** **@g1** **=** **ST\_GeomFromText('Polygon((0** **0,0** **3,3** **3,3** **0,0** **0))');**

mysql> **SET** **@g2** **=** **ST\_GeomFromText('Polygon((0** **0,0** **5,5** **5,5** **0,0** **0))');**

mysql> **SELECT** **MBRWithin(@g1,@g2),** **MBRWithin(@g2,@g1);**

+--------------------+--------------------+

| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |

+--------------------+--------------------+

| 1 | 0 |

+--------------------+--------------------+

**12.17.10** **Spatial** **Geohash** **Functions**

Geohash is a system for encoding latitude and longitude coordinates of arbitrary precision into a text string. Geohash values are strings that contain only characters chosen from

"0123456789bcdefghjkmnpqrstuvwxyz".

The functions in this section enable manipulation of geohash values, which provides applications the

capabilities of importing and exporting geohash data, and of indexing and searching geohash values. Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL, the return value is NULL.

• If any argument is invalid, an error occurs.

• If any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

• If any point argument does not have SRID 0 or 4326, an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs. *point* argument SRID validity is not checked.

• If any SRID argument refers to an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• If any SRID argument is not within the range of a 32-bit unsigned integer, an [ER\_DATA\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_data_out_of_range) error occurs.

• Otherwise, the return value is non-NULL. These geohash functions are available:

• [ST\_GeoHash(*longitude*, *latitude*, *max\_length*)](#_bookmark306), [ST\_GeoHash(*point*, *max\_length*)](#_bookmark306) Returns a geohash string in the connection character set and collation.

For the first syntax, the *longitude* must be a number in the range [−180, 180], and the *latitude* must be a number in the range [−90, 90]. For the second syntax, a POINT value is required, where the X and Y coordinates are in the valid ranges for longitude and latitude, respectively.

The resulting string is no longer than *max\_length* characters, which has an upper limit of 100. The string might be shorter than *max\_length* characters because the algorithm that creates the geohash value continues until it has created a string that is either an exact representation of the location or *max\_length* characters, whichever comes first.

[ST\_GeoHash()](#_bookmark306) handles its arguments as described in the introduction to this section.

mysql> **SELECT** **ST\_GeoHash(180,0,10),** **ST\_GeoHash(-180,-90,15);**

+----------------------+-------------------------+

| ST\_GeoHash(180,0,10) | ST\_GeoHash(-180,-90,15) |

+----------------------+-------------------------+

| xbpbpbpbpb | 000000000000000 |

+----------------------+-------------------------+

• [ST\_LatFromGeoHash(*geohash\_str*)](#_bookmark322)

Returns the latitude from a geohash string value, as a double-precision number in the range [−90, 90].

The [ST\_LatFromGeoHash()](#_bookmark322) decoding function reads no more than 433 characters from the *geohash\_str* argument. That represents the upper limit on information in the internal representation of coordinate values. Characters past the 433rd are ignored, even if they are otherwise illegal and produce an error.

[ST\_LatFromGeoHash()](#_bookmark322) handles its arguments as described in the introduction to this section.

mysql> **SELECT** **ST\_LatFromGeoHash(ST\_GeoHash(45,-20,10));**

+------------------------------------------+

| ST\_LatFromGeoHash(ST\_GeoHash(45,-20,10)) |

+------------------------------------------+

| -20 |

+------------------------------------------+

• [ST\_LongFromGeoHash(*geohash\_str*)](#_bookmark328)

Returns the longitude from a geohash string value, as a double-precision number in the range [−180, 180].

The remarks in the description of [ST\_LatFromGeoHash()](#_bookmark322) regarding the maximum number of characters processed from the *geohash\_str* argument also apply to [ST\_LongFromGeoHash()](#_bookmark328).

[ST\_LongFromGeoHash()](#_bookmark328) handles its arguments as described in the introduction to this section.

mysql> **SELECT** **ST\_LongFromGeoHash(ST\_GeoHash(45,-20,10));**

+-------------------------------------------+

| ST\_LongFromGeoHash(ST\_GeoHash(45,-20,10)) |

+-------------------------------------------+

| 45 |

+-------------------------------------------+

• [ST\_PointFromGeoHash(*geohash\_str*, *srid*)](#_bookmark342)

Returns a POINT value containing the decoded geohash value, given a geohash string value.

The X and Y coordinates of the point are the longitude in the range [−180, 180] and the latitude in the range [−90, 90], respectively.

The *srid* argument is an 32-bit unsigned integer.

The remarks in the description of [ST\_LatFromGeoHash()](#_bookmark322) regarding the maximum number of characters processed from the *geohash\_str* argument also apply to [ST\_PointFromGeoHash()](#_bookmark342).

[ST\_PointFromGeoHash()](#_bookmark342) handles its arguments as described in the introduction to this section.

mysql> **SET** **@gh** **=** **ST\_GeoHash(45,-20,10);**

mysql> **SELECT** **ST\_AsText(ST\_PointFromGeoHash(@gh,0));**

+---------------------------------------+

| ST\_AsText(ST\_PointFromGeoHash(@gh,0)) |

+---------------------------------------+

|  |  |
| --- | --- |
| | POINT(45 -20) | | |

+---------------------------------------+

**12.17.11** **Spatial** **GeoJSON** **Functions**

This section describes functions for converting between GeoJSON documents and spatial values. GeoJSON is an open standard for encoding geometric/geographical features. For more information, see <http://geojson.org>. The functions discussed here follow GeoJSON specification revision 1.0.

GeoJSON supports the same geometric/geographic data types that MySQL supports. Feature and FeatureCollection objects are not supported, except that geometry objects are extracted from them. CRS support is limited to values that identify an SRID.

MySQL also supports a native JSON data type and a set of SQL functions to enable operations on JSON values. For more information, see Section 11.5, “The JSON Data Type” , and Section 12.18,

“JSON Functions” .

• [ST\_AsGeoJSON(*g* [, *max\_dec\_digits* [, *options*]])](#_bookmark287)

Generates a GeoJSON object from the geometry *g*. The object string has the connection character set and collation.

If any argument is NULL, the return value is NULL. If any non-NULL argument is invalid, an error occurs.

*max\_dec\_digits*, if specified, limits the number of decimal digits for coordinates and causes rounding of output. If not specified, this argument defaults to its maximum value of 232 − 1. The minimum is 0.

*options*, if specified, is a bitmask. The following table shows the permitted flag values. If the geometry argument has an SRID of 0, no CRS object is produced even for those flag values that request one.

|  |  |
| --- | --- |
| **Flag** **Value** | **Meaning** |
| 0 | No options. This is the default if *options* is not specified. |
| 1 | Add a bounding box to the output. |
| 2 | Add a short-format CRS URN to the output. The default format is a short format (EPSG:*srid*). |
| 4 | Add a long-format CRS URN  (urn:ogc:def:crs:EPSG::*srid*). This flag overrides flag 2. For example, option values of 5 and 7 mean the same (add a bounding box and a long-format CRS URN). |

mysql> **SELECT** **ST\_AsGeoJSON(ST\_GeomFromText('POINT(11** **.11111** **12** **.22222)'),2);**

+-------------------------------------------------------------+

| ST\_AsGeoJSON(ST\_GeomFromText('POINT(11 .11111 12 .22222)'),2) |

+-------------------------------------------------------------+

| {"type": "Point", "coordinates": [11 .11, 12 .22]} |

+-------------------------------------------------------------+

• [ST\_GeomFromGeoJSON(*str* [, *options* [, *srid*]])](#_bookmark311)

Parses a string *str* representing a GeoJSON object and returns a geometry.

If any argument is NULL, the return value is NULL. If any non-NULL argument is invalid, an error occurs.

*options*, if given, describes how to handle GeoJSON documents that contain geometries with coordinate dimensions higher than 2. The following table shows the permitted *options* values.

|  |  |
| --- | --- |
| **Option** **Value** | **Meaning** |
| 1 | Reject the document and produce an error. This is the default if *options* is not specified. |
| 2, 3, 4 | Accept the document and strip off the coordinates for higher coordinate dimensions. |

*options* values of 2, 3, and 4 currently produce the same effect. If geometries with coordinate dimensions higher than 2 are supported in the future, you can expect these values to produce different effects.

The *srid* argument, if given, must be a 32-bit unsigned integer. If not given, the geometry return value has an SRID of 4326.

If *srid* refers to an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) error occurs.

• If a latitude value is not in the range [−90, 90], an [ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

GeoJSON geometry, feature, and feature collection objects may have a crs property. The parsing function parses named CRS URNs in the urn:ogc:def:crs:EPSG::*srid* and EPSG:*srid* namespaces, but not CRSs given as link objects. Also, urn:ogc:def:crs:OGC:1.3:CRS84 is

recognized as SRID 4326. If an object has a CRS that is not understood, an error occurs, with the exception that if the optional *srid* argument is given, any CRS is ignored even if it is invalid.

If a crs member that specifies an SRID different from the top-level object SRID is found at a lower level of the GeoJSON document, an [ER\_INVALID\_GEOJSON\_CRS\_NOT\_TOP\_LEVEL](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_invalid_geojson_crs_not_top_level) error occurs.

As specified in the GeoJSON specification, parsing is case-sensitive for the type member of the GeoJSON input (Point, LineString, and so forth). The specification is silent regarding case sensitivity for other parsing, which in MySQL is not case-sensitive.

This example shows the parsing result for a simple GeoJSON object. Observe that the order of coordinates depends on the SRID used.

mysql> **SET** **@json** **=** **'{** **"type":** **"Point",** **"coordinates":** **[102** **.0,** **0** **.0]}';**

mysql> **SELECT** **ST\_AsText(ST\_GeomFromGeoJSON(@json));**

+--------------------------------------+

| ST\_AsText(ST\_GeomFromGeoJSON(@json)) |

+--------------------------------------+

| POINT(0 102) |

+--------------------------------------+

mysql> **SELECT** **ST\_SRID(ST\_GeomFromGeoJSON(@json));**

+------------------------------------+

| ST\_SRID(ST\_GeomFromGeoJSON(@json)) |

+------------------------------------+

| 4326 |

+------------------------------------+

mysql> **SELECT** **ST\_AsText(ST\_SRID(ST\_GeomFromGeoJSON(@json),0));**

+-------------------------------------------------+

| ST\_AsText(ST\_SRID(ST\_GeomFromGeoJSON(@json),0)) |

+-------------------------------------------------+

| POINT(102 0) |

+-------------------------------------------------+

**12.17.12** **Spatial** **Aggregate** **Functions**

MySQL supports aggregate functions that perform a calculation on a set of values. For general information about these functions, see Section 12.20.1, “Aggregate Function Descriptions” . This section describes the [ST\_Collect()](#_bookmark292) spatial aggregate function.

[ST\_Collect()](#_bookmark292) can be used as a window function, as signified in its syntax description by [*over\_clause*], representing an optional OVER clause. *over\_clause* is described in

Section 12.21.2, “Window Function Concepts and Syntax” , which also includes other information about window function usage.

• [ST\_Collect([DISTINCT] *g*) [*over\_clause*]](#_bookmark292)

Aggregates geometry values and returns a single geometry collection value. With the DISTINCT option, returns the aggregation of the distinct geometry arguments.

As with other aggregate functions, GROUP BY may be used to group arguments into subsets. [ST\_Collect()](#_bookmark292) returns an aggregate value for each subset.

This function executes as a window function if *over\_clause* is present. *over\_clause* is as described in Section 12.21.2, “Window Function Concepts and Syntax” . In contrast to most aggregate functions that support windowing, [ST\_Collect()](#_bookmark292) permits use of *over\_clause* together with DISTINCT.

[ST\_Collect()](#_bookmark292) handles its arguments as follows:

• NULL arguments are ignored.

• If all arguments are NULL or the aggregate result is empty, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• If there are multiple geometry arguments and those arguments are in the same SRS, the return value is in that SRS. If those arguments are not in the same SRS, an [ER\_GIS\_DIFFERENT\_SRIDS\_AGGREGATION](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_different_srids_aggregation) error occurs.

• The result is the narrowest Multi*Xxx* or GeometryCollection value possible, with the result type determined from the non-NULL geometry arguments as follows:

• If all arguments are Point values, the result is a MultiPoint value.

• If all arguments are LineString values, the result is a MultiLineString value.

• If all arguments are Polygon values, the result is a MultiPolygon value.

• Otherwise, the arguments are a mix of geometry types and the result is a GeometryCollection value.

This example data set shows hypothetical products by year and location of manufacture:

CREATE TABLE product (

year INTEGER,

product VARCHAR(256),

location Geometry

);

INSERT INTO product

(year, product, location) VALUES

(2000, "Calculator", ST\_GeomFromText('point(60 -24)',4326)),

(2000, "Computer" , ST\_GeomFromText('point(28 -77)',4326)),

(2000, "Abacus" , ST\_GeomFromText('point(28 -77)',4326)),

(2000, "TV" , ST\_GeomFromText('point(38 60)',4326)),

(2001, "Calculator", ST\_GeomFromText('point(60 -24)',4326)),

(2001, "Computer" , ST\_GeomFromText('point(28 -77)',4326));

Some sample queries using [ST\_Collect()](#_bookmark292) on the data set:

mysql> **SELECT** **ST\_AsText(ST\_Collect(location))** **AS** **result**

**FROM** **product;**

+------------------------------------------------------------------+

| result |

+------------------------------------------------------------------+

| MULTIPOINT((60 -24),(28 -77),(28 -77),(38 60),(60 -24),(28 -77)) |

+------------------------------------------------------------------+

mysql> **SELECT** **ST\_AsText(ST\_Collect(DISTINCT** **location))** **AS** **result**

**FROM** **product;**

+---------------------------------------+

| result |

+---------------------------------------+

| MULTIPOINT((60 -24),(28 -77),(38 60)) |

+---------------------------------------+

mysql> **SELECT** **year,** **ST\_AsText(ST\_Collect(location))** **AS** **result**

**FROM** **product** **GROUP** **BY** **year;**

+------+------------------------------------------------+

| year | result |

+------+------------------------------------------------+

| 2000 | MULTIPOINT((60 -24),(28 -77),(28 -77),(38 60)) |

| 2001 | MULTIPOINT((60 -24),(28 -77)) |

+------+------------------------------------------------+

mysql> **SELECT** **year,** **ST\_AsText(ST\_Collect(DISTINCT** **location))** **AS** **result**

**FROM** **product** **GROUP** **BY** **year;**

+------+---------------------------------------+

| year | result |

+------+---------------------------------------+

| 2000 | MULTIPOINT((60 -24),(28 -77),(38 60)) |

| 2001 | MULTIPOINT((60 -24),(28 -77)) |

+------+---------------------------------------+

# selects nothing

mysql> **SELECT** **ST\_Collect(location)** **AS** **result**

**FROM** **product** **WHERE** **year** **=** **1999;**

+--------+

| result |

+--------+

| NULL |

+--------+

mysql> **SELECT** **ST\_AsText(ST\_Collect(location)**

**OVER** **(ORDER** **BY** **year,** **product** **ROWS** **BETWEEN** **1** **PRECEDING** **AND** **CURRENT** **ROW))**

**AS** **result**

**FROM** **product;**

+-------------------------------+

| result |

+-------------------------------+

| MULTIPOINT((28 -77)) |

| MULTIPOINT((28 -77),(60 -24)) |

| MULTIPOINT((60 -24),(28 -77)) |

| MULTIPOINT((28 -77),(38 60)) |

| MULTIPOINT((38 60),(60 -24)) |

| MULTIPOINT((60 -24),(28 -77)) |

+-------------------------------+

This function was added in MySQL 8.0.24.

**12.17.13** **Spatial** **Convenience** **Functions**

The functions in this section provide convenience operations on geometry values.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

• If any argument is NULL, the return value is NULL.

• If any geometry argument is not a syntactically well-formed geometry, an [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs.

• If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an [ER\_SRS\_NOT\_FOUND](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_srs_not_found) error occurs.

• For functions that take multiple geometry arguments, if those arguments are not in the same SRS, an [ER\_GIS\_DIFFERENT\_SRIDS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_different_srids) error occurs.

• Otherwise, the return value is non-NULL. These convenience functions are available:

• [ST\_Distance\_Sphere(*g1*, *g2* [, *radius*])](#_bookmark300)

Returns the minimum spherical distance between Point or MultiPoint arguments on a sphere, in meters. (For general-purpose distance calculations, see the [ST\_Distance()](#_bookmark299) function.) The optional *radius* argument should be given in meters.

If both geometry parameters are valid Cartesian Point or MultiPoint values in SRID 0, the return value is shortest distance between the two geometries on a sphere with the provided radius. If omitted, the default radius is 6,370,986 meters, Point X and Y coordinates are interpreted as longitude and latitude, respectively, in degrees.

If both geometry parameters are valid Point or MultiPoint values in a geographic spatial reference system (SRS), the return value is the shortest distance between the two geometries on a sphere with the provided radius. If omitted, the default radius is equal to the mean radius, defined as (2a+b)/3, where a is the semi-major axis and b is the semi-minor axis of the SRS.

[ST\_Distance\_Sphere()](#_bookmark300) handles its arguments as described in the introduction to this section, with these exceptions:

• Supported geometry argument combinations are Point and Point, or Point and MultiPoint (in any argument order). If at least one of the geometries is neither Point nor MultiPoint, and its SRID is 0, an [ER\_NOT\_IMPLEMENTED\_FOR\_CARTESIAN\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_cartesian_srs) error occurs. If at least one of the geometries is neither Point nor MultiPoint, and its SRID refers to a geographic SRS, an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs. If any geometry refers to a projected SRS, an [ER\_NOT\_IMPLEMENTED\_FOR\_PROJECTED\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_projected_srs) error occurs.

• If any argument has a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

• If the *radius* argument is present but not positive, an [ER\_NONPOSITIVE\_RADIUS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_nonpositive_radius) error occurs.

• If the distance exceeds the range of a double-precision number, an [ER\_STD\_OVERFLOW\_ERROR](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_std_overflow_error) error occurs.

mysql> **SET** **@pt1** **=** **ST\_GeomFromText('POINT(0** **0)');**

mysql> **SET** **@pt2** **=** **ST\_GeomFromText('POINT(180** **0)');**

mysql> **SELECT** **ST\_Distance\_Sphere(@pt1,** **@pt2);**

+--------------------------------+

| ST\_Distance\_Sphere(@pt1, @pt2) |

+--------------------------------+

| 20015042.813723423 |

+--------------------------------+

• [ST\_IsValid(*g*)](#_bookmark321)

Returns 1 if the argument is geometrically valid, 0 if the argument is not geometrically valid. Geometry validity is defined by the OGC specification.

The only valid empty geometry is represented in the form of an empty geometry collection value. [ST\_IsValid()](#_bookmark321) returns 1 in this case. MySQL does not support GIS EMPTY values such as POINT

EMPTY.

[ST\_IsValid()](#_bookmark321) handles its arguments as described in the introduction to this section, with this exception:

• If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

mysql> **SET** **@ls1** **=** **ST\_GeomFromText('LINESTRING(0** **0,-0** **.00** **0,0** **.0** **0)');**

mysql> **SET** **@ls2** **=** **ST\_GeomFromText('LINESTRING(0** **0,** **1** **1)');**

mysql> **SELECT** **ST\_IsValid(@ls1);**

+------------------+

| ST\_IsValid(@ls1) |

+------------------+

| 0 |

+------------------+

mysql> **SELECT** **ST\_IsValid(@ls2);**

+------------------+

| ST\_IsValid(@ls2) |

+------------------+

| 1 |

+------------------+

• [ST\_MakeEnvelope(*pt1*, *pt2*)](#_bookmark330)

Returns the rectangle that forms the envelope around two points, as a Point, LineString, or

Polygon.

Calculations are done using the Cartesian coordinate system rather than on a sphere, spheroid, or on earth.

Given two points *pt1* and *pt2*, [ST\_MakeEnvelope()](#_bookmark330) creates the result geometry on an abstract plane like this:

• If *pt1* and *pt2* are equal, the result is the point *pt1*.

• Otherwise, if (*pt1*, *pt2*) is a vertical or horizontal line segment, the result is the line segment (*pt1*, *pt2*).

• Otherwise, the result is a polygon using *pt1* and *pt2* as diagonal points. The result geometry has an SRID of 0.

[ST\_MakeEnvelope()](#_bookmark330) handles its arguments as described in the introduction to this section, with these exceptions:

• If the arguments are not Point values, an [ER\_WRONG\_ARGUMENTS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_arguments) error occurs.

• An [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error occurs for the additional condition that any coordinate value of the two points is infinite or NaN.

• If any geometry has an SRID value for a geographic spatial reference system (SRS), an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

mysql> **SET** **@pt1** **=** **ST\_GeomFromText('POINT(0** **0)');**

mysql> **SET** **@pt2** **=** **ST\_GeomFromText('POINT(1** **1)');**

mysql> **SELECT** **ST\_AsText(ST\_MakeEnvelope(@pt1,** **@pt2));**

+----------------------------------------+

| ST\_AsText(ST\_MakeEnvelope(@pt1, @pt2)) |

+----------------------------------------+

| POLYGON((0 0,1 0,1 1,0 1,0 0)) |

+----------------------------------------+

• [ST\_Simplify(*g*, *max\_distance*)](#_bookmark348)

Simplifies a geometry using the Douglas-Peucker algorithm and returns a simplified value of the same type.

The geometry may be any geometry type, although the Douglas-Peucker algorithm may not actually process every type. A geometry collection is processed by giving its components one by one to the simplification algorithm, and the returned geometries are put into a geometry collection as result.

The *max\_distance* argument is the distance (in units of the input coordinates) of a vertex to other segments to be removed. Vertices within this distance of the simplified linestring are removed.

According to Boost.Geometry, geometries might become invalid as a result of the simplification process, and the process might create self-intersections. To check the validity of the result, pass it to [ST\_IsValid()](#_bookmark321).

[ST\_Simplify()](#_bookmark348) handles its arguments as described in the introduction to this section, with this exception:

• If the *max\_distance* argument is not positive, or is NaN, an [ER\_WRONG\_ARGUMENTS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_arguments) error occurs.

mysql> **SET** **@g** **=** **ST\_GeomFromText('LINESTRING(0** **0,0** **1,1** **1,1** **2,2** **2,2** **3,3** **3)');**

mysql> **SELECT** **ST\_AsText(ST\_Simplify(@g,** **0** **.5));**

+---------------------------------+

| ST\_AsText(ST\_Simplify(@g, 0 .5)) |

+---------------------------------+

| LINESTRING(0 0,0 1,1 1,2 3,3 3) |

+---------------------------------+

mysql> **SELECT** **ST\_AsText(ST\_Simplify(@g,** **1** **.0));**

+---------------------------------+

| ST\_AsText(ST\_Simplify(@g, 1 .0)) |

+---------------------------------+

|  |  |
| --- | --- |
| | LINESTRING(0 0,3 3) | | |

+---------------------------------+

• [ST\_Validate(*g*)](#_bookmark356)

Validates a geometry according to the OGC specification. A geometry can be syntactically well- formed (WKB value plus SRID) but geometrically invalid. For example, this polygon is geometrically invalid: POLYGON((0 0, 0 0, 0 0, 0 0, 0 0))

[ST\_Validate()](#_bookmark356) returns the geometry if it is syntactically well-formed and is geometrically valid, NULL if the argument is not syntactically well-formed or is not geometrically valid or is NULL.

[ST\_Validate()](#_bookmark356) can be used to filter out invalid geometry data, although at a cost. For applications that require more precise results not tainted by invalid data, this penalty may be worthwhile.

If the geometry argument is valid, it is returned as is, except that if an input Polygon or MultiPolygon has clockwise rings, those rings are reversed before checking for validity. If the geometry is valid, the value with the reversed rings is returned.

The only valid empty geometry is represented in the form of an empty geometry collection value. [ST\_Validate()](#_bookmark356) returns it directly without further checks in this case.

As of MySQL 8.0.13, [ST\_Validate()](#_bookmark356) handles its arguments as described in the introduction to this section, with these exceptions:

• If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:

• If a longitude value is not in the range (−180, 180], an [ER\_GEOMETRY\_PARAM\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_longitude_out_of_range) error occurs ([ER\_LONGITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_longitude_out_of_range) prior to MySQL 8.0.12).

• If a latitude value is not in the range [−90, 90], an [ER\_GEOMETRY\_PARAM\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_geometry_param_latitude_out_of_range) error occurs ([ER\_LATITUDE\_OUT\_OF\_RANGE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_latitude_out_of_range) prior to MySQL 8.0.12).

Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

Prior to MySQL 8.0.13, [ST\_Validate()](#_bookmark356) handles its arguments as described in the introduction to this section, with these exceptions:

• If the geometry is not syntactically well-formed, the return value is NULL. An [ER\_GIS\_INVALID\_DATA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_gis_invalid_data) error does not occur.

• If the geometry has an SRID value for a geographic spatial reference system (SRS), an [ER\_NOT\_IMPLEMENTED\_FOR\_GEOGRAPHIC\_SRS](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_not_implemented_for_geographic_srs) error occurs.

|  |  |
| --- | --- |
| mysql> **SET** **@ls1** **=** **ST\_GeomFromText('LINESTRING(0**  mysql> **SET** **@ls2** **=** **ST\_GeomFromText('LINESTRING(0**  mysql> **SELECT** **ST\_AsText(ST\_Validate(@ls1));**  +------------------------------+  | ST\_AsText(ST\_Validate(@ls1)) |  +------------------------------+ | **0)');**  **0,** **1** **1)');** |

|  |  |
| --- | --- |
| | NULL | | |