

and collation (see [Section 10.1.4, “Connection Character Sets and Collations”](#)). This applies only to `CAST()`, `CONV()`, `FORMAT()`, `HEX()`, and `SPACE()`.

If there is any question about the character set or collation of the result returned by a string function, use the `CHARSET()` or `COLLATION()` function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+-----+-----+-----+
| USER()          | CHARSET(USER()) | COLLATION(USER()) |
+-----+-----+-----+
| test@localhost  | utf8            | utf8_general_ci   |
+-----+-----+-----+
mysql> SELECT CHARSET(COMPRESS('abc')), COLLATION(COMPRESS('abc'));
+-----+-----+
| CHARSET(COMPRESS('abc')) | COLLATION(COMPRESS('abc')) |
+-----+-----+
| binary                  | binary                     |
+-----+-----+
```

12.6 Numeric Functions and Operators

Table 12.10 Numeric Functions and Operators

Name	Description
<code>ABS()</code>	Return the absolute value
<code>ACOS()</code>	Return the arc cosine
<code>ASIN()</code>	Return the arc sine
<code>ATAN()</code>	Return the arc tangent
<code>ATAN2(), ATAN()</code>	Return the arc tangent of the two arguments
<code>CEIL()</code>	Return the smallest integer value not less than the argument
<code>CEILING()</code>	Return the smallest integer value not less than the argument
<code>CONV()</code>	Convert numbers between different number bases
<code>COS()</code>	Return the cosine
<code>COT()</code>	Return the cotangent
<code>CRC32()</code>	Compute a cyclic redundancy check value
<code>DEGREES()</code>	Convert radians to degrees
<code>DIV</code>	Integer division
<code>/</code>	Division operator
<code>EXP()</code>	Raise to the power of
<code>FLOOR()</code>	Return the largest integer value not greater than the argument
<code>LN()</code>	Return the natural logarithm of the argument
<code>LOG()</code>	Return the natural logarithm of the first argument
<code>LOG10()</code>	Return the base-10 logarithm of the argument
<code>LOG2()</code>	Return the base-2 logarithm of the argument
<code>-</code>	Minus operator
<code>MOD()</code>	Return the remainder
<code>%, MOD</code>	Modulo operator
<code>PI()</code>	Return the value of pi
<code>+</code>	Addition operator
<code>POW()</code>	Return the argument raised to the specified power

Name	Description
<code>POWER ()</code>	Return the argument raised to the specified power
<code>RADIANS ()</code>	Return argument converted to radians
<code>RAND ()</code>	Return a random floating-point value
<code>ROUND ()</code>	Round the argument
<code>SIGN ()</code>	Return the sign of the argument
<code>SIN ()</code>	Return the sine of the argument
<code>SQRT ()</code>	Return the square root of the argument
<code>TAN ()</code>	Return the tangent of the argument
<code>*</code>	Multiplication operator
<code>TRUNCATE ()</code>	Truncate to specified number of decimal places
<code>-</code>	Change the sign of the argument

12.6.1 Arithmetic Operators

Table 12.11 Arithmetic Operators

Name	Description
<code>DIV</code>	Integer division
<code>/</code>	Division operator
<code>-</code>	Minus operator
<code>%, MOD</code>	Modulo operator
<code>+</code>	Addition operator
<code>*</code>	Multiplication operator
<code>-</code>	Change the sign of the argument

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

- In the case of `-`, `+`, and `*`, 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 `+`, `-`, `/`, `*`, `%` 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 `/`, 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 12.10, “Cast Functions and Operators”](#).

For information about handling of overflow in numeric expression evaluation, see [Section 11.2.6, “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()`; see [Section 12.7, “Date and Time Functions”](#).

- `+`

Addition:

```
mysql> SELECT 3+5;
-> 8
```

- `-`

Subtraction:

```
mysql> SELECT 3-5;
-> -2
```

- `-`

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

```
mysql> SELECT - 2;
-> -2
```

**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 -2^{63} .

- `*`

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 11.2, “Numeric Types”](#).)

- `/`

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`

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

As of MySQL 5.5.3, 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. Before MySQL 5.5.3, incorrect results may occur for noninteger operands that exceed [BIGINT](#) range.

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

- $N \% M, N \text{ MOD } M$

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

12.6.2 Mathematical Functions

Table 12.12 Mathematical Functions

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CONV()	Convert numbers between different number bases
COS()	Return the cosine
COT()	Return the cotangent
CRC32()	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
MOD()	Return the remainder
PI()	Return the value of pi
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument

Name	Description
<code>SQRT()</code>	Return the square root of the argument
<code>TAN()</code>	Return the tangent of the argument
<code>TRUNCATE()</code>	Truncate to specified number of decimal places

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

- `ABS(X)`

Returns the absolute value of `X`.

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

This function is safe to use with `BIGINT` values.

- `ACOS(X)`

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`.

```
mysql> SELECT ACOS(1);
-> 0
mysql> SELECT ACOS(1.0001);
-> NULL
mysql> SELECT ACOS(0);
-> 1.5707963267949
```

- `ASIN(X)`

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`.

```
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)`

Returns the arc tangent of `X`, that is, the value whose tangent is `X`.

```
mysql> SELECT ATAN(2);
-> 1.1071487177941
mysql> SELECT ATAN(-2);
-> -1.1071487177941
```

- `ATAN(Y, X)`, `ATAN2(Y, X)`

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.

```
mysql> SELECT ATAN(-2,2);
-> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
-> 1.5707963267949
```

- `CEIL(X)`

`CEIL()` is a synonym for `CEILING()`.

- `CEILING(X)`

Returns the smallest integer value not less than *X*.

```
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)`

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()` works with 64-bit precision.

```
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)`

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

```
mysql> SELECT COS(PI());
-> -1
```

- `COT(X)`

Returns the cotangent of *X*.

```
mysql> SELECT COT(12);
-> -1.5726734063977
mysql> SELECT COT(0);
-> NULL
```

- `CRC32(expr)`

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)`

Returns the argument `X`, converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
-> 180
mysql> SELECT DEGREES(PI() / 2);
-> 90
```

- `EXP(X)`

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

```
mysql> SELECT EXP(2);
-> 7.3890560989307
mysql> SELECT EXP(-2);
-> 0.13533528323661
mysql> SELECT EXP(0);
-> 1
```

- `FLOOR(X)`

Returns the largest integer value not greater than `X`.

```
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)`

Formats the number `X` to a format like `'#,###,###.##'`, rounded to `D` decimal places, and returns the result as a string. For details, see [Section 12.5, “String Functions”](#).

- `HEX(N_or_S)`

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 12.5, “String Functions”](#), for details.

- `LN(X)`

Returns the natural logarithm of `X`; that is, the base-*e* logarithm of `X`. If `X` is less than or equal to 0, then `NULL` is returned.

```
mysql> SELECT LN(2);
-> 0.69314718055995
mysql> SELECT LN(-2);
-> NULL
```

This function is synonymous with `LOG(X)`. The inverse of this function is the `EXP()` function.

- `LOG(X)`, `LOG(B,X)`

If called with one parameter, this function returns the natural logarithm of *X*. If *X* is less than or equal to 0, then `NULL` is returned.

The inverse of this function (when called with a single argument) is the `EXP()` 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)` is equivalent to `LOG(X) / LOG(B)`.

- `LOG2(X)`

Returns the base-2 logarithm of *X*.

```
mysql> SELECT LOG2(65536);
-> 16
mysql> SELECT LOG2(-100);
-> NULL
```

`LOG2()` is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression `LOG(X) / LOG(2)`.

- `LOG10(X)`

Returns the base-10 logarithm of *X*.

```
mysql> SELECT LOG10(2);
-> 0.30102999566398
mysql> SELECT LOG10(100);
-> 2
mysql> SELECT LOG10(-100);
-> NULL
```

`LOG10(X)` is equivalent to `LOG(10,X)`.

- `MOD(N,M)`, `N % M`, `N MOD M`

Modulo operation. Returns the remainder of *N* divided by *M*.

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



```
-> 2
```

This function is safe to use with `BIGINT` values.

`MOD()` 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)` returns `NULL`.

- `PI()`

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()+0.00000000000000000000;
-> 3.141592653589793116
```

- `POW(X, Y)`

Returns the value of `X` raised to the power of `Y`.

```
mysql> SELECT POW(2,2);
-> 4
mysql> SELECT POW(2,-2);
-> 0.25
```

- `POWER(X, Y)`

This is a synonym for `POW()`.

- `RADIANS(X)`

Returns the argument `X`, converted from degrees to radians. (Note that π radians equals 180 degrees.)

```
mysql> SELECT RADIANS(90);
-> 1.5707963267949
```

- `RAND([N])`

Returns a random floating-point value `v` in the range $0 \leq v < 1.0$. To obtain a random integer `R` in the range $i \leq R < j$, use the expression `FLOOR(i + RAND() * (j - i))`. For example, to obtain a random integer in the range the range $7 \leq R < 12$, 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()`.

One implication of this behavior is that for equal argument values, `RAND(N)` 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

mysql> SELECT i, RAND() FROM t;
+-----+-----+
| i      | RAND() |
+-----+-----+
| 1      | 0.61914388706828 |
| 2      | 0.93845168309142 |
| 3      | 0.83482678498591 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+-----+-----+
| i      | RAND(3) |
+-----+-----+
| 1      | 0.90576975597606 |
| 2      | 0.37307905813035 |
| 3      | 0.14808605345719 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND() FROM t;
+-----+-----+
| i      | RAND() |
+-----+-----+
| 1      | 0.35877890638893 |
| 2      | 0.28941420772058 |
| 3      | 0.37073435016976 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+-----+-----+
| i      | RAND(3) |
+-----+-----+
| 1      | 0.90576975597606 |
| 2      | 0.37307905813035 |
| 3      | 0.14808605345719 |
+-----+-----+
3 rows in set (0.01 sec)
```

`RAND()` 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()` is not a constant value and cannot be used for index optimizations. For more information, see [Section 8.2.1.16, “Function Call Optimization”](#).

Use of a column with `RAND()` values in an `ORDER BY` or `GROUP BY` clause may yield unexpected results because for either clause a `RAND()` 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()` 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. Beginning with MySQL 5.5.2, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #49222)

- `ROUND(X), ROUND(X,D)`

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.

```
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
```

The return type is the same type as that of 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()` uses the following rules depending on the type of the first argument:

- For exact-value numbers, `ROUND()` 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()` uses the “round to nearest even” rule: A value with any fractional part 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 12.18, “Precision Math”](#).

- `SIGN(X)`

Returns the sign of the argument as -1 , 0 , or 1 , depending on whether x is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
-> -1
mysql> SELECT SIGN(0);
-> 0
mysql> SELECT SIGN(234);
-> 1
```

- `SIN(x)`

Returns the sine of x , where x is given in radians.

```
mysql> SELECT SIN(PI());
-> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
-> 0
```

- `SQRT(x)`

Returns the square root of a nonnegative number x .

```
mysql> SELECT SQRT(4);
-> 2
mysql> SELECT SQRT(20);
-> 4.4721359549996
mysql> SELECT SQRT(-16);
-> NULL
```

- `TAN(x)`

Returns the tangent of x , where x is given in radians.

```
mysql> SELECT TAN(PI());
-> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
-> 1.5574077246549
```

- `TRUNCATE(x , D)`

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.

```
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.

12.7 Date and Time Functions