18 Operators and Functions

Like any other programming language, SQL carries among its core elements operators and named procedures. This reference lists all of those operators and functions and explains how they evaluate into useful expressions.

Operators

MySQL operators may be divided into three kinds of operators: arithmetic, comparison, and logical.

Rules of Precedence

When your SQL contains complex expressions, the sub-expressions are evaluated based on MySQL's rules of precedence. Of course, you may always override MySQL's rules of precedence by enclosing an expression in parentheses.

1. BINARY
2. NOT
3. - (unary minus)
4. * / %
5. + -
6. << >>
7. &
8. |
9. < <= > >= = <=> <> IN IS LIKE REGEXP
10. BETWEEN

11. AND
12. OR

Arithmetic Operators

Arithmetic operators perform basic arithmetic on two values.

`+`	Adds two numerical values
`-`	Subtracts two numerical values
`*`	Multiplies two numerical values
`/`	Divides two numerical values
`%`	Gives the modulo of two numerical values
`|`	Performs a bitwise OR on two integer values
`&`	Performs a bitwise AND on two integer values
`<<`	Performs a bitwise left shift on an integer value
`>>`	Performs a bitwise right shift on an integer value

Comparison Operators

Comparison operators compare values and return 1 if the comparison is true, 0 otherwise. Except for the <==> operator, NULL values cause a comparison operator to evaluate to NULL.

`<>` *or* `!=`
: Match rows if the two values are not equal.

`<=`
: Match rows if the left value is less than or equal to the right value.

`<`
: Match rows if the left value is less than the right value.

`>=`
: Match rows if the left value is greater than or equal to the right value.

`>`
: Match rows if the left value is greater than the right value.

`value` `BETWEEN` *`value1`* `AND` *`value2`*
: Match rows if *value* is between *value1* and *value2*, or equal to one of them.

`value` `IN (`*`value1,value2,`*`...)`
: Match rows if *value* is among the values listed.

`value NOT IN (value1, value2,...)`
: Match rows if *value* is not among the values listed.

`value1 LIKE value2`
: Compares *value1* to *value2* and matches the rows if they match. The right-hand value can contain the wildcard `'%'` which matches any number of characters (including 0) and `'_'` which matches exactly one character. This is probably the single most used comparison in SQL. The most common usage is to compare a field value with a literal containing a wildcard (e.g., `SELECT name FROM people WHERE name LIKE 'B%'`).

`value1 NOT LIKE value2`
: Compares *value1* to *value2* and matches the rows if they differ. This is identical to `NOT (value1 LIKE value2)`.

`value1 REGEXP/RLIKE value2`
: Compares *value1* to *value2* using the extended regular expression syntax and matches the rows if they match. The right hand value can contain full Unix regular expression wildcards and constructs (e.g., `SELECT name FROM people WHERE name RLIKE '^B.*'`).

`value1 NOT REGEXP value2`
: Compares *value1* to *value2* using the extended regular expression syntax and matches the rows if they differ. This is identical to `NOT (value1 REXEXP value2)`.

Logical Operators

Logical operators check the truth value of one or more expressions. In SQL terms, a logical operator checks whether its operands are 0, non-zero, or `NULL`. A 0 value means false, non-zero true, and `NULL` means no value.

`NOT` *or* `!`
: Performs a logical not (returns 1 if the value is 0 and returns 0 otherwise).

`OR` *or* `||`
: Performs a logical or (returns 1 if any of the arguments are not 0, otherwise returns 0)

`AND` *or* `&&`
: Performs a logical and (returns 0 if any of the arguments are 0, otherwise returns 1)

Functions

MySQL provides built-in functions that perform special operations.

Aggregate Functions

Aggregate functions operate on a set of data. The usual method of using these is to perform some action on a complete set of returned rows. For example, `SELECT AVG(height) FROM kids` would return the average of all of the values of the `height` field in the `kids` table.

`AVG(`*`expression`*`)`
: Returns the average value of the values in *expression* (e.g., `SELECT AVG(score) FROM tests`).

`BIT_AND(`*`expression`*`)`
: Returns the bitwise `AND` aggregate of all of the values in *expression* (e.g., `SELECT BIT_AND(flags) FROM options`).

`BIT_OR(`*`expression`*`)`
: Returns the bitwise `OR` aggregate of all of the values in *expression* (e.g., `SELECT BIT_OR(flags) FROM options`).

`COUNT(`*`expression`*`)`
: Returns the number of times *expression* was not null. `COUNT(*)` will return the number of rows with some data in the entire table (e.g., `SELECT COUNT(*) FROM folders`).

`MAX(`*`expression`*`)`
: Returns the largest of the values in *expression* (e.g., `SELECT MAX (elevation) FROM mountains`).

`MIN(`*`expression`*`)`
: Returns the smallest of the values in *expression* (e.g., `SELECT MIN(level) FROM toxic_waste`).

`STD(`*`expression`*`)`/`STDDEV(`*`expression`*`)`
: Returns the standard deviation of the values in *expression* (e.g., `SELECT STDDEV(points) FROM data`).

`SUM(`*`expression`*`)`

Returns the sum of the values in *expression* (e.g., `SELECT SUM(calories) FROM daily_diet`).

General Functions

General functions operate on one or more discreet values.

`ABS(`*`number`*`)`
: Returns the absolute value of *number* (e.g., `ABS(-10)` returns 10).

`ACOS(`*`number`*`)`
: Returns the inverse cosine of *number* in radians (e.g., `ACOS(0)` returns 1. 570796).

`ASCII(`*`char`*`)`
: Returns the ASCII value of the given character (e.g., `ASCII('h')` returns 104).

`ASIN(`*`number`*`)`
: Returns the inverse sine of number in radians (e.g., `ASIN(0)` returns 0. 000000).

`ATAN(`*`number`*`)`
: Returns the inverse tangent of number in radians (e.g., `ATAN(1)` returns 0. 785398.)

`ATAN2(`*`X, Y`*`)`
: Returns the inverse tangent of the point (*X*,*Y*) (e.g., `ATAN(-3,3)` returns -0. 785398).

`BIN(`*`decimal`*`)`
: Returns the binary value of the given decimal number. This is equivalent to the function `CONV(decimal,10,2)` (e.g., `BIN(8)` returns 1000).

`BIT_COUNT(`*`number`*`)`
: Returns the number of bits that are set to 1 in the binary representation of the number (e.g., `BIT_COUNT(17)` returns 2).

`CEILING(`*`number`*`)`
: Returns the smallest integer larger than or equal to *number* (e.g., `CEILING (5. 67)` returns 6).

`CHAR(`*`num1`*`[,`*`num2`*`,. . .])`
: Returns a string made from converting each of the numbers to the character corresponding to that ASCII value (e.g., `CHAR(122)` returns 'z').

`COALESCE(`*`expr1`*`, `*`expr2`*`, ...)`
: Returns the first non-null expression in the list (e.g., `COALESCE(NULL, NULL, 'cheese', 2)` returns 3).

`CONCAT(`*`string1`*`,`*`string2`*`[,`*`string3`*`,. . .])`
: Returns the string formed by joining together all of the arguments (e.g., `CONCAT('Hi',' ','Mom','!')` returns "Hi Mom!").

`CONV(`*`number, base1, base2`*`)`
: Returns the value of *number* converted from *base1* to *base2*. *Number* must be an integer value (either as a bare number or as a string). The bases can be any integer from 2 to 36 (e.g., `CONV(8,10,2)` returns 1000 (the number 8 in decimal converted to binary)).

`COS(`*`radians`*`)`

Returns the cosine of the given number, which is in radians (e.g., `COS(0)` returns 1.000000).

`COT(`*`radians`*`)`

Returns the cotangent of the given number, which must be in radians (e.g., `COT(1)` returns 0.642093).

`CURDATE()`/`CURRENT_DATE()`

Returns the current date. A number of the form YYYYMMDD is returned if this is used in a numerical context, otherwise a string of the form `'YYYY-MM-DD'` is returned (e.g., `CURDATE()` could return "1998-08-24").

`CURTIME()`/`CURRENT_TIME()`

Returns the current time. A number of the form HHMMSS is returned if this is used in a numerical context, otherwise a string of the form HH:MM:SS is returned (e.g., `CURRENT_TIME()` could return 13:02:43).

`DATABASE()`

Returns the name of the current database (e.g., `DATABASE()` could return "mydata").

`DATE_ADD(`*`date,`*` INTERVAL `*`amount type`*`)`/`ADDDATE(`*`date,`*` INTERVAL `*`amount type`*`)`

Returns a date formed by adding the given amount of time to the given date. The type of time to add can be one of the following: `SECOND`, `MINUTE`, `HOUR`, `DAY`, `MONTH`, `YEAR`, `MINUTE_SECOND` (as "minutes:seconds"), `HOUR_MINUTE` (as "hours:minutes"), `DAY_HOUR` (as "days hours"), `YEAR_MONTH` (as "years-months"), `HOUR_SECOND` (as "hours:minutes:seconds"), `DAY_MINUTE` (as "days hours:minutes") and `DAY_SECOND` (as "days hours:minutes:seconds"). Except for those types with forms specified above, the amount must be an integer value (e.g., `DATE_ADD("1998-08-24 13:00:00", INTERVAL 2 MONTH)` returns "1998-10-24 13:00:00").

`DATE_FORMAT(`*`date, format`*`)`

Returns the date formatted as specified. The format string prints as given with the following values substituted:

%a Short weekday name (Sun, Mon, etc.)

%b Short month name (Jan, Feb, etc.)

%D Day of the month with ordinal suffix (1st, 2nd, 3rd, etc.)

%d Day of the month

%H 24-hour hour (always two digits, e.g., 01)

%h/%I

12-hour hour (always two digits, e.g., 09)

%i	Minutes
%j	Day of the year
%k	24-hour hour (one or two digits, e.g., 1)
%l	12-hour hour (one or two digits, e.g., 9)
%M	Name of the month
%m	Number of the month (January is 1).
%p	AM or PM
%r	12-hour total time (including AM/PM)
%S	Seconds (always two digits, e.g., 04)
%s	Seconds (one or two digits, e.g., 4)
%T	24-hour total time
%U	Week of the year (new weeks begin on Sunday)
%W	Name of the weekday
%w	Number of weekday (0 is Sunday)
%Y	Four digit year
%y	Two digit year
%%	A literal "%" character.

`DATE_SUB(`*`date,`*` INTERVAL `*`amount type`*`)`/`SUBDATE(`*`date,`*` INTERVAL `*`amount type`*`)`

Returns a date formed by subtracting the given amount of time from the given date. The same interval types are used as with `DATE_ADD` (e.g., `SUBDATE("1999-05-20 11:04:23", INTERVAL 2 DAY)` returns "1999-05-18 11:04:23").

`DAYNAME(`*`date`*`)`

Returns the name of the day of the week for the given date (e.g., `DAYNAME('1998-08-22')` returns "Saturday").

`DAYOFMONTH(`*`date`*`)`

Returns the day of the month for the given date (e.g., `DAYOFMONTH('1998-08-22')` returns 22).

`DAYOFWEEK(`*`date`*`)`

Returns the number of the day of the week (1 is Sunday) for the given date (e.g., DAY_OF_WEEK('1998-08-22') returns 7).

`DAYOFYEAR(`*`date`*`)`

Returns the day of the year for the given date (e.g., `DAYOFYEAR('1983-02-15')` returns 46).

`DECODE(`*`blob`*`, `*`passphrase`*`)`

Decodes encrypted binary data using the specified passphrase. The encrypted binary is expected to be one encrypted with the `ENCODE()` function:

```
mysql> SELECT DECODE(ENCODE('open sesame', 'please'), 'please');
+---------------------------------------------------+
| DECODE(ENCODE('open sesame', 'please'), 'please') |
+---------------------------------------------------+
| open sesame                                       |
+---------------------------------------------------+
1 row in set (0.01 sec)
```

`DEGREES(`*`radians`*`)`

Returns the given argument converted from radians to degrees (e.g., `DEGREES(2*PI())` returns 360.000000).

`ELT(`*`number,string1,string2, . . .`*`)`

Returns *string1* if *number* is 1, *string2* if *number* is 2, etc. A null value is returned if *number* does not correspond with a string (e.g., `ELT(3, "once","twice","thrice","fourth")` returns “thrice”).

`ENCODE(`*`secret`*`, `*`passphrase`*`)`

Creates a binary encoding of the *secret* using the *passphrase* as salt. You may later decode the secret using `DECODE()` and the passphrase.

`ENCRYPT(`*`string`*`[, `*`salt`*`])`

Password-encrypts the given string. If a salt is provided, it is used to generate the password (e.g., `ENCRYPT('mypass','3a')` could return “3afi4004idgv”).

`EXP(`*`power`*`)`

Returns the number `e` raised to the given power (e.g., `EXP(1)` returns 2. 718282).

`EXPORT_SET(`*`num`*`, `*`on`*`, `*`off`*`, [`*`separator`*`, [`*`num_bits`*`]])`

Examines a number and maps the on and off bits in that number to the strings specified by the on and off arguments. Examples:

```
mysql> SELECT EXPORT_SET(5, "y", "n", "", 8);
+--------------------------------+
| EXPORT_SET(5, "y", "n", "", 8) |
+--------------------------------+
| ynynnnnn                       |
+--------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT EXPORT_SET(5, "y", "n", ",", 8);
+-------------------------------+
| EXPORT_SET(5, "y", "n", ",", 8) |
+-------------------------------+
| y,n,y,n,n,n,n,n               |
+-------------------------------+
1 row in set (0.00 sec)
```

EXTRACT(*interval* FROM *datetime*)

Returns the specified part of a DATETIME (e.g., EXTRACT(YEAR FROM '2001-08-10 19:45:32') returns 2001).

FIELD(*string,string1,string2, . . .*)

Returns the position in the argument list (starting with *string1*) of the first string that is identical to *string*. Returns 0 if no other string matches *string* (e.g., FIELD('abe','george','john','abe','bill') returns).

FIND_IN_SET(*string,set*)

Returns the position of *string* within *set*. The *set* argument is a series of strings separated by commas (e.g., FIND_IN_SET ('abe', 'george, john, abe, bill') returns 3).

FLOOR(*number*)

Returns the largest integer smaller than or equal to *number* (e.g., FLOOR(5.67) returns 5).

FORMAT(*number, decimals*)

Neatly formats the given number, using the given number of decimals (e.g., FORMAT(4432.99134,2) returns "4,432.99").

FROM_DAYS(*days*)

Returns the date that is the given number of days (where day 1 is the Jan 1 of year 1) (e.g., FROM_DAYS(728749) returns "1995-04-02").

FROM_UNIXTIME(*seconds*[*, format*])

Returns the date (in GMT) corresponding to the given number of seconds since the epoch (January 1, 1970 GMT). If a format string (using the same format as DATE_FORMAT) is given, the returned time is formatted accordingly (e.g., FROM_UNIXTIME(903981584) returns "1998-08-24 18:00:02").

GET_LOCK(*name,seconds*)

Creates a named user-defined lock that waits for the given number of seconds until timeout. This lock can be used for client-side application locking between programs that cooperatively use the same lock names. If the lock is successful, 1 is returned. If the lock times out while waiting, 0 is returned. All others errors return a NULL value. Only one named lock may be active at a time for a singe session. Running GET_LOCK() more than once will silently remove any previous locks (e.g., `GET_LOCK("mylock",10)` could return 1 within the following 10 seconds).

`GREATEST(`*`num1, num2`*`[`*`, num3, . . .`*` ])`
Returns the numerically largest of all of the arguments (e.g., `GREATEST(5,6,68,1,4)` returns 68).

`HEX(`*`decimal`*`)`
Returns the hexadecimal value of the given decimal number. This is equivalent to the function `CONV(decimal,10,16)` (e.g., `HEX(90)` returns “3a”).

`HOUR(`*`time`*`)`
Returns the hour of the given time (e.g., `HOUR('15:33:30')` returns 15).

`IF(`*`test, value1, value2`*`)`
If *test* is true, returns *value1*, otherwise returns *value2*. The *test* value is considered to be an integer, therefore floating point values must be used with comparison operations to generate an integer (e.g., `IF(1>0,"true","false")` returns true).

`IFNULL(`*`value, value2`*`)`
Returns *value* if it is not null, otherwise returns *value2* (e.g., `IFNULL(NULL, "bar")` returns “bar”).

`INSERT(`*`string,position,length,new`*`)`
Returns the string created by replacing the substring of *string* starting at *position* and going *length* characters with *new* (e.g., `INSERT('help',3,1,' can jum')` returns “he can jump”).

`INSTR(`*`string,substring`*`)`
Identical to `LOCATE` except that the arguments are reversed (e.g., `INSTR('makebelieve','lie')` returns 7).

`ISNULL(`*`expression`*`)`
Returns 1 if the expression evaluates to `NULL`, otherwise returns 0 (e.g., `ISNULL(3)` returns 0).

`INTERVAL(`*`A,B,C,D,`*` . . . )`
Returns 0 if *A* is the smallest value, 1 if *A* is between *B* and C, 2 if *A* is between *C* and *D*, etc. All of the values except for *A* must be in order (e.g., `INTERVAL(5,2,4,6,8)` returns 2 (because 5 is in the second interval, between 4 and 6).

`LAST_INSERT_ID()`
Returns the last value that was automatically generated for an `AUTO_INCREMENT` field (e.g., `LAST_INSERT_ID()` could return 4).

`LCASE(`*`string`*`)`/`LOWER(`*`string`*`)`

Returns *string* with all characters turned into lower case (e.g., `LCASE('BoB')` returns “bob”).

`LEAST(`*`num1, num2`*`[`*`, num3,`*` . . .])`

Returns the numerically smallest of all of the arguments (e.g., `LEAST(5,6,68,1,4)` returns 1).

`LEFT(`*`string, length`*`)`

Returns *length* characters from the left end of *string* (e.g., `LEFT("12345",3)` returns “123”).

`LENGTH(`*`string`*`)`/`OCTET_LENGTH(`*`string`*`)`/`CHAR_LENGTH(`*`string`*`)`/`CHARACTER_ LENGTH(`*`string`*`)`

Returns the length of *string* (e.g., `CHAR_LENGTH('Hi Mom!')` returns 7). In character sets that use multibyte characters (such as Unicode, and several Asian character sets), one character may take up more than one byte. In these cases, MySQL’s string functions should correctly count the number of characters, not bytes, in the string. However, in versions prior to 3.23, this did not work properly and the function returned the number of bytes.

`LOAD_FILE(`*`filename`*`)`

Reads the contents of the specified file as a string. This file must exist on the server and be world readable. Naturally, you must also have `FILE` privileges.

`LOCATE(`*`substring,string`*`[`*`,number`*`])`/`POSITION(`*`substring,string`*`)`

Returns the character position of the first occurrence of *substring* within *string*. If *substring* does not exist in *string*, 0 is returned. If a numerical third argument is supplied to `LOCATE`, the search for *substring* within *string* does not start until the given position within *string* (e.g., `LOCATE('SQL','MySQL')` returns 3).

`LOG(`*`number`*`)`

Returns the natural logarithm of *number* (e.g., `LOG(2)` returns 0.693147).

`LOG10(`*`number`*`)`

Returns the common logarithm of *number* (e.g., `LOG10(1000)` returns 3. 000000).

`LPAD(`*`string, length, padding`*`)`

Returns *string* with padding added to the left end until the new string is *length* characters long (e.g., `LPAD(' Merry X-Mas',18,'Ho')` returns “HoHoHo Merry X-Mas”).

`LTRIM(`*`string`*`)`

Returns *string* with all leading whitespace removed (e.g., `LTRIM('    Oops')` returns “Oops”).

`MAKE_SET(`*`bits`*`, `*`string1`*`, `*`string2`*`, ...)`

Creates a MySQL `SET` based on the binary representation of a number by mapping the on bits in the number to string values. Example:

```
mysql> SELECT MAKE_SET(5, "a", "b", "c", "d", "e", "f");
+-----------------------------------------+
| MAKE_SET(5, "a", "b", "c", "d", "e", "f") |
+-----------------------------------------+
| a,c                                     |
+-----------------------------------------+
1 row in set (0.01 sec)
```

`MD5(`*`string`*`)`

Creates an MD5 checksum for the specified string. The MD5 checksum is always a string of 32 hexadecimal numbers.

`MID(`*`string,position, length`*`)`/`SUBSTRING(`*`string,position, length`*`)`/ `SUBSTRING(`*`string`*` FROM `*`position`*` FOR `*`length`*`)`

Returns the substring formed by taking *length* characters from *string*, starting at *position* (e.g., `SUBSTRING('12345',2,3)` returns "234").

`MINUTE(`*`time`*`)`

Returns the minute of the given time (e.g., `MINUTE('15:33:30')` returns 33).

`MOD(`*`num1, num2`*`)`

Returns the modulo of *num1* divided by *num2*. This is the same as the % operator (e.g., `MOD(11,3)` returns 2).

`MONTH(`*`date`*`)`

Returns the number of the month (1 is January) for the given date (e.g., `MONTH('1998-08-22')` returns 8).

`MONTHNAME(`*`date`*`)`

Returns the name of the month for the given date (e.g., `MONTHNAME('1998-08-22')` returns "August").

`NOW()`/`SYSDATE()`/`CURRENT_TIMESTAMP()`

Returns the current date and time. A number of the form YYYYMMDDHHMMSS is returned if this is used in a numerical context, otherwise a string of the form `'YYYY-MM-DD HH:MM:SS'` is returned (e.g., `SYSDATE()` could return "1998-08-24 12:55:32").

`OCT(`*`decimal`*`)`

Returns the octal value of the given decimal number. This is equivalent to the function `CONV(decimal,10,8)` (e.g., `OCT(8)` returns 10).

`PASSWORD(`*`string`*`)`

Returns a password-encrypted version of the given string (e.g., `PASSWD('mypass')` could return "3afi4004idgv").

`PERIOD_ADD(`*`date,months`*`)`
: Returns the date formed by adding the given number of months to *date* (which must be of the form YYMM or YYYYMM) (e.g., `PERIOD_ADD(9808,14)` returns 199910).

`PERIOD_DIFF(`*`date1, date2`*`)`
: Returns the number of months between the two dates (which must be of the form YYMM or YYYYMM) (e.g., `PERIOD_DIFF(199901,8901)` returns 120).

`PI()`
: Returns the value of pi: 3.141593.

`POW(`*`num1, num2`*`)`/`POWER(`*`num1, num2`*`)`
: Returns the value of *num1* raised to the *num2* power (e.g., `POWER(3,2)` returns 9.000000).

`QUARTER(`*`date`*`)`
: Returns the number of the quarter of the given date (1 is January-March) (e.g., `QUARTER('1998-08-22')` returns 3).

`RADIANS(`*`degrees`*`)`
: Returns the given argument converted from degrees to radians (e.g., `RADIANS(-90)` returns -1.570796).

`RAND([`*`seed`*`])`
: Returns a random decimal value between 0 and 1. If an argument is specified, it is used as the seed of the random number generator (e.g., `RAND(3)` could return 0.435434).

`RELEASE_LOCK(`*`name`*`)`
: Removes the named locked created with the `GET_LOCK` function. Returns 1 if the release is successful, 0 if it failed because the current thread did not own the lock and a `NULL` value if the lock did not exist (e.g., `RELEASE_LOCK("mylock")`).

`REPEAT(`*`string,number`*`)`
: Returns a string consisting of the original *string* repeated *number* times. Returns an empty string if *number* is less than or equal to zero (e.g., `REPEAT('ma',4)` returns ‘mamamama’).

`REPLACE(`*`string,old,new`*`)`
: Returns a string that has all occurrences of the substring *old* replaced with *new* (e.g., `REPLACE('`*`black jack`*`','`*`ack`*`','`*`oke`*`')` returns “bloke joke”).

`REVERSE(`*`string`*`)`
: Returns the character reverse of *string* (e.g., `REVERSE('my bologna')` returns “angolob ym”).

`RIGHT(`*`string, length`*`)/SUBSTRING(string FROM length)`

Returns *length* characters from the right end of *string* (e.g., `SUBSTRING("12345" FROM 3)` returns "345").

`ROUND(`*`number`*`[,`*`decimal`*`])`

Returns *number*, rounded to the given number of decimals. If no *decimal* argument is supplied, *number* is rounded to an integer (e.g., `ROUND(5.67,1)` returns 5.7).

`RPAD(`*`string, length, padding`*`)`

Returns *string* with padding added to the right end until the new string is *length* characters long (e.g., `RPAD('Yo',5,'!')` returns "Yo!!!").

`RTRIM(`*`string`*`)`

Returns *string* with all trailing whitespace removed (e.g., `RTRIM('Oops       ')` returns "Oops").

`SECOND(`*`time`*`)`

Returns the seconds of the given time (e.g., `SECOND('15:33:30')` returns 30).

`SEC_TO_TIME(`*`seconds`*`)`

Returns the number of hours, minutes and seconds in the given number of seconds. A number of the form HHMMSS is returned if this is used in a numerical context, otherwise a string of the form HH:MM:SS is returned (e.g., `SEC_TO_TIME(3666)` returns "01:01:06").

`SIGN(`*`number`*`)`

Returns -1 if *number* is negative, 0 if it's zero, or 1 if it's positive (e.g., `SIGN(4)` returns 1).

`SIN(`*`radians`*`)`

Returns the sine of the given number, which is in radians (e.g., `SIN(2*PI())` returns 0.000000).

`SOUNDEX(`*`string`*`)`

Returns the Soundex code associated with string (e.g., `SOUNDEX('Jello')` returns "J400").

`SPACE(`*`number`*`)`

Returns a string that contains *number* spaces (e.g., `SPACE(5)` returns "   ").

`SQRT(`*`number`*`)`

Returns the square root of *number* (e.g., `SQRT(16)` returns 4.000000).

`STRCMP(`*`string1, string2`*`)`

Returns 0 if the strings are the same, -1 if *string1* would sort before than *string2*, or 1 if *string1* would sort after than *string2* (e.g., `STRCMP('bob','bobbie')` returns -1).

`SUBSTRING(`*`string,position`*`)`

Returns all of *string* starting at *position* characters (e.g., `SUBSTRING("123456",3)` returns “3456”).

`SUBSTRING_INDEX(`*`string,character,number`*`)`

Returns the substring formed by counting *number* of *character* within *string* and then returning everything to the right if count is positive, or everything to the left if count is negative (e.g., `SUBSTRING_INDEX('1,2,3,4,5',',',-3)` returns “1,2,3”).

`TAN(`*`radians`*`)`

Returns the tangent of the given number, which must be in radians (e.g., `TAN(0)` returns 0.000000).

`TIME_FORMAT(`*`time, format`*`)`

Returns the given time using a format string. The format string is of the same type as `DATE_FORMAT`, as shown earlier.

`TIME_TO_SEC(`*`time`*`)`

Returns the number of seconds in the *time* argument (e.g., `TIME_TO_SEC('01:01:06')` returns 3666).

`TO_DAYS(`*`date`*`)`

Returns the number of days (where day 1 is the Jan 1 of year 1) to the given date. The date may be a value of type `DATE`, `DATETIME` or `TIMESTAMP`, or a number of the form YYMMDD or YYYYMMDD (e.g., `TO_DAYS(19950402)` returns 728749).

`TRIM([BOTH|LEADING|TRAILING] [`*`remove`*`] [FROM]` *`string`*`)`

With no modifiers, returns *string* with all trailing and leading whitespace removed. You can specify whether to remove either the leading or the trailing whitespace, or both. You can also specify another character other than space to be removed (e.g., `TRIM(both '-' from '---look here---')` returns “look here”).

`TRUNCATE(`*`number, decimals`*`)`

Returns *number* truncated to the given number of decimals (e.g., `TRUNCATE(3.33333333,2)` returns 3.33).

`UCASE(`*`string`*`)`/`UPPER(`*`string`*`)`

Returns *string* with all characters turned into uppercase (e.g., `UPPER('Scooby')` returns “SCOOBY”).

`UNIX_TIMESTAMP([`*`date`*`])`

Returns the number of seconds from the epoch (January 1, 1970 GMT) to the given date (in GMT). If no date is given, the number of seconds to the current date is used (e.g., `UNIX_TIMESTAMP('1998-08-24 18:00:02')` returns 903981584).

`USER()`/`SYSTEM_USER()`/`SESSION_USER()`
: Returns the name of the current user (e.g., `SYSTEM_USER()` could return "ryarger").

`VERSION()`
: Returns the version of the MySQL server itself (e.g., `VERSION()` could return "3.22.5c-alpha").

`WEEK(`*`date`*`)`
: Returns the week of the year for the given date (e.g., `WEEK('1998-12-29')` returns 52).

`WEEKDAY(`*`date`*`)`
: Returns the numeric value of the day of the week for the specified date. Day numbers start with Monday as 0 and end with Sunday as 6.

`YEAR(`*`date`*`)`
: Returns the year of the given date (e.g., `YEAR('1998-12-29')` returns 1998).