

```
-> 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.

## 10.6. Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See [Section 9.3, “Date and Time 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.

Name	Description
<a href="#">ADDDATE ()</a> (v4.1.1)	Add dates
<a href="#">ADDTIME ()</a> (v4.1.1)	Add time
<a href="#">CONVERT_TZ ()</a> (v4.1.3)	Convert from one timezone to another
<a href="#">CURDATE ()</a>	Return the current date
<a href="#">CURRENT_DATE (), CURRENT_DATE</a>	Synonyms for CURDATE()
<a href="#">CURRENT_TIME (), CURRENT_TIME</a>	Synonyms for CURTIME()
<a href="#">CURRENT_TIMESTAMP (), CURRENT_TIMESTAMP</a>	Synonyms for NOW()
<a href="#">CURTIME ()</a>	Return the current time
<a href="#">DATE_ADD ()</a>	Add two dates
<a href="#">DATE_FORMAT ()</a>	Format date as specified
<a href="#">DATE_SUB ()</a>	Subtract two dates
<a href="#">DATE ()</a> (v4.1.1)	Extract the date part of a date or datetime expression
<a href="#">DATEDIFF ()</a> (v4.1.1)	Subtract two dates
<a href="#">DAY ()</a> (v4.1.1)	Synonym for DAYOFMONTH()
<a href="#">DAYNAME ()</a> (v4.1.21)	Return the name of the weekday
<a href="#">DAYOFMONTH ()</a>	Return the day of the month (1-31)
<a href="#">DAYOFWEEK ()</a>	Return the weekday index of the argument
<a href="#">DAYOFYEAR ()</a>	Return the day of the year (1-366)
<a href="#">EXTRACT</a>	Extract part of a date
<a href="#">FROM_DAYS ()</a>	Convert a day number to a date
<a href="#">FROM_UNIXTIME ()</a>	Format date as a UNIX timestamp
<a href="#">GET_FORMAT ()</a> (v4.1.1)	Return a date format string
<a href="#">HOUR ()</a>	Extract the hour

Name	Description
<code>LAST_DAY(v4.1.1)</code>	Return the last day of the month for the argument
<code>LOCALTIME()</code> , <code>LOCALTIME</code>	Synonym for NOW()
<code>LOCALTIMESTAMP</code> , <code>LOCALTIMESTAMP()</code> (v4.0.6)	Synonym for NOW()
<code>MAKEDATE()</code> (v4.1.1)	Create a date from the year and day of year
<code>MAKETIME(v4.1.1)</code>	MAKETIME()
<code>MICROSECOND()</code> (v4.1.1)	Return the microseconds from argument
<code>MINUTE()</code>	Return the minute from the argument
<code>MONTH()</code>	Return the month from the date passed
<code>MONTHNAME()</code> (v4.1.21)	Return the name of the month
<code>NOW()</code>	Return the current date and time
<code>PERIOD_ADD()</code>	Add a period to a year-month
<code>PERIOD_DIFF()</code>	Return the number of months between periods
<code>QUARTER()</code>	Return the quarter from a date argument
<code>SEC_TO_TIME()</code>	Converts seconds to 'HH:MM:SS' format
<code>SECOND()</code>	Return the second (0-59)
<code>STR_TO_DATE()</code> (v4.1.1)	Convert a string to a date
<code>SUBDATE()</code>	When invoked with three arguments a synonym for DATE_SUB()
<code>SUBTIME()</code> (v4.1.1)	Subtract times
<code>SYSDATE()</code>	Return the time at which the function executes
<code>TIME_FORMAT()</code>	Format as time
<code>TIME_TO_SEC()</code>	Return the argument converted to seconds
<code>TIME()</code> (v4.1.1)	Extract the time portion of the expression passed
<code>TIMEDIFF()</code> (v4.1.1)	Subtract time
<code>TIMESTAMP()</code> (v4.1.1)	With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments
<code>TIMESTAMPADD()</code> (v5.0.0)	Add an interval to a datetime expression
<code>TIMESTAMPDIFF()</code> (v5.0.0)	Subtract an interval from a datetime expression
<code>TO_DAYS()</code>	Return the date argument converted to days
<code>UNIX_TIMESTAMP()</code>	Return a UNIX timestamp
<code>UTC_DATE()</code> (v4.1.1)	Return the current UTC date
<code>UTC_TIME()</code> (v4.1.1)	Return the current UTC time
<code>UTC_TIMESTAMP()</code> (v4.1.1)	Return the current UTC date and time
<code>WEEK()</code>	Return the week number
<code>WEEKDAY()</code>	Return the weekday index
<code>WEEKOFYEAR()</code> (v4.1.1)	Return the calendar week of the date (1-53)
<code>YEAR()</code>	Return the year
<code>YEARWEEK()</code>	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;
```

Note that 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()` within a single query always produce the same result (for our purposes a single query also includes a call to a stored routine or trigger and all sub-routines called by that routine/trigger). This principle also applies to `CURDATE()`, `CURTIME()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`, and to any of their synonyms.

The `CURRENT_TIMESTAMP()`, `CURRENT_TIME()`, `CURRENT_DATE()`, and `FROM_UNIXTIME()` functions return values in the connection's current time zone, which is available as the value of the `time_zone` system variable. In addition, `UNIX_TIMESTAMP()` assumes that its argument is a datetime value in the current time zone. See [Section 8.9, “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. 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
```

- `ADDDATE(date, INTERVAL expr unit), ADDDATE(expr, days)`

When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` is a synonym for `DATE_ADD()`. The related function `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL unit` argument, see the discussion for `DATE_ADD()`.

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
       -> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
       -> '1998-02-02'
```

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('1998-01-02', 31);
       -> '1998-02-02'
```

- `ADDTIME(expr1, expr2)`

`ADDTIME()` adds `expr2` to `expr1` and returns the result. `expr1` is a time or datetime expression, and `expr2` is a time expression.

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999',
       -> '1 1:1:1.000002');
       -> '1998-01-02 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)`

`CONVERT_TZ()` 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 8.9, “MySQL Server Time Zone Support”](#). This function returns `NULL` if the arguments are invalid.

If the value falls out of the supported range of the `TIMESTAMP` type when converted from `from_tz` to UTC, no conversion occurs. The `TIMESTAMP` range is described in [Section 9.1.2, “Overview of Date and Time Types”](#).

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00', 'GMT', 'MET');
       -> '2004-01-01 13:00:00'
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00', '+00:00', '+10:00');
       -> '2004-01-01 22:00:00'
```

## Note

To use named time zones such as `'MET'` or `'Europe/Moscow'`, the time zone tables must be properly set up. See [Section 8.9, “MySQL Server Time Zone Support”](#), for instructions.

- `CURDATE()`

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

```
mysql> SELECT CURDATE();
-> '1997-12-15'
mysql> SELECT CURDATE() + 0;
-> 19971215
```

- CURRENT\_DATE, CURRENT\_DATE()

CURRENT\_DATE and CURRENT\_DATE() are synonyms for CURDATE().

- CURTIME()

Returns the current time as a value in 'HH:MM:SS' or HHMMSS.aaaaaa format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();
-> '23:50:26'
mysql> SELECT CURTIME() + 0;
-> 235026.000000
```

- CURRENT\_TIME, CURRENT\_TIME()

CURRENT\_TIME and CURRENT\_TIME() are synonyms for CURTIME().

- CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP()

CURRENT\_TIMESTAMP and CURRENT\_TIMESTAMP() are synonyms for NOW().

- DATE(expr)

Extracts the date part of the date or datetime expression *expr*.

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

- DATEDIFF(expr1, expr2)

DATEDIFF() 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('1997-12-31 23:59:59', '1997-12-30');
-> 1
mysql> SELECT DATEDIFF('1997-11-30 23:59:59', '1997-12-31');
-> -31
```

- DATE\_ADD(date, INTERVAL *expr* *unit*), DATE\_SUB(date, INTERVAL *expr* *unit*)

These functions perform date arithmetic. *date* is a DATETIME or DATE value specifying the starting date. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a “-” for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted.

The INTERVAL keyword and the *unit* specifier are not case sensitive.

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

<i>unit</i> Value	Expected <i>expr</i> Format
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOURL	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS

SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOURL_MICROSECOND	'HOURS.MICROSECONDS'
HOURL_SECOND	'HOURS:MINUTES:SECONDS'
HOURL_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

MySQL allows any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a *DATE* value and your calculations involve only *YEAR*, *MONTH*, and *DAY* parts (that is, no time parts), the result is a *DATE* value. Otherwise, the result is a *DATETIME* value.

Date arithmetic also can be performed using *INTERVAL* together with the *+* or *-* operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

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

```
mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
-> '1998-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
-> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
-> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 SECOND);
-> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 DAY);
-> '1998-01-01 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL '1:1' MINUTE_SECOND);
-> '1998-01-01 00:01:00'
mysql> SELECT DATE_SUB('1998-01-01 00:00:00',
-> INTERVAL '1 1:1:1' DAY_SECOND);
-> '1997-12-30 22:58:59'
mysql> SELECT DATE_ADD('1998-01-01 00:00:00',
-> INTERVAL '-1 10' DAY_HOUR);
-> '1997-12-30 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'
```

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

Because *expr* is treated as a string, be careful if you specify a non-string value with *INTERVAL*. For example, with an interval specifier of *HOURL\_MINUTE*, *6/4* evaluates to *1.5000* and is treated as 1 hour, 5000 minutes:

```
mysql> SELECT 6/4;
-> 1.5000
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 6/4 HOURL_MINUTE);
-> '1999-01-04 12:20:00'
```

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

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

```
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
-> INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR_MINUTE);
-> '1970-01-01 13:05:00'
```

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

```
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
-> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
-> '1999-01-01 01:00:00'
```

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

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
-> '1998-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as `'2006-07-00'` or badly malformed dates:

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);
-> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
-> NULL
```

- `DATE_FORMAT(date, format)`

Formats the *date* value according to the *format* string.

The following specifiers may be used in the *format* string. The “%” character is required before format specifier characters.

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
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; 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 allows the storing of incomplete dates such as '2004-00-00'.

The language used for day and month names and abbreviations is controlled by the value of the `lc_time_names` system variable (Section 8.10, “MySQL Server Locale Support”).

The `DATE_FORMAT()` 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('1997-10-04 22:23:00', '%W %M %Y');
-> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
      '%D %y %a %d %m %b %j');
-> '4th 97 Sat 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)`

See the description for `DATE_ADD()`.

- `DAY(date)`

`DAY()` is a synonym for `DAYOFMONTH()`.

- `DAYNAME(date)`

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 (Section 8.10, “MySQL Server Locale Support”).

```
mysql> SELECT DAYNAME('1998-02-05');
-> 'Thursday'
```

- `DAYOFMONTH(date)`

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.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
-> 3
```

- `DAYOFWEEK(date)`

Returns the weekday index for `date` (1 = Sunday, 2 = Monday, ..., 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
-> 3
```

- `DAYOFYEAR(date)`

Returns the day of the year for `date`, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
```

-> 34

- `EXTRACT(unit FROM date)`

The `EXTRACT()` function uses the same kinds of unit specifiers as `DATE_ADD()` or `DATE_SUB()`, but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
-> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '1999-07-02 01:02:03');
-> 20102
mysql> SELECT EXTRACT(MICROSECOND
-> FROM '2003-01-02 10:30:00.000123');
-> 123
```

- `FROM_DAYS(N)`

Given a day number *N*, returns a `DATE` value.

```
mysql> SELECT FROM_DAYS(729669);
-> '1997-10-07'
```

Use `FROM_DAYS()` with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See [Section 10.7, “What Calendar Is Used By MySQL?”](#).

- `FROM_UNIXTIME(unix_timestamp)`, `FROM_UNIXTIME(unix_timestamp, format)`

Returns a representation of the `unix_timestamp` argument as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDH-HMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone. `unix_timestamp` is an internal timestamp value such as is produced by the `UNIX_TIMESTAMP()` function.

If `format` is given, the result is formatted according to the `format` string, which is used the same way as listed in the entry for the `DATE_FORMAT()` function.

```
mysql> SELECT FROM_UNIXTIME(1196440219);
-> '2007-11-30 10:30:19'
mysql> SELECT FROM_UNIXTIME(1196440219) + 0;
-> 20071130103019.000000
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
-> '%Y %D %M %h:%i:%s %x');
-> '2007 30th November 10:30:59 2007'
```

Note: If you use `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` to convert between `TIMESTAMP` values 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()` function.

- `GET_FORMAT(DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')`

Returns a format string. This function is useful in combination with the `DATE_FORMAT()` and the `STR_TO_DATE()` functions.

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()` function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
<code>GET_FORMAT(DATE, 'USA')</code>	<code>'%m.%d.%Y'</code>
<code>GET_FORMAT(DATE, 'JIS')</code>	<code>'%Y-%m-%d'</code>
<code>GET_FORMAT(DATE, 'ISO')</code>	<code>'%Y-%m-%d'</code>
<code>GET_FORMAT(DATE, 'EUR')</code>	<code>'%d.%m.%Y'</code>
<code>GET_FORMAT(DATE, 'INTERNAL')</code>	<code>'%Y%m%d'</code>
<code>GET_FORMAT(DATETIME, 'USA')</code>	<code>'%Y-%m-%d %H.%i.%s'</code>
<code>GET_FORMAT(DATETIME, 'JIS')</code>	<code>'%Y-%m-%d %H:%i:%s'</code>
<code>GET_FORMAT(DATETIME, 'ISO')</code>	<code>'%Y-%m-%d %H:%i:%s'</code>
<code>GET_FORMAT(DATETIME, 'EUR')</code>	<code>'%Y-%m-%d %H.%i.%s'</code>
<code>GET_FORMAT(DATETIME, 'INTERNAL')</code>	<code>'%Y%m%d%H%i%s'</code>



GET_FORMAT (TIME, 'USA')	'%h:%i:%s %p'
GET_FORMAT (TIME, 'JIS')	'%H:%i:%s'
GET_FORMAT (TIME, 'ISO')	'%H:%i:%s'
GET_FORMAT (TIME, 'EUR')	'%H.%i.%s'
GET_FORMAT (TIME, 'INTERNAL')	'%H%i%s'

**TIMESTAMP** can also be used as the first argument to **GET\_FORMAT()**, 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*)**

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.

```
mysql> SELECT HOUR('10:05:03');
-> 10
mysql> SELECT HOUR('272:59:59');
-> 272
```

- **last\_day(*date*)**

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.

```
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, localtime()**

**localtime** and **localtime()** are synonyms for **now()**.

- **localtimestamp, localtimestamp()**

**localtimestamp** and **localtimestamp()** are synonyms for **now()**.

- **make\_date(*year*, *dayofyear*)**

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

```
mysql> SELECT MAKE_DATE(2001, 31), MAKE_DATE(2001, 32);
-> '2001-01-31', '2001-02-01'
mysql> SELECT MAKE_DATE(2001, 365), MAKE_DATE(2004, 365);
-> '2001-12-31', '2004-12-30'
mysql> SELECT MAKE_DATE(2001, 0);
-> NULL
```

- **make\_time(*hour*, *minute*, *second*)**

Returns a time value calculated from the *hour*, *minute*, and *second* arguments.

```
mysql> SELECT MAKE_TIME(12, 15, 30);
-> '12:15:30'
```

- **microsecond(*expr*)**

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

```
mysql> SELECT MICROSECOND('12:00:00.123456');
-> 123456
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');
```

```
-> 10
```

- `MINUTE(time)`

Returns the minute for *time*, in the range 0 to 59.

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
-> 5
```

- `MONTH(date)`

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.

```
mysql> SELECT MONTH('1998-02-03');
-> 2
```

- `MONTHNAME(date)`

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 8.10, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('1998-02-05');
-> 'February'
```

- `NOW()`

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT NOW();
-> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
-> 20071215235026.000000
```

`NOW()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, `NOW()` returns the time at which the routine or triggering statement began to execute.) This differs from the behavior for `SYSDATE()`, 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 |
+-----+-----+-----+
```

See the description for `SYSDATE()` for additional information about the differences between the two functions.

- `PERIOD_ADD(P, N)`

Adds *N* months to period *P* (in the format `YYMM` or `YYYYMM`). Returns a value in the format `YYYYMM`. Note that the period argument *P* is *not* a date value.

```
mysql> SELECT PERIOD_ADD(9801, 2);
-> 199803
```

- `PERIOD_DIFF(P1, P2)`

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.

```
mysql> SELECT PERIOD_DIFF(9802, 199703);
-> 11
```

- `QUARTER(date)`

Returns the quarter of the year for *date*, in the range 1 to 4.

```
mysql> SELECT QUARTER('98-04-01');
-> 2
```

- `SECOND(time)`

Returns the second for *time*, in the range 0 to 59.

```
mysql> SELECT SECOND('10:05:03');
-> 3
```

- `SEC_TO_TIME(seconds)`

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.

```
mysql> SELECT SEC_TO_TIME(2378);
-> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
-> 3938
```

- `STR_TO_DATE(str, format)`

This is the inverse of the `DATE_FORMAT()` 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.

The date, time, or datetime values contained in *str* should be given in the format indicated by *format*. For the specifiers that can be used in *format*, see the `DATE_FORMAT()` function description. If *str* contains an illegal date, time, or datetime value, `STR_TO_DATE()` returns `NULL`. An illegal value also produces a warning.

Range checking on the parts of date values is as described in [Section 9.3.1, “The DATETIME, DATE, and TIMESTAMP Types”](#). This means, for example, that “zero” dates or dates with part values of 0 are allowed 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'
```

### 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, then you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
-> '2004-10-18'
```

- `SUBDATE(date, INTERVAL expr unit), SUBDATE(expr, days)`

When invoked with the `INTERVAL` form of the second argument, `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL unit` argument, see the discussion for `DATE_ADD()`.

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
```

The second form allows 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('1998-01-02 12:00:00', 31);
-> '1997-12-02 12:00:00'
```

- `SUBTIME(expr1, expr2)`

`SUBTIME()` 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.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999', '1 1:1:1.000002');
-> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
-> '-00:59:59.999999'
```

- `SYSDATE()`

Returns the current date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context.

`SYSDATE()` returns the time at which it executes. This differs from the behavior for `NOW()`, which returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, `NOW()` returns the time at which the routine 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()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`.

Because `SYSDATE()` can return different values even within the same statement, and is not affected by `SET TIMESTAMP`, it is non-deterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging, or start the server with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`. The non-deterministic nature of `SYSDATE()` also means that indexes cannot be used for evaluating expressions that refer to it.

- `TIME(expr)`

Extracts the time part of the time or datetime expression `expr` and returns it as a string.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
-> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
-> '01:02:03.000123'
```

- `TIMEDIFF(expr1, expr2)`

`TIMEDIFF()` returns `expr1 - expr2` expressed as a time value. `expr1` and `expr2` are time or date-and-time expressions, but both must be of the same type.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
-> '2000:01:01 00:00:00.000001');
-> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
-> '1997-12-30 01:01:01.000002');
-> '46:58:57.999999'
```

- `TIMESTAMP(expr)`, `TIMESTAMP(expr1, 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.

```
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)`

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: `FRAC_SECOND` (microseconds), `SECOND`, `MINUTE`, `HOURL`, `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.

```
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'
```

- `TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)`

Returns the integer difference between the date or datetime expressions `datetime_expr1` and `datetime_expr2`. The unit for the result is given by the `unit` argument. The legal values for `unit` are the same as those listed in the description of the `TIMESTAMPADD()` function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
-> -1
```

- `TIME_FORMAT(time,format)`

This is used like the `DATE_FORMAT()` function, but the `format` string may contain format specifiers only for hours, minutes, and seconds. Other specifiers produce a `NULL` value or 0.

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)`

Returns the `time` argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
-> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
-> 2378
```

- `TO_DAYS(date)`

Given a date `date`, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('1997-10-07');
-> 729669
```

`TO_DAYS()` 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 10.7, “What Calendar Is Used By MySQL?”](#), for details.

Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in [Section 9.3, “Date and Time Types”](#). For example, `'1997-10-07'` and `'97-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');
-> 729669, 729669
```

- `UNIX_TIMESTAMP(), UNIX_TIMESTAMP(date)`

If called with no argument, returns a Unix timestamp (seconds since `'1970-01-01 00:00:00'` UTC) as an unsigned integer. If `UNIX_TIMESTAMP()` is called with a `date` argument, it returns the value of the argument as seconds since `'1970-01-01 00:00:00'` UTC. `date` may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format `YYMMDD` or `YYYYMMDD`. The server interprets `date` as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in [Section 8.9, “MySQL Server Time Zone Support”](#).

```
mysql> SELECT UNIX_TIMESTAMP();
-> 1196440210
mysql> SELECT UNIX_TIMESTAMP('2007-11-30 10:30:19');
-> 1196440219
```

When `UNIX_TIMESTAMP` is used on a `TIMESTAMP` column, the function returns the internal timestamp value directly, with no implicit “string-to-Unix-timestamp” conversion. If you pass an out-of-range date to `UNIX_TIMESTAMP()`, it returns 0.

Note: If you use `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` to convert between `TIMESTAMP` values 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, it is possible for two `UNIX_TIMESTAMP()` to map two `TIMESTAMP` values to the same Unix timestamp value. `FROM_UNIXTIME()` will map that value back to only one of the original `TIMESTAMP` values. Here is an example, using `TIMESTAMP` values in the `CET` time zone:

```
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 |
+-----+
```

If you want to subtract `UNIX_TIMESTAMP()` columns, you might want to cast the result to signed integers. See [Section 10.9, “Cast Functions and Operators”](#).

- `UTC_DATE, UTC_DATE()`

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

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814
```

- `UTC_TIME, UTC_TIME()`

Returns the current UTC time as a value in `'HH:MM:SS'` or `HHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
-> '18:07:53', 180753.000000
```

- `UTC_TIMESTAMP, UTC_TIMESTAMP()`

Returns the current UTC date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
-> '2003-08-14 18:08:04', 20030814180804.000000
```

- `WEEK(date[, mode])`

This function returns the week number for `date`. The two-argument form of `WEEK()` allows 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 5.1.3, “System Variables”](#).

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

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

7	Monday	1-53	with a Monday in this year
---	--------	------	----------------------------

```
mysql> SELECT WEEK('1998-02-20');
-> 7
mysql> SELECT WEEK('1998-02-20',0);
-> 7
mysql> SELECT WEEK('1998-02-20',1);
-> 8
mysql> SELECT WEEK('1998-12-31',1);
-> 53
```

Note that 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 MySQL should return 52 for the `WEEK()` function, because the given date actually occurs in the 52nd week of 1999. We decided to return 0 instead because we want the function to return “the week number in the given year.” This makes use of the `WEEK()` function reliable when combined with other functions that extract a date part from a date.

If you would prefer the result to be 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()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');
-> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
-> '52'
```

- `WEEKDAY(date)`

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

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
-> 1
mysql> SELECT WEEKDAY('1997-11-05');
-> 2
```

- `WEEKOFYEAR(date)`

Returns the calendar week of the date as a number in the range from 1 to 53. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(date,3)`.

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
-> 8
```

- `YEAR(date)`

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

```
mysql> SELECT YEAR('98-02-03');
-> 1998
```

- `YEARWEEK(date)`, `YEARWEEK(date,mode)`

Returns year and week for a date. The *mode* argument works exactly like the *mode* argument to `WEEK()`. 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.

```
mysql> SELECT YEARWEEK('1987-01-01');
-> 198653
```

Note that the week number is different from what the `WEEK()` function would return (0) for optional arguments 0 or 1, as `WEEK()` then returns the week in the context of the given year.