12.7 Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 11.3, “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.

Table 12.13 Date/Time Functions

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CONVERT_TZ()	Convert from one time zone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date

Name	Description
GET_FORMAT()	Return a date format string
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression

Name	Description
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	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() 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(), 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 10.6, “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 more strict when passed a `DATE()` function value as their argument and reject incomplete dates with a day part of zero. These functions are affected: `CONVERT_TZ()`, `DATE_ADD()`, `DATE_SUB()`, `DAYOFYEAR()`, `LAST_DAY()` (permits a day part of zero), `TIMESTAMPDIFF()`, `TO_DAYS()`, `TO_SECONDS()`, `WEEK()`, `WEEKDAY()`, `WEEKOFYEAR()`, `YEARWEEK()`.

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)`, `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('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'
  ```

- `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('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)`

  `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 10.6, "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 11.1.2, "Date and Time Type Overview".

```
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 10.6, "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();
        -> '2008-06-13'
mysql> SELECT CURDATE() + 0;
        -> 20080613
```

- `CURRENT_DATE`, `CURRENT_DATE()`

  `CURRENT_DATE` and `CURRENT_DATE()` are synonyms for `CURDATE()`.

- `CURRENT_TIME`, `CURRENT_TIME([fsp])`

  `CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP([fsp])`

  `CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()`.

- `CURTIME([fsp])`

  Returns the current time as a value in `'HH:MM:SS'` or `HHMMSS` format, depending on whether the function is

used in a string or numeric context. The value is expressed in the current 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();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026.000000
```

- `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('2007-12-31 23:59:59','2007-12-30');
        -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
        -> -31
```

- `DATE_ADD(date,INTERVAL expr unit)` `DATE_SUB(date,INTERVAL expr unit)`

  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 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.

`unit` Value	Expected `expr` Format
`MICROSECOND`	`MICROSECONDS`
`SECOND`	`SECONDS`
`MINUTE`	`MINUTES`
`HOUR`	`HOURS`
`DAY`	`DAYS`

unit Value	Expected expr Format
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

The return value depends on the arguments:

- 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.
- String otherwise.

To ensure that the result is DATETIME, you can use CAST() to convert the first argument to DATETIME.

MySQL permits 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 permitted on either side of the + operator if the expression on the other side is a

INTERVAL expr
date or datetime value. For the - operator, unit is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '2009-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '2008-12-31';
        -> '2009-01-01'
mysql> SELECT '2005-01-01' - INTERVAL 1 SECOND;
        -> '2004-12-31 23:59:59'
mysql> SELECT DATE_ADD('2000-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2001-01-01 00:00:00'
mysql> SELECT DATE_ADD('2010-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2011-01-01 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('2005-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2004-12-30 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'
```

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 nonstring value with INTERVAL. For example, with an interval specifier of HOUR_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('2009-01-01', INTERVAL 6/4 HOUR_MINUTE);
        -> '2009-01-04 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));
        -> 1.5
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
    ->                 INTERVAL CAST(6/4 AS DECIMAL(3,1))
HOUR_MINUTE);
        -> '1970-01-01 13:05:00'
```

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

```
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 DAY);
        -> '2013-01-02'
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 HOUR);
        -> '2013-01-01 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('2009-01-30', INTERVAL 1 MONTH);
        -> '2009-02-28'
```

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`)

Specifier	Description
%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() mode 0
%u	Week (00..53), where Monday is the first day of the week; WEEK() mode 1
%V	Week (01..53), where Sunday is the first day of the week; WEEK() mode 2; used with %X
%v	Week (01..53), where Monday is the first day of the week; WEEK() 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 10.7, “MySQL Server Locale Support”).

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

`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('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)`

  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 10.7, “MySQL Server Locale Support”).

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

- `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('2007-02-03');
        -> 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('2007-02-03');
        -> 7
```

- `DAYOFYEAR(date)`

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

```
mysql> SELECT DAYOFYEAR('2007-02-03');
        -> 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 '2009-07-02');
       -> 2009
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
       -> 200907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 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(730669);
        -> '2007-07-03'
```

  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 12.8, “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 `YYYYMMDDHHMMSS` 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(1447430881);
        -> '2015-11-13 10:08:01'
mysql> SELECT FROM_UNIXTIME(1447430881) + 0;
        -> 20151113100801
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
    ->                      '%Y %D %M %h:%i:%s
%x');
        -> '2015 13th November 10:08:01 2015'
```

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
`GET_FORMAT(DATE,'USA')`	`'%m.%d.%Y'`
`GET_FORMAT(DATE,'JIS')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'ISO')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'EUR')`	`'%d.%m.%Y'`
`GET_FORMAT(DATE,'INTERNAL')`	`'%Y%m%d'`
`GET_FORMAT(DATETIME,'USA')`	`'%Y-%m-%d %H.%i.%s'`
`GET_FORMAT(DATETIME,'JIS')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'ISO')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'EUR')`	`'%Y-%m-%d %H.%i.%s'`
`GET_FORMAT(DATETIME,'INTERNAL')`	`'%Y%m%d%H%i%s'`
`GET_FORMAT(TIME,'USA')`	`'%h:%i:%s %p'`
`GET_FORMAT(TIME,'JIS')`	`'%H:%i:%s'`

Function Call	Result
`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([fsp])`

  `LOCALTIME` and `LOCALTIME()` are synonyms for `NOW()`.

- `LOCALTIMESTAMP`, `LOCALTIMESTAMP([fsp])`

  `LOCALTIMESTAMP` and `LOCALTIMESTAMP()` are synonyms for `NOW()`.

- `MAKEDATE(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 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)`

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

  The `second` argument can have a fractional part.

```
mysql> SELECT MAKETIME(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('2009-12-31
23:59:59.000010');
        -> 10
```

- `MINUTE(time)`

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

```
mysql> SELECT MINUTE('2008-02-03 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('2008-02-03');
        -> 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 10.7, “MySQL Server Locale Support”).

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

- `NOW([fsp])`

  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 a string or numeric context. The value is expressed in the current 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()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function 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
|
+---------------------+----------+---------------------+
```

  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()`. Setting the timestamp to a nonzero value causes each subsequent invocation of `NOW()` to return that value. Setting the timestamp to zero cancels this effect so that `NOW()` once again returns the current date and time.

  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(200801,2);
        -> 200803
```

- `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(200802,200703);
        -> 11
```

- `QUARTER(date)`

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

```
mysql> SELECT QUARTER('2008-04-01');
        -> 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. If the date, time, or datetime value extracted from `str` is illegal, `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()` 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 11.3.1, “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` or `NO_ZERO_IN_DATE` SQL mode is enabled, zero dates or part of dates are disallowed. In that case, `STR_TO_DATE()` returns `NULL` and generates a warning:

```
mysql> SET sql_mode = '';
mysql> SELECT STR_TO_DATE('15:35:00', '%H:%i:%s');
+-----------------------------------+
| STR_TO_DATE('15:35:00', '%H:%i:%s') |
+-----------------------------------+
| 15:35:00                          |
+-----------------------------------+
mysql> SET sql_mode = 'NO_ZERO_IN_DATE';
mysql> SELECT STR_TO_DATE('15:35:00', '%h:%i:%s');
+-----------------------------------+
| STR_TO_DATE('15:35:00', '%h:%i:%s') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1411
Message: Incorrect datetime value: '15:35:00' 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'
```

- `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('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'
```

- `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('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'
```

- `SYSDATE([fsp])`

  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 a 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()` 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 function or trigger, `NOW()` 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()` 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 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()` to be an alias for `NOW()`. This works if the option is used on both the master and the slave.

The nondeterministic 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.

  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');
        -> '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.

  The result returned by `TIMEDIFF()` is limited to the range allowed for `TIME` values. Alternatively, you can use either of the functions `TIMESTAMPDIFF()` and `UNIX_TIMESTAMP()`, 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)`, `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: `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.

```
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 `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()` function.

  ```
  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()` function when invoked with 2 arguments.

- `TIME_FORMAT(time,format)`

  This is used like the `DATE_FORMAT()` function, but the `format` string may contain format specifiers only for hours, minutes, seconds, and microseconds. 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('2007-10-07');
        -> 733321
```

`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 12.8, “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 11.3, “Date and Time 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()` 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)`

  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, 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()`, `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 12.8, “What Calendar Is Used By MySQL?”, for details.

  Like `TO_DAYS()`, `TO_SECONDS()`, converts two-digit year values in dates to four-digit form using the rules in Section 11.3, “Date and Time 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()` 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(), UNIX_TIMESTAMP(date)

  If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC). 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.

  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. 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 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 10.6, “MySQL Server Time Zone Support”.

```
mysql> SELECT UNIX_TIMESTAMP();
        -> 1447431666
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19');
        -> 1447431619
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19.012');
        -> 1447431619.012
```

  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. The valid range of 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.

  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 12.10, “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([fsp])`

  Returns the current UTC time as a value in `'HH:MM:SS'` or `HHMMSS` format, depending on whether the function is used in a 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`, `UTC_TIMESTAMP([fsp])`

  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 a 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])`

  This function returns the week number for `date`. The two-argument form of `WEEK()` 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 5.1.5, “Server System Variables”.

  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 WEEK('2008-02-20');
        -> 7
mysql> SELECT WEEK('2008-02-20',0);
        -> 7
mysql> SELECT WEEK('2008-02-20',1);
        -> 8
mysql> SELECT WEEK('2008-12-31',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 `WEEK()` should return `52` because the given date actually occurs in the 52nd week of 1999. `WEEK()` returns `0` instead so that the return value is "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 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()` 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('2008-02-03 22:23:00');
        -> 6
mysql> SELECT WEEKDAY('2007-11-06');
        -> 1
```

- `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('2008-02-20');
        -> 8
```

- `YEAR(date)`

  Returns the year for `date`, in the range `1000` to `9999`, or `0` for the "zero" date.

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

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

  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.

  The `mode` argument works exactly like the `mode` argument to `WEEK()`. For the single-argument syntax, a `mode` value of 0 is used. Unlike `WEEK()`, the value of `default_week_format` does not influence

`YEARWEEK()`.

```
mysql> SELECT YEARWEEK('1987-01-01');
        -> 198652
```

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.