12.7 Date and Time Functions

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

Table 12.11 Date and Time Functions

Name	**Description**
`ADDDATE()`	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
`GET_FORMAT()`	Return a date format string

Name	Description
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
TIMESTAMPDIFF()	Return the difference of two datetime expressions, using the units specified
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

Name	Description
`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 current session time zone, which is available as the session value of the `time_zone` system variable. In addition, `UNIX_TIMESTAMP()` assumes that its argument is a datetime value in the session time zone. See Section 5.1.15, “MySQL Server Time Zone Support”.

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

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
        -> 0, 0
```

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

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
        -> NULL
mysql> SELECT DAYNAME('2006-05-00');
        -> NULL
```

Several functions are strict when passed a `DATE()` function value as their argument and reject incomplete dates with a day part of zero: `CONVERT_TZ()`, `DATE_ADD()`, `DATE_SUB()`, `DAYOFYEAR()`, `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(`***`date`***`,`***`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 Temporal Intervals.

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

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

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

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

- `ADDTIME(`***`expr1`***`,`***`expr2`***`)`

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

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

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

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

```
mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
        -> '2008-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
        -> '03:00:01.999997'
```

- `CONVERT_TZ(`***`dt`***`,`***`from_tz`***`,`***`to_tz`***`)`

  `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 5.1.15, “MySQL Server Time Zone Support”. This function returns `NULL` if any of the arguments are invalid, or if any of them are `NULL`.

  On 32-bit platforms, the supported range of values for this function is the same as for the `TIMESTAMP` type (see Section 11.2.1, “Date and Time Data Type Syntax”, for range information). On 64-bit platforms, beginning with MySQL 8.0.28, the maximum supported value is `'3001-01-18 23:59:59.999999'` UTC.

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

```
mysql> SELECT CONVERT_TZ('2004-01-01 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/Amsterdam'`, the time zone tables must be properly set up. For instructions, see Section 5.1.15, “MySQL Server Time Zone Support”.

- `CURDATE()`

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

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

- `CURRENT_DATE`, `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 string or numeric context. The value is expressed in the session time zone.

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

```
mysql> SELECT CURTIME();
+-----------+
| CURTIME() |
+-----------+
| 19:25:37  |
+-----------+

mysql> SELECT CURTIME() + 0;
+---------------+
| CURTIME() + 0 |
+---------------+
|        192537 |
+---------------+

mysql> SELECT CURTIME(3);
+--------------+
| CURTIME(3)   |
+--------------+
| 19:25:37.840 |
+--------------+
```

- `DATE(`***`expr`***`)`

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

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

- `DATEDIFF(`***expr1***`,`***expr2***`)`

  `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
  ```

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

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

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

  The return value depends on the arguments:

  - If ***date*** is `NULL`, the function returns `NULL`.
  - `DATE` if the ***date*** argument is a `DATE` value and your calculations involve only `YEAR`, `MONTH`, and `DAY` parts (that is, no time parts).
  - (*MySQL 8.0.28 and later*:) `TIME` if the ***date*** argument is a `TIME` value and the calculations involve only `HOURS`, `MINUTES`, and `SECONDS` parts (that is, no date parts).
  - `DATETIME` if the first argument is a `DATETIME` (or `TIMESTAMP`) value, or if the first argument is a `DATE` and the ***unit*** value uses `HOURS`, `MINUTES`, or `SECONDS`, or if the first argument is of type `TIME` and the ***unit*** value uses `YEAR`, `MONTH`, or `DAY`.

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

- String otherwise (type `VARCHAR`).

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

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

```
mysql> SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
        -> '2018-05-02'
mysql> SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
        -> '2017-05-01'
mysql> SELECT DATE_ADD('2020-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2021-01-01 00:00:00'
mysql> SELECT DATE_ADD('2018-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2019-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2025-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2024-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

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

```
mysql> SELECT DATE_ADD('2024-03-30', INTERVAL 1 MONTH) AS d1,
     >        DATE_ADD('2024-03-31', INTERVAL 1 MONTH) AS d2;
+------------+------------+
| d1         | d2         |
+------------+------------+
| 2024-04-30 | 2024-04-30 |
+------------+------------+
1 row in set (0.00 sec)
```

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

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

  The specifiers shown in the following table may be used in the ***`format`*** string. The `%` character is required before format specifier characters. The specifiers apply to other functions as well: `STR_TO_DATE()`, `TIME_FORMAT()`, `UNIX_TIMESTAMP()`.

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`)

Specifier	Description
`%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 10.16, "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 (see Section 10.16, “MySQL Server Locale Support”). Returns `NULL` if ***`date`*** is `NULL`.

  ```
  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. Returns `NULL` if ***`date`*** is `NULL`.

  ```
  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. Returns `NULL` if ***`date`*** is `NULL`.

  ```
  mysql> SELECT DAYOFWEEK('2007-02-03');
          -> 7
  ```

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

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

```
mysql> SELECT DAYOFYEAR('2007-02-03');
        -> 34
```

- `EXTRACT(`***`unit`***` FROM `***`date`***`)`

  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. For information on the ***`unit`*** argument, see Temporal Intervals. Returns `NULL` if ***`date`*** is `NULL`.

```
mysql> SELECT EXTRACT(YEAR FROM '2019-07-02');
        -> 2019
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2019-07-02 01:02:03');
        -> 201907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2019-07-02 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. Returns `NULL` if ***`N`*** is `NULL`.

```
mysql> SELECT FROM_DAYS(730669);
        -> '2000-07-03'
```

  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 11.2.7, “What Calendar Is Used By MySQL?”.

- `FROM_UNIXTIME(`***`unix_timestamp`***`[,`***`format`***`])`

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

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

  If ***`unix_timestamp`*** or ***`format`*** is `NULL`, this function returns `NULL`.

If ***unix_timestamp*** is an integer, the fractional seconds precision of the `DATETIME` is zero. When ***unix_timestamp*** is a decimal value, the fractional seconds precision of the `DATETIME` is the same as the precision of the decimal value, up to a maximum of 6. When ***unix_timestamp*** is a floating point number, the fractional seconds precision of the datetime is 6.

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

format is used to format the result in the same way as the format string used for the `DATE_FORMAT()` function. If ***format*** is supplied, the value returned is a `VARCHAR`.

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

> **Note**
>
> If you use `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` to convert between values in a non-UTC time zone and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the `UNIX_TIMESTAMP()` 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.

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

The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the `DATE_FORMAT()` function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
`GET_FORMAT(DATE,'USA')`	`'%m.%d.%Y'`

Function Call	**Result**
`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'`
`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`. Returns `NULL` if ***time*** is `NULL`.

```
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 or `NULL`.

```
mysql> SELECT LAST_DAY('2003-02-05');
        -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
        -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
        -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
        -> NULL
```

- `LOCALTIME`, `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`. The result is also `NULL` if either argument is `NULL`.

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
        -> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
        -> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
        -> NULL
```

- `MAKETIME(`***`hour`***`,`***`minute`***`,`***`second`***`)`

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

  The ***`second`*** argument can have a fractional part.

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

- `MICROSECOND(`***`expr`***`)`

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

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('2019-12-31 23:59:59.000010');
        -> 10
```

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

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

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
        -> 5
```

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

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

```
mysql> SELECT MONTH('2008-02-03');
        -> 2
```

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

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

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

- `NOW([`***`fsp`***`])`

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

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

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

`NOW()` 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**
  >
  > The period argument ***`P`*** is *not* a date value.

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

```
mysql> SELECT PERIOD_ADD(200801,2);
        -> 200803
```

- `PERIOD_DIFF(`***`P1`***`,`***`P2`***`)`

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

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

```
mysql> SELECT PERIOD_DIFF(200802,200703);
        -> 11
```

- `QUARTER(`***`date`***`)`

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

```
mysql> SELECT QUARTER('2008-04-01');
        -> 2
```

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

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

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

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

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

  The function returns `NULL` if ***`seconds`*** is `NULL`.

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

- `STR_TO_DATE(`***`str`***`,`***`format`***`)`

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 ***`str`*** or ***`format`*** is `NULL`, the function returns `NULL`. If the date, time, or datetime value extracted from ***`str`*** cannot be parsed according to the rules followed by the server, `STR_TO_DATE()` returns `NULL` and produces a warning.

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

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
        -> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
        -> '2013-05-01'
```

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

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
        -> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
        -> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
        -> '09:30:17'
```

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

```
mysql> SELECT STR_TO_DATE('abc','abc');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
        -> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
        -> '00:00:09'
```

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

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
        -> '2004-04-31'
```

If the `NO_ZERO_DATE` SQL mode is enabled, zero dates are disallowed. In that case, `STR_TO_DATE()` returns `NULL` and generates a warning:

```
mysql> SET sql_mode = '';
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
+---------------------------------------+
| STR_TO_DATE('00/00/0000', '%m/%d/%Y') |
+---------------------------------------+
| 0000-00-00                            |
+---------------------------------------+
mysql> SET sql_mode = 'NO_ZERO_DATE';
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
+---------------------------------------+
| STR_TO_DATE('00/00/0000', '%m/%d/%Y') |
+---------------------------------------+
| NULL                                  |
+---------------------------------------+
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1411
Message: Incorrect datetime value: '00/00/0000' for function str_to_date
```

Prior to MySQL 8.0.35, it was possible to pass an invalid date string such as `'2021-11-31'` to this function. In MySQL 8.0.35 and later, `STR_TO_DATE()` performs complete range checking and raises an error if the date after conversion would be invalid.

> **Note**
>
> You cannot use format `"%X%V"` to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:
>
> ```
> mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
>         -> '2004-10-18'
> ```

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

- `SUBDATE(`***`date`***`,INTERVAL `***`expr unit`***`)`, `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'
  ```

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

- `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.

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

  ```
  mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
          -> '2007-12-30 22:58:58.999997'
  mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
          -> '-00:59:59.999999'
  ```

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

- `SYSDATE([`***`fsp`***`])`

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

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

  `SYSDATE()` 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 replication source server and the replica.

  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. Returns `NULL` if ***`expr`*** is `NULL`.

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

```
mysql> SELECT TIME('2003-12-31 01:02:03');
        -> '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 strings which are converted to `TIME` or `DATETIME` expressions; these must be of the same type following conversion. Returns `NULL` if ***`expr1`*** or ***`expr2`*** is `NULL`.

  The result returned by `TIMEDIFF()` is limited to the range allowed for `TIME` values. Alternatively, you can use either of the functions `TIMESTAMPDIFF()` 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. Returns `NULL` if ***`expr`***, ***`expr1`***, or ***`expr2`*** is `NULL`.

```
mysql> SELECT TIMESTAMP('2003-12-31');
        -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
        -> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(`***`unit`***`,`***`interval`***`,`***`datetime_expr`***`)`

  Adds the integer expression ***`interval`*** to the date or datetime expression ***`datetime_expr`***. The unit for ***`interval`*** is given by the ***`unit`*** argument, which should be one of the following values:

`MICROSECOND` (microseconds), `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

The ***`unit`*** value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

This function returns `NULL` if ***`interval`*** or ***`datetime_expr`*** is `NULL`.

```
mysql> SELECT TIMESTAMPADD(MINUTE, 1, '2003-01-02');
        -> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
        -> '2003-01-09'
```

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

```
mysql> SELECT TIMESTAMPADD(MONTH, 1, DATE '2024-03-30') AS t1,
     >        TIMESTAMPADD(MONTH, 1, DATE '2024-03-31') AS t2;
+------------+------------+
| t1         | t2         |
+------------+------------+
| 2024-04-30 | 2024-04-30 |
+------------+------------+
1 row in set (0.00 sec)
```

- `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.

  This function returns `NULL` if ***`datetime_expr1`*** or ***`datetime_expr2`*** is `NULL`.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
        -> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
        -> -1
mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
        -> 128885
```

Note

> The order of the date or datetime arguments for this function is the opposite of that used with the `TIMESTAMP()` 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` or `0`. `TIME_FORMAT()` returns `NULL` if ***`time`*** or ***`format`*** is `NULL`.

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

  ```
  mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
          -> '100 100 04 04 4'
  ```

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

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

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

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

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

  ```
  mysql> SELECT TO_DAYS(950501);
          -> 728779
  mysql> SELECT TO_DAYS('2007-10-07');
          -> 733321
  ```

  `TO_DAYS()` 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 11.2.7, “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 11.2, “Date and Time Data Types”. For example, `'2008-10-07'` and `'08-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
        -> 733687, 733687
```

In MySQL, the zero date is defined as `'0000-00-00'`, even though this date is itself considered invalid. This means that, for `'0000-00-00'` and `'0000-01-01'`, `TO_DAYS()` 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 (including `NULL`), it returns `NULL`.

```
mysql> SELECT TO_SECONDS(950501);
        -> 62966505600
mysql> SELECT TO_SECONDS('2009-11-29');
        -> 63426672000
mysql> SELECT TO_SECONDS('2009-11-29 13:43:32');
```

```
        -> 63426721412
mysql> SELECT TO_SECONDS( NOW() );
        -> 63426721458
```

Like TO_DAYS(), 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 11.2.7, “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 11.2, “Date and Time Data Types”.

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

```
mysql> SELECT TO_SECONDS('0000-00-00');
+--------------------------+
| TO_SECONDS('0000-00-00') |
+--------------------------+
|                     NULL |
+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_SECONDS('0000-01-01');
+--------------------------+
| TO_SECONDS('0000-01-01') |
+--------------------------+
|                    86400 |
+--------------------------+
1 row in set (0.00 sec)
```

This is true whether or not the ALLOW_INVALID_DATES SQL server mode is enabled.

- UNIX_TIMESTAMP([***date***])

  If UNIX_TIMESTAMP() is called with no ***date*** argument, it returns a Unix timestamp representing seconds since '1970-01-01 00:00:00' UTC.

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 server interprets ***date*** as a value in the session time zone and converts it to an internal Unix timestamp value in UTC. (Clients can set the session time zone as described in Section 5.1.15, “MySQL Server Time Zone Support”.) The ***date*** argument may be a DATE, DATETIME, or TIMESTAMP string, or a number in ***YYMMDD***, ***YYMMDDhhmmss***, ***YYYYMMDD***, or ***YYYYMMDDhhmmss*** format. If the argument includes a time part, it may optionally include a fractional seconds part.

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

When the ***date*** argument is a TIMESTAMP column, UNIX_TIMESTAMP() returns the internal timestamp value directly, with no implicit “string-to-Unix-timestamp” conversion.

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

Regardless of MySQL version or platform architecture, if you pass an out-of-range date to UNIX_TIMESTAMP(), it returns 0. If ***date*** is NULL, it returns NULL.

```
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
```

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

```
mysql> SET time_zone = 'MET';
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT FROM_UNIXTIME(1111885200);
+---------------------------+
| FROM_UNIXTIME(1111885200) |
+---------------------------+
| 2005-03-27 03:00:00       |
+---------------------------+
```

Note

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

If you want to subtract `UNIX_TIMESTAMP()` columns, you might want to cast them 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 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 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 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 5.1.8, "Server System Variables". For a `NULL` date value, the function returns `NULL`.

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

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

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

  - If the week containing January 1 has 4 or more days in the new year, it is week 1.
  - Otherwise, it is the last week of the previous year, and the next week is week 1.

```
mysql> SELECT 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
```

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). Returns `NULL` if ***date*** is `NULL`.

```
mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
        -> 6
mysql> SELECT WEEKDAY('2007-11-06');
        -> 1
```

- `WEEKOFYEAR(`***date***`)`

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

`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. Returns `NULL` if ***`date`*** is `NULL`.

  ```
  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. Returns `NULL` if ***`date`*** is `NULL`.

  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
  ```

  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.

© 2024 Oracle