

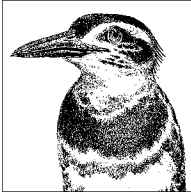
# MYSQL

## IN A NUTSHELL

*A Desktop Quick Reference*

O'REILLY®

*Russell J.T. Dyer*



# 6

## Date and Time Functions

The ability to record dates and times in a MySQL database is a very common requirement. This chapter presents the date and time functions for MySQL.

Date and time data comprises only numeric strings, so it can be stored in a regular character column. However, by using temporal datatype columns, you can use several built-in functions offered by MySQL. Currently, five temporal datatypes are available: `date`, `time`, `datetime`, `timestamp`, and `year`. The `date` column type is only for recording the date and uses the format `yyyy-mm-dd`. The `time` column type is for recording time in the format `hh:mm:ss`. To record a combination of date and time, you can use the `datetime` column type: `yyyy-mm-dd hh:mm:ss`. The `timestamp` column is similar to `datetime`, but is a little limited in its range of allowable time: it starts at the Unix epoch time (i.e., 1970-01-01) and ends at the end of 2037. Finally, the `year` datatype is used only for recording the year in a column.

Incidentally, any function that calls for a date or a time datatype will also accept a combined `datetime` datatype. For more information on date and time datatypes, see Appendix A.

Validation of date strings is limited: MySQL makes sure that months range only from 0 to 12, and days range from 0 to 31. Therefore, a date such as February 30 would be accepted. Version 5.0.2 of MySQL will offer more refined validation that would reject such a date.

At the end of this introduction is a listing of date and time functions, grouped by type of function. The bulk of this chapter consists of an alphabetical listing of date and time functions, with explanations of each. Many functions come with examples, along with a resulting display. For help in locating functions, see the index at the back of this book.

For the examples in this chapter, I used the scenario of a professional services firm (e.g., a law firm or an investment advisory firm) that tracks appointments and seminars in MySQL.

# Date and Time Functions Grouped by Type

This section lists the functions according to their purpose: to retrieve a time, extract an element of one, or perform calculations on it.

---

## Determining the Date and Time

`CURDATE()`, `CURRENT_DATE`, `CURTIME()`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`, `NOW()`, `LOCALTIME()`, `LOCALTIMESTAMP()`, `SYSDATE()`, `UNIX_TIMESTAMP()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`

---

## Extracting and Formatting the Date and Time

`DATE()`, `DATE_FORMAT()`, `DAY()`, `DAYNAME()`, `DAYOFMONTH()`, `DAYOFWEEK()`, `DAYOFYEAR()`, `EXTRACT()`, `GET_FORMAT()`, `hour()`, `LAST_DAY()`, `MAKEDATE()`, `MAKETIME()`, `MINUTE()`, `MONTH()`, `MONTHNAME()`, `QUARTER()`, `SECOND()`, `STR_TO_DATE()`, `TIME_FORMAT()`, `TIMESTAMP()`, `WEEK()`, `WEEKDAY()`, `WEEKOFYEAR()`, `YEAR()`, `YEARWEEK()`

---

## Calculating and Modifying the Date and Time

`ADDDATE()`, `ADDTIME()`, `CONVERT_TZ()`, `DATE_ADD()`, `DATE_SUB()`, `DATEDIFF()`, `FROM_DAYS()`, `FROM_UNIXTIME()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `SEC_TO_TIME()`, `SUBDATE()`, `SUBTIME()`, `TIME_TO_SEC()`, `TO_DAYS()`, `TIMEDIFF()`, `TIMESTAMPADD()`, `TIMESTAMPDIFF()`

# Date and Time Functions in Alphabetical Order

The rest of the chapter lists each function in alphabetical order.

---

## ADDDATE()

`ADDDATE(date, INTERVAL value type)`

This function adds the given interval of time to the date or time provided. This is an alias for `DATE_ADD()`; see its definition for details and interval types.

```
UPDATE seminars
  SET seminar_date = ADDDATE(seminar_date, INTERVAL 7 DAY)
  WHERE seminar_date = '2004-12-15';
```

This example postpones the seminar that was scheduled for December 15, 2004 to December 22—seven days later. As of Version 4.1 of MySQL, for adding days the second argument of the function may simply be the number of days (i.e., just 7 instead of `INTERVAL 7 DAY`).

---

## ADDTIME()

`ADDTIME(datetime, datetime)`

This function returns the date and the time for given string or column, incremented by the time given as the second argument (*d hh:mm:ss*). If a negative number is given, the time is subtracted, and the function is the equivalent of `SUBTIME()`. This function is available as of Version 4.1.1 of MySQL.

```
SELECT NOW() AS Now,
       ADDTIME(NOW(), '1:00:00.00') AS 'Hour Later';
```

```
+-----+-----+
| Now           | Hour Later   |
+-----+-----+
| 2005-01-11 23:20:30 | 2005-01-12 00:20:30 |
+-----+-----+
```

Notice that the hour was increased by one, and because the time is near midnight, the function causes the date to be altered by one day, as well. To increase the date, add the number of days before the time (separated by a space) like so:

```
SELECT NOW() AS Now,
       ADDTIME(NOW(), '30 0:0:0') AS 'Thirty Days Later';
```

```
+-----+-----+
| NOW()         | Thirty Days Later |
+-----+-----+
| 2005-01-11 23:20:30 | 2005-02-10 23:20:30 |
+-----+-----+
```

---

## CONVERT\_TZ()

`CONVERT_TZ(datetime, day)`

This function converts a given date and time from one given time zone to another. It requires time-zone tables to be installed in the *mysql* database. If they're not already installed on your system, go to MySQL AB's site (<http://dev.mysql.com/downloads/time-zones.html>) to download the tables. Copy them into the *mysql* subdirectory of the *data* directory of MySQL. Change the ownership to the system *mysql* user with a system command like *chmod* and restart the server. This function is available as of Version 4.1.3 of MySQL.

```
SELECT NOW() AS 'New Orleans',
       CONVERT_TZ(NOW(), 'US/Central', 'Europe/Berlin')
       AS Berlin;
```

```
+-----+-----+
| New Orleans   | Berlin       |
+-----+-----+
| 2005-01-12 01:37:11 | 2005-01-12 08:37:11 |
+-----+-----+
```

This example retrieves the current time of the server located in New Orleans and converts it to the time in Berlin.

---

## CURDATE()

CURDATE()

This function returns the current system date in *yyyy-mm-dd* format. It will return the date in a *yyyymmdd* format if it's used as part of a numeric calculation (see example). You can use the function in SELECT statements as shown here, in INSERT and UPDATE statements to set a value, or in a WHERE clause. CURDATE() is synonymous with CURRENT\_DATE; see its definition for more details.

```
SELECT CURDATE() AS Today,  
       CURDATE() + 1 AS Tomorrow;
```

```
+-----+-----+  
| Today   | Tomorrow |  
+-----+-----+  
| 2005-01-15 | 20050116 |  
+-----+-----+
```

Because the second use of the function involves a numeric calculation, tomorrow's date is displayed without dashes.

---

## CURRENT\_DATE()

CURRENT\_DATE()

This function returns the current date. The usual parentheses are not required. This function is synonymous with CURDATE(). You can use both in SELECT statements to dynamically set values or in WHERE clauses.

```
UPDATE appointment  
SET appt_date = CURRENT_DATE()  
WHERE rec_id = '1250';
```

This statement changes the appointment date for a client that came in today unexpectedly.

---

## CURRENT\_TIME()

CURRENT\_TIME()

This function returns the current time in *hh:mm:ss* format. It will return the time in an *hhmmss* format if it's used as part of a numeric calculation. The usual parentheses are not required. It's synonymous with CURTIME().

```
INSERT INTO appointments  
  (client_id, appt_date, start_time)  
VALUES('1403', CURRENT_DATE(), CURRENT_TIME());
```

In this example, we're logging an unscheduled appointment that has just begun so that we can bill the client later.

---

## CURRENT\_TIMESTAMP()

CURRENT\_TIMESTAMP()

This function returns current date and time in *yyyy-mm-dd hh:mm:ss* format. It will return the time in a *yyyymmddhhmmss* format if it's used as part of a numeric calculation (see example). Parentheses aren't required.

```
SELECT CURRENT_TIMESTAMP() AS Now,  
       CURRENT_TIMESTAMP() + 10000 AS 'Hour Later';
```

```
+-----+-----+  
| Now           | Hour Later   |  
+-----+-----+  
| 2005-01-12 16:41:47 | 20050112174147 |  
+-----+-----+
```

By adding 10,000 to the current time, the hour is increased by 1 and the minutes and seconds by zero each, and the time is displayed in the second field without dashes.

---

## CURTIME()

CURTIME()

This function returns the current system time in *hh:mm:ss* format. It will return the time in an *hhmmss* format if it's used as part of a numeric calculation (see example). This is an alias for `CURRENT_TIME()`.

```
SELECT CURTIME() AS Now,  
       CURTIME() + 10000 AS 'Hour Later';
```

```
+-----+-----+  
| Now          | Hour Later   |  
+-----+-----+  
| 16:35:43    | 163643      |  
+-----+-----+
```

By adding 10,000 to the current time, this statement increases the hour by 1 and the minutes and seconds by zero each, and displays the time in the second field without dashes.

---

## DATE()

DATE(*date*)

This function returns the date from a given string. This function is available as of Version 4.1.1 of MySQL.

```
SELECT appointment, DATE(appointment)  
   FROM appointments  
   WHERE client_id = '8639' LIMIT 1;
```

```
+-----+-----+  
| appointment          | DATE(appointment) |  
+-----+-----+  
| 2005-01-11 14:11:43 | 2005-01-11        |  
+-----+-----+
```

In this SQL statement, the value of the *appointment* column, which is a DATETIME type column, is shown first. The second field is the date extracted by the function from the same column.

---

## DATE\_ADD()

DATE\_ADD(*date*, INTERVAL *value type*)

This function extracts time or date information and, thereby, adds time to the value extracted. It's synonymous with the ADDDATE() function.

```
UPDATE appointments
  SET appt_date = DATE_ADD(appt_date, INTERVAL 1 DAY)
  WHERE rec_id='1202';
```

In this example, the appointment date is changed to its current value plus, one additional day (i.e., we're postponing the appointment by one day). If we changed the 1 to a -1, MySQL would subtract a day instead. The format of value depends on the type and is shown in Table 6-1.

Table 6-1. DATE\_ADD intervals and formats of values

Type of increment	Description
DAY	dd
DAY_HOUR	dd hh
DAY_MINUTE	dd hh:mm
DAY_SECOND	dd hh:mm:ss
HOUR	hh
HOUR_MINUTE	hh:mm
HOUR_SECOND	hh:mm:ss
MINUTE	mm
MINUTE_SECOND	mm:ss
MONTH	mm
SECOND	ss
YEAR	yyyy
YEAR_MONTH	yy-mm

---

## DATE\_FORMAT()

DATE\_FORMAT(*date*, '*format\_code*')

This function returns date and time information in a format desired, based on formatting codes listed in the second argument of function.

```
SELECT DATE_FORMAT(appointment, '%W - %M %e, %Y at %r')
  AS 'Appointment'
FROM appointments
WHERE client_id = '8392'
  AND appointment > CURDATE();
```

```

+-----+
| Appointment |
+-----+
| Tuesday - June 15, 2004 at 01:00:00 PM |
+-----+

```

Using the formatting codes, we're specifying that we want the name of the day of the week followed by a dash and then the date of the appointment in a typical U.S. format, with the month name and a comma after the day. We're ending with the word "at" followed by the full non-military time. Table 6-2 contains a list of all the formatting codes you can use with `DATE_FORMAT()`. You can also use these codes with `TIME_FORMAT()` and `EXTRACT()`.

Table 6-2. `DATE_FORMAT()` format codes and resulting formats

Code	Example
<code>%%</code>	A literal <code>'%'</code>
<code>%a</code>	Abbreviated weekday name (Sun...Sat)
<code>%b</code>	Abbreviated month name (Jan...Dec)
<code>%c</code>	Month, numeric (1...12)
<code>%d</code>	Day of the month, numeric (00...31)
<code>%D</code>	Day of the month with English suffix (1st, 2nd, 3rd, etc.)
<code>%e</code>	Day of the month, numeric (0...31)
<code>%h</code>	Hour (01...12)
<code>%H</code>	Hour (00...23)
<code>%i</code>	Minutes, numeric (00...59)
<code>%l</code>	Hour (01...12)
<code>%j</code>	Day of the year (001...366)
<code>%k</code>	Hour (0...23)
<code>%l</code>	Hour (1...12)
<code>%m</code>	Month, numeric (01...12)
<code>%M</code>	Month name (January...December)
<code>%p</code>	AM or PM
<code>%r</code>	Time, 12-hour (hh:mm:ss [AP]M)
<code>%s</code>	Seconds (00...59)
<code>%S</code>	Seconds (00...59)
<code>%T</code>	Time, 24-hour (hh:mm:ss)
<code>%u</code>	Week (0...52), where Monday is the first day of the week
<code>%U</code>	Week (0...52), where Sunday is the first day of the week
<code>%v</code>	Week (1...53), where Monday is the first day of the week; used with <code>'%x'</code>
<code>%V</code>	Week (1...53), where Sunday is the first day of the week; used with <code>'%X'</code>
<code>%w</code>	Day of the week (0=Sunday...6=Saturday)
<code>%W</code>	Weekday name (Sunday...Saturday)
<code>%x</code>	Year for the week, where Monday is the first day of the week, numeric, 4 digits; used with <code>'%v'</code>
<code>%X</code>	Year for the week, where Sunday is the first day of the week, numeric, 4 digits; used with <code>'%V'</code>
<code>%y</code>	Year, numeric, 2 digits
<code>%Y</code>	Year, numeric, 4 digits

---

## DATE\_SUB()

`DATE_SUB(date, INTERVAL value type)`

Use this function to subtract from the results of a date or time datatype column. See `DATE_ADD()` for a table of incremental types.

```
SELECT DATE_SUB(NOW(), INTERVAL 1 DAY)
       AS Yesterday;
```

```
+-----+
| Yesterday |
+-----+
| 2004-05-08 07:05:08 |
+-----+
```

This statement was entered on the morning of May 9, a little after 7 a.m. Notice that the time remains unchanged, but the date was reduced by one day. By placing a negative sign in front of the value the reverse effect would occur, giving a result of May 10 in this example. Whatever intervals that can be used with `DATE_ADD()` can be used with `DATE_SUB()`.

---

## DATEDIFF()

`DATEDIFF(date, date)`

This function returns the number of days in the difference between the two dates given. Although a parameter may be given in date and time format, only the date is used for determining the difference. This function is available as of Version 4.1.1 of MySQL.

```
SELECT CURDATE() AS Today,
       DATEDIFF('2005-12-25', NOW())
       AS 'Days to Christmas';
```

```
+-----+-----+
| Today | Days to Christmas |
+-----+-----+
| 2005-01-11 | 348 |
+-----+-----+
```

---

## DAY()

`DAY(date)`

This function returns the day of the month for a given date provided. This function is available as of Version 4.1.1 of MySQL. It's synonymous with the `DAYOFMONTH()` function.

```
SELECT DAY('2005-12-15') AS 'Day';
```

```
+-----+
| Day |
+-----+
| 15 |
+-----+
```

This function is more meaningful when you apply it to a date column where the date is unknown before entering the SQL statement.

---

## DAYNAME( )

DAYNAME(*date*)

This function returns the name of the day for the date provided.

```
SELECT DAYNAME(appt_date) AS Appointment
FROM appointments
WHERE rec_id = '1439';
```

```
+-----+
| Appointment |
+-----+
| Saturday   |
+-----+
```

---

## DAYOFMONTH( )

DAYOFMONTH(*date*)

This function returns the day of the month for the date given. It returns NULL if the day for the date is greater than 31. It will accept a date like 2005-02-31 as valid and will return 31. Future releases of MySQL will resolve this problem.

```
SELECT DAYOFMONTH('2005-03-01')
       AS 'Day of Month';
```

```
+-----+
| Day of Month |
+-----+
|              1 |
+-----+
```

This is more meaningful when you apply it to a date column where the date is unknown before entering the SQL statement.

---

## DAYOFWEEK( )

DAYOFWEEK(*date*)

This function returns numerical day of the week for a given date. Sunday returns a value of 1, and Saturday returns a value of 7.

```
SELECT DAYOFWEEK('2005-03-01')
       AS 'Day of Week',
       DAYNAME('2005-03-01')
       AS 'Name of Day';
```

```
+-----+-----+
| Day of Week | Name of Day |
+-----+-----+
|              3 | Tuesday     |
+-----+-----+
```

In this example, the date is on the third day of the week, which is a Tuesday.

---

## DAYOFYEAR( )

DAYOFYEAR(*date*)

This function returns the day of the year. January 1 would give a value of 1, and December 31 would normally be 365, except on leap years, when it would be 366.

```
SELECT (DAYOFYEAR('2004-03-01') -
        DAYOFYEAR('2004-02-28'))
        AS 'Difference for Leap Year';
```

```
+-----+
| Difference for Leap Year |
+-----+
|                          2 |
+-----+
```

Here we are using the function to calculate the number of days from the first date, March 1, 2004, to the second date, February 28 of the same year. 2004 was a leap year, so the result is 2 days.

---

## EXTRACT( )

EXTRACT(*type* FROM *date*)

This function extracts date or time information from a date in the format type requested. The acceptable types are the same as for DATE\_ADD( ).

```
SELECT EXTRACT(HOUR_MINUTE FROM NOW())
        AS 'Time Now';
```

```
+-----+
| Time Now |
+-----+
|      1121 |
+-----+
```

When this SQL statement was run, it was 11:21 a.m.

---

## FROM\_DAYS( )

FROM\_DAYS(*value*)

This function returns the date based on the number of days given, which are from the beginning of the currently used standard calendar. Problems occur for dates before 1582 when the Gregorian calendar became the standard. The opposite of this is TO\_DAYS( ).

```
SELECT FROM_DAYS((365.25*2005))
        AS 'Start of 2005?';
```

```
+-----+
| Start of 2005? |
+-----+
| 2005-01-15    |
+-----+
```

Assuming that there are 365.25 days in a year on average (allowing for the leap year), you would think that multiplying that factor by 2005 would give a result of January 1, 2005, but it doesn't because of the calendar changes centuries ago. This function is useful for comparing dates, not for determining static information on one date.

---

## FROM\_UNIXTIME()

`FROM_UNIXTIME(unix_timestamp [, format])`

This function returns the date based on Unix time, which is the number of seconds since January 1, 1970, Greenwich Mean Time (GMT), with 12:00:01 being the first second of Unix time (the epoch). Optionally the results may be formatted using the formatting codes from `DATE_FORMAT()`. It will return the date and time in a `yyyy-mm-dd hh:mm:ss` format, but will return the data in a `yyyymmdd` format if it's used as part of a numeric calculation.

```
SELECT FROM_UNIXTIME(0)
       AS 'My Epoch Start';
```

```
+-----+
| My Epoch Start |
+-----+
| 1969-12-31 18:00:00 |
+-----+
```

Here we're selecting the date based on zero seconds since the start of Unix time. The results are off by six hours because I'm not located in the GMT zone. This function is typically used on columns in which their values were derived from `UNIXTIME_STAMP()`.

---

## GET\_FORMAT()

`GET_FORMAT(data_type, format_type)`

This function returns the format for a format type given as the second argument for a datatype given as the first. The format codes returned are the same codes used by the `DATE_FORMAT()` function. Four datatypes may be given: `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP`. Five format types may be given as the second argument: `EUR`, `INTERNAL`, `ISO`, `JIS`, and `USA`. This function is available as of Version 4.1.1 of MySQL. The `TIMESTAMP` datatype isn't acceptable until Version 4.1.4. Here's an example using the function that returns the USA format:

```
SELECT GET_FORMAT(DATE, 'USA');
```

```
+-----+
| GET_FORMAT(DATE, 'USA') |
+-----+
| %m.%d.%Y                |
+-----+
```

You can hand off to the `DATE_FORMAT()` function for formatting the value of a date column like so:

```
SELECT DATE_FORMAT(appointment, GET_FORMAT(DATE, 'USA'))
       AS Appointment LIMIT 1;
```

```

+-----+
| Appointment |
+-----+
| 01.11.2005 |
+-----+

```

Table 6-3 lists the results for the different combinations. The datatype of `TIMESTAMP` is not listed, because the results are the same as `DATETIME`.

Table 6-3. `DATE_FORMAT` arguments and their results

Combination	Results
<code>DATE, 'EUR'</code>	<code>%d.%m.%Y</code>
<code>DATE, 'INTERNAL'</code>	<code>%Y%m%d</code>
<code>DATE, 'ISO'</code>	<code>%Y-%m-%d</code>
<code>DATE, 'JIS'</code>	<code>%Y-%m-%d</code>
<code>DATE, 'USA'</code>	<code>%m.%d.%Y</code>
<code>TIME, 'EUR'</code>	<code>%H.%i.%S</code>
<code>TIME, 'INTERNAL'</code>	<code>%H%i%s</code>
<code>TIME, 'ISO'</code>	<code>%H:%i:%s</code>
<code>TIME, 'JIS'</code>	<code>%H:%i:%s</code>
<code>TIME, 'USA'</code>	<code>%h:%i:%s %p</code>
<code>DATETIME, 'EUR'</code>	<code>%Y-%m-%d-%H.%i.%s</code>
<code>DATETIME, 'INTERNAL'</code>	<code>%Y%m%d%H%i%s</code>
<code>DATETIME, 'ISO'</code>	<code>%Y-%m-%d %H:%i:%s</code>
<code>DATETIME, 'JIS'</code>	<code>%Y-%m-%d %H:%i:%s</code>
<code>DATETIME, 'USA'</code>	<code>%Y-%m-%d-%H.%i.%s</code>

## hour()

`hour(time)`

This function returns the hour for the time given. For column types containing the time of day (e.g., `DATETIME`), the range of results will be from 0 to 23. For `TIME` datatype columns that contain data not restricted to day limits, this function may return values greater than 23.

```

SELECT hour(appointment)
FROM appointments
WHERE client_id = '3992'
AND appointment > CURDATE();

```

```

+-----+
| hour(appointment) |
+-----+
|                13 |
+-----+

```

This statement is selecting the upcoming appointment for a particular client. The hour is returned in military time (i.e., 13 is 1 p.m.).

---

## LAST\_DAY()

LAST\_DAY(*date*)

This function returns the date of the last day of the month for a given date. This function is available as of Version 4.1.1 of MySQL.

```
SELECT LAST_DAY('2005-12-15')
       AS 'End of Month';
```

```
+-----+
| End of Month |
+-----+
| 2005-12-31  |
+-----+
```

This function is more meaningful when you apply it to a date column where the date is unknown before entering the SQL statement.

---

## LOCALTIME()

LOCALTIME()

This function returns the current system date in *yyyy-mm-dd hh:mm:ss* format. You can use it in SELECT statements as shown here, in INSERT and UPDATE statements to set a value, or in a WHERE clause. This statement is available as of Version 4.0.6 of MySQL. Incidentally, the parentheses are not required. This function is synonymous with LOCALTIMESTAMP() and NOW().

```
SELECT LOCALTIME();
```

```
+-----+
| LOCALTIME() |
+-----+
| 2005-01-10 14:27:41 |
+-----+
```

---

## LOCALTIMESTAMP()

LOCALTIMESTAMP()

This function returns the current system date in *yyyy-mm-dd hh:mm:ss* format. You can use it in SELECT statements as shown, in INSERT and UPDATE statements to set a value, or in a WHERE clause. This statement is available as of Version 4.0.6 of MySQL. Incidentally, the parentheses are not required. This function is synonymous with LOCALTIME() and NOW().

```
SELECT LOCALTIMESTAMP();
```

```
+-----+
| LOCALTIMESTAMP() |
+-----+
| 2005-01-10 14:49:56 |
+-----+
```

---

## MAKEDATE()

MAKEDATE(*year*, *day*)

This function converts a given day of the year of a given year to a date in *yyyy-mm-dd* format. The day given can be from 1 to 365 (366 for leap years). It returns NULL if a value less is than 1 or greater than the maximum number of days allowed. This function is available as of Version 4.1.1 of MySQL.

```
SELECT MAKEDATE(2005, 1) AS 'First Day',
       MAKEDATE(2005, 365) AS 'Last Day';
```

```
+-----+-----+
| First Day | Last Day |
+-----+-----+
| 2005-01-01 | 2005-12-31 |
+-----+-----+
```

---

## MAKETIME()

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

This function converts a given hour, minute, and second to *hh:mm:ss* format. It returns NULL if the value for *minute* is greater than 60 or if *second* is greater than 59. It will accept an hour greater than 24. This function is available as of Version 4.1.1 of MySQL.

```
SELECT MAKETIME(14, 32, 5)
       AS Time;
```

```
+-----+
| Time   |
+-----+
| 14:32:05 |
+-----+
```

---

## MICROSECOND()

MICROSECOND(*time*)

This function extracts the microseconds value of a given time. This function is available as of Version 4.1.1 of MySQL.

```
SELECT MICROSECOND('2005-01-11 19:28:45.80')
       AS 'MicroSecond';
```

```
+-----+
| MicroSecond |
+-----+
|          800000 |
+-----+
```

---

## MINUTE()

MINUTE(*time*)

This function returns the minute value (0–59) of a given time.

```
SELECT CONCAT(HOUR(appointment), ':',
              MINUTE(appointment)) AS 'Appointment'
FROM appointments
WHERE client_id = '3992'
AND appointment > CURDATE();
```

```
+-----+
| Appointment |
+-----+
|      13:30 |
+-----+
```

This statement is using the string function `CONCAT()` to paste together the hour and the minute, with a colon as a separator.

---

## MONTH()

MONTH(*date*)

This function returns the numeric value of the month (1–12) for the date provided.

```
SELECT MONTH(appointment)
        AS Appointment
FROM appointments
WHERE client_id = '8302'
AND appointment > CURDATE();
```

```
+-----+
| Appointment |
+-----+
|           6 |
+-----+
```

This SQL statement is retrieving the month of any appointments after the current date for a particular client. There's only one appointment, and it's in June.

---

## MONTHNAME()

MONTHNAME(*date*)

This function returns the name of the month for the date provided.

```
SELECT MONTHNAME(appointment)
        AS 'Appointment'
FROM appointments
WHERE client_id = '8302'
AND appointment > NOW();
```

```

+-----+
| Appointment |
+-----+
| June       |
+-----+

```

In this example, the client has only one appointment after the current date, and it's in June. As you can with any date and time function, you can use this one in conjunction with other formatting functions like `CONCAT()` for human formatting of data.

---

## NOW()

`NOW()`

This function returns the current date and time. It will return the date and time in a `yyyy-mm-dd hh:mm:ss` format, but will return the data in a `yyyymmdd` format if it's used as part of a numeric calculation. It's synonymous with `LOCALTIME()`.

```

SELECT NOW() AS Now,
       NOW() + 10000 AS 'Hour Later';

```

```

+-----+-----+
| Now           | Hour Later   |
+-----+-----+
| 2005-01-12 17:10:28 | 20050112181028 |
+-----+-----+

```

By adding 10,000 to the current time, the hour is increased by 1 and the minutes and seconds by zero each, and the time is displayed in the second field, without dashes.

---

## PERIOD\_ADD()

`PERIOD_ADD(yearmonth, value)`

This function adds a specified number of months to the year and month given. The date must be in string format and can contain only the year and month in either `yyyymm` or `yyymm` format. The value for the second argument of the function specifies the number of months to add to the period given.

```

SELECT PERIOD_ADD(200412, 1)
       AS 'Next Accounting Period';

```

```

+-----+
| Next Accounting Period |
+-----+
|           200501      |
+-----+

```

Functions such as this one are particularly useful when you're building a script or program, and you need to design an SQL statement that will account for accounting periods that roll into new years.

---

## PERIOD\_DIFF()

PERIOD\_DIFF(*yearmonth*,*yearmonth*)

This function returns the number of months between periods. The dates must be in string format and contain only the year and month, in either *yyyymm* or *yymm* format.

```
SELECT PERIOD_DIFF(200403, 200401)
       AS 'Accounting Periods Apart';
```

```
+-----+
| Accounting Periods Apart |
+-----+
|                          2 |
+-----+
```

The first period is for March 2004 and the second is for January 2004. This function doesn't work on standard date columns unless you put them into the format shown here. An example of doing this conversion is:

```
SELECT PERIOD_DIFF(EXTRACT(YEAR_MONTH FROM CURDATE()),
                   EXTRACT(YEAR_MONTH FROM appointment))
       AS 'Accounting Periods Apart'
FROM appointments
WHERE client_id = '5620'
ORDER BY appointment DESC;
```

```
+-----+
| Accounting Periods Apart |
+-----+
|                          -2 |
+-----+
```

This SQL statement determines that it's been two months since this client's last appointment. This works, but it's cumbersome.

---

## QUARTER()

QUARTER(*date*)

This function returns the number of quarters (1–4) for the date provided. The first three months of each year have a value of 1.

```
SELECT COUNT(appointment)
       AS 'Appts. Last Quarter'
FROM appointments
WHERE QUARTER(appointment) = (QUARTER(NOW()) - 1)
     AND client_id = '7393';
```

```
+-----+
| Appts. Last Quarter |
+-----+
|                      16 |
+-----+
```

In this example, we're having MySQL calculate the total number of appointments for a particular client that occurred before this quarter. The flaw in this SQL statement is

that it doesn't work when it's run during the first quarter of a year. In the first quarter, the calculation on the fourth line would produce a quarter value of 0. This statement also doesn't consider appointments in previous quarters of previous years. To solve these problems, we could set up user-defined variables for the values of the previous quarter and for its year:

```
SET @LASTQTR:=IF((QUARTER(CURDATE()-1) = 0,
    4, QUARTER(CURDATE()-1));

SET @YR:=IF(@LASTQTR = 4,
    YEAR(NOW()-1, YEAR(NOW()));

SELECT COUNT(appointment)
    AS 'Appts. Last Quarter'
FROM appointments
WHERE QUARTER(appointment) = @LASTQTR
    AND YEAR(appointment) = @YR
    AND client_id = '7393';
```

In the first SQL statement here, we're using an IF statement to test whether reducing the quarter by one would yield us a 0 value. If so, we'll set the user variable for the last quarter to 4. In the second statement we're establishing the year for the last quarter, based on the value determined for @LASTQTR. The last SQL statement selects rows and counts them where the QUARTER() function yields a value equal to the @LASTQTR variable and where the YEAR() function yields a value equal to the @YR variable based on the appointment date, and where the client is the one for which we are running the statement.

---

## SEC\_TO\_TIME()

SEC\_TO\_TIME(*seconds*)

This function returns the period for a given number of seconds, in the format *hh:mm:ss*. It will return the time in *hhmmss* format if it's used as part of a numeric calculation.

```
SELECT SEC_TO_TIME(3600)
    AS 'Actual Time';
```

```
+-----+
| Actual Time |
+-----+
| 01:00:00   |
+-----+
```

In this example, we have a value of 3,600 seconds into the day, which the function has translated to 1 a.m. Incidentally, if the number of seconds exceeds 86,400, or one day's worth, the value for hours will result in an amount greater than 23 and will not be reset back to 0.

---

## SECOND()

SECOND(*time*)

This function returns seconds value (0–59) for a given time.

```
SELECT NOW(), SECOND(NOW());
```

NOW()	SECOND(NOW())
2004-05-09 14:56:11	11

The first column generated shows the time that this statement was entered, using NOW() function. The second column displays only the seconds value for the results of NOW().

---

## STR\_TO\_DATE()

STR\_TO\_DATE(*datetime*, '*format\_code*')

This function returns the date and time of a given string for a given format. The function takes a string containing a date or time, or both. So that the function may convert the string given, the formatting code for the string needs to be provided. The formatting codes are the same codes used by the DATE\_FORMAT() function. This function is available as of Version 4.1.1 of MySQL.

```
SELECT STR_TO_DATE('January 15, 2005 1:30 PM',
                  '%M %d, %Y %h:%i %p')
       AS Anniversary;
```

Anniversary
2005-01-15 13:30:00

---

## SUBDATE()

SUBDATE(*date*, INTERVAL *value type*)

Use this function to subtract a time interval from the results of a date or time datatype column. If a negative value is given, the interval is added and is equivalent to the ADDDATE() function. This is an alias for the DATE\_SUB() function. See DATE\_ADD() for a table of incremental types.

```
SELECT SUBDATE(NOW(), INTERVAL 1 DAY)
       AS 'Yesterday',
       SUBDATE(NOW(), INTERVAL -1 DAY)
       AS 'Tomorrow';
```

Yesterday	Tomorrow
2004-05-09 16:11:56	2004-05-11 16:11:56

As of Version 4.1 of MySQL, for subtracting days the second argument of the function may simply be the number of days (i.e., just 1 instead of INTERVAL 1 DAY).

---

## SUBTIME()

SUBTIME(*datetime*, *datetime*)

This function returns the date and time for the given string or column, decreased by the time given as the second argument (*d hh:mm:ss*). If a negative number is given, the time is added and the function is the equivalent of ADDTIME(). This function is available as of Version 4.1.1 of MySQL.

```
SELECT NOW() AS Now,  
       SUBTIME(NOW(), '1:00:00.00') AS 'Hour Ago';
```

```
+-----+-----+  
| Now           | Hour Ago       |  
+-----+-----+  
| 2005-01-12 00:54:59 | 2005-01-11 23:54:59 |  
+-----+-----+
```

Notice that the hour was decreased by one, and because the time is just after midnight, the function causes the date to be altered by one day, as well. To decrease the date, add the number of days before the time (separated by a space) like so:

```
SELECT NOW() AS Now,  
       SUBTIME(NOW(), '30 0:0.0') AS 'Thirty Days Ago';
```

```
+-----+-----+  
| Now           | Thirty Days Ago |  
+-----+-----+  
| 2005-01-12 00:57:04 | 2004-12-13 00:57:04 |  
+-----+-----+
```

---

## SYSDATE()

SYSDATE()

This function returns the system date. It will return the date and time in a *yyyy-mm-dd hh:mm:ss* format, but will return the data in a *yyyymmdd* format if it's used as part of a numeric calculation. This function is an alias for the NOW() function.

```
SELECT SYSDATE();
```

```
+-----+  
| SYSDATE() |  
+-----+  
| 2004-05-09 18:44:51 |  
+-----+
```

---

## TIME()

TIME(*time*)

This function returns the time from a given string or column containing date and time data. This function is available as of Version 4.1.1 of MySQL.

```
SELECT TIME(NOW()) , NOW();
```

```
+-----+-----+
| TIME(NOW()) | NOW() |
+-----+-----+
| 21:14:20    | 2005-01-11 21:14:20 |
+-----+-----+
```

---

## TIMEDIFF()

TIMEDIFF(*time*, *time*)

This function returns the time difference between the two times given. Although the arguments may be given in time or date-and-time format, both arguments must be of the same datatype. This function is available as of Version 4.1.1 of MySQL.

```
SELECT appointment AS Appointment, NOW() AS Now,
       TIMEDIFF(appointment, NOW()) AS 'Time Remaining'
FROM appointments
WHERE rec_id='3783';
```

```
+-----+-----+-----+
| Appointment      | Now          | Time Remaining |
+-----+-----+-----+
| 2005-01-11 10:30:00 | 2005-01-11 22:28:09 | 12:01:51      |
+-----+-----+-----+
```

---

## TIMESTAMP()

TIMESTAMP(*date*, *time*)

This function returns date and time (in *yyyy-mm-dd hh:mm:ss* format) from a given string or column containing date and time data, respectively. If only the date or only the time is given, the function will return zeros for the missing parameters. This function is available as of Version 4.1.1 of MySQL.

```
SELECT TIMESTAMP(appt_date, appt_time)
FROM appointments LIMIT 1;
```

```
+-----+
| TIMESTAMP(appt_date, appt_time) |
+-----+
| 2005-01-15 10:30:00              |
+-----+
```

---

## TIMESTAMPDIFF()

`TIMESTAMPDIFF(interval, datetime, datetime)`

This function returns the time difference between the two times given but only for the interval being compared. The intervals accepted are the same as those for the `TIMESTAMPADD()` function. This function is available as of Version 5 of MySQL.

```
SELECT NOW() AS Today,  
       TIMESTAMPDIFF(DAY, NOW(), LAST_DAY(NOW()))  
       AS 'Days Remaining in Month';
```

```
+-----+-----+  
| Today           | Days Remaining in Month |  
+-----+-----+  
| 2005-01-12 02:19:26 | 19                       |  
+-----+-----+
```

This SQL statement retrieves the current date and time and uses the `LAST_DAY()` function to determine the date of the last day of the month. Then `TIMESTAMPDIFF()` function determines the difference between the day of the date now and the day for the date at the end of the month.

---

## TIMESTAMPADD()

`TIMESTAMPADD(interval, value, datetime)`

This function adds the given interval of time to the given date or time. The intervals accepted are `FRAC_SECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, and `YEAR`. For compatibility with other systems, you can add the `SQL_TSI_` prefix to these interval names (e.g., `SQL_TSI_YEAR`). This function is available as of Version 5 of MySQL. This function is similar to `DATE_ADD()`. In this example, an appointment is set to an hour later.

```
UPDATE appointments  
   SET appointment = TIMESTAMPADD(HOUR, 1, appointment)  
   WHERE rec_id = '8930';
```

---

## TIME\_FORMAT()

`TIME_FORMAT(time, format)`

This function returns the time value of a time element provided and formats it according to formatting codes given as the second argument in parentheses. See the `DATE_FORMAT()` function for formatting codes but only those related to time values. This function will return `NULL` for other formatting codes.

```
SELECT TIME_FORMAT(appointment, '%r')  
       AS 'Appt. Time' FROM appointments  
WHERE client_id = '8373'  
       AND appointment > SYSDATE();
```

```
+-----+  
| Appt. Time |  
+-----+  
| 01:00:00 PM |  
+-----+
```

---

## TIME\_TO\_SEC()

`TIME_TO_SEC(time)`

This function returns the number of seconds that the given time represents. This function is the inverse of `SEC_TO_TIME()`.

```
SELECT TIME_TO_SEC('01:00')
       AS 'Seconds up to 1 a.m.';
```

```
+-----+
| Seconds up to 1 a.m. |
+-----+
|                   3600 |
+-----+
```

Here we're calculating the number of seconds up until the time of 1 a.m. (i.e., 60 seconds times 60 minutes) or one hour into the day.

---

## TO\_DAYS()

`TO_DAYS(date)`

This function returns the date based on the number of days given, which are from the beginning of the currently used standard calendar. Problems occur for dates before 1582 when the Gregorian calendar became the standard. The opposite of this function is `FROM_DAYS()`.

```
SELECT (TO_DAYS('2005-01-15') -
       TO_DAYS('2005-01-01'))
       AS 'Difference';
```

```
+-----+
| Difference |
+-----+
|          14 |
+-----+
```

In this example, the `TO_DAYS()` function is employed to calculate the difference in the number of days between the two dates.

---

## UNIX\_TIMESTAMP()

`UNIX_TIMESTAMP([datetime])`

This function returns the number of seconds since the start of the Unix epoch (January 1, 1970, Greenwich Mean Time). Without a given time, this function will return the Unix time for the current date and time. Optionally, a date and time value (directly or by way of a column value) may be given for conversion to Unix time with this function.

```
SELECT UNIX_TIMESTAMP()
       AS 'Now',
       UNIX_TIMESTAMP('2004-05-09 20:45:00')
       AS 'Same Time';
```

Now	Same Time
1084153500	1084153500

The first column uses the function to determine the Unix time for the moment that the statement was entered. The second column uses the same function to determine the Unix time for the same date and time provided in a usual, readable format.

---

## UTC\_DATE()

UTC\_DATE()

This function returns the current Universal Time Clock (UTC) date in *yyyy-mm-dd* format. This function will return the UTC date in a *yyyymmdd* format if it's used as part of a numeric calculation (see example). You can use this in SELECT statements as shown here, in INSERT and UPDATE statements to set a value, or in a WHERE clause. This function is available as of Version 4.1.1 of MySQL. The pair of parentheses is optional.

```
SELECT UTC_DATE() + 1, UTC_DATE();
```

UTC_DATE() + 1	UTC_DATE()
20050112	2005-01-11

---

## UTC\_TIME()

UTC\_TIME()

This function returns the current UTC time in *hh:mm:ss* format. It will return the UTC time in an *hhmmss* format if it's used as part of a numeric calculation (see example). You can use this in SELECT statements as shown here, in INSERT and UPDATE statements to set a value, or in a WHERE clause. This statement is available as of Version 4.1.1 of MySQL. The pair of parentheses is not required.

```
SELECT UTC_TIME() + 1, UTC_TIME();
```

UTC_TIME() + 1	UTC_TIME()
195858	19:58:57

---

## UTC\_TIMESTAMP()

UTC\_TIMESTAMP()

This function returns the current UTC date and time in *yyyy-mm-dd hh:mm:ss* format. It will return the UTC date and time in a *yyyymmddhhmmss* format if it's used as part of a numeric calculation (see example). You can use this in SELECT statements as shown

here, in INSERT and UPDATE statements to set a value, or in a WHERE clause. This statement is available as of Version 4.1.1 of MySQL. The pair of parentheses is not required. Use UTC\_TIME() for only the UTC time and UTC\_DATE() for only the UTC date.

```
SELECT UTC_TIMESTAMP() + 1, UTC_TIMESTAMP();
```

```
+-----+-----+
| UTC_TIMESTAMP() + 1 | UTC_TIMESTAMP() |
+-----+-----+
|      20050111200238 | 2005-01-11 20:02:37 |
+-----+-----+
```

---

## WEEK()

WEEK(*date* [, *value*])

This function returns the number of the week starting from the beginning of the year for a date provided. By default, Sunday is considered to be the first day of a week for this calculation. To set the first day to Monday, enter a value of 1 as a second argument to this function. This function returns 0–52 by default. However, when the start is set to Monday, the range is 1–53.

```
SELECT WEEK('2004-01-15') AS 'Sunday Start',
       WEEK('2004-01-15',1) AS 'Monday Start';
```

```
+-----+-----+
| Sunday Start | Monday Start |
+-----+-----+
|              2 |              3 |
+-----+-----+
```

Notice in the first column, that the first day of the week is the default of Sunday, so the date January 15, 2004 returns a value of 2. When the first day of the week is set to Monday, though, the value of 3, or the third week, is returned for the same date.

---

## WEEKDAY()

WEEKDAY(*date*)

This function returns the number for the day of the week. Monday is considered the first day of the week for this function and it returns a value of 0; Sunday returns 6.

```
SELECT WEEKDAY('2005-01-01')
       AS 'Saturday, Jan. 1';
```

```
+-----+
| Saturday, Jan. 1 |
+-----+
|                   5 |
+-----+
```

---

## WEEKOFYEAR()

WEEKOFYEAR(*date*)

This function returns the calendar week of the year for the given date. This function was added in Version 4.1.1 of MySQL.

```
SELECT CURDATE() AS Date,  
       WEEKOFYEAR(CURDATE()) AS Week;
```

```
+-----+-----+  
| Date       | Week |  
+-----+-----+  
| 2005-01-11 |    2 |  
+-----+-----+
```

---

## YEAR()

YEAR(*date*)

This function returns the year of the given date provided. It returns values from 1,000–9,999.

```
SELECT YEAR('2005-01-01')  
       AS 'Year';
```

```
+-----+  
| Year |  
+-----+  
| 2005 |  
+-----+
```

---

## YEARWEEK()

YEARWEEK(*date* [, *value*])

This function returns the year coupled with the number of the week into the year: *yyyymm*. By default, the first day of the week is Sunday and the basis of the calculation. Optionally, you can set this to Monday as the first day of the week by entering a value of 1 for the second argument.

```
SELECT YEARWEEK('2005-01-07')  
       AS 'YearWeek';
```

```
+-----+  
| YearWeek |  
+-----+  
| 200501 |  
+-----+
```

This function may be useful in conjunction with the PERIOD\_ADD() and PERIOD\_DIFF() functions.