MySQL DATETIME Data Type

Summary: in this tutorial, you will learn about MySQL `DATETIME` data type and how to use some handy functions for manipulating `DATETIME` effectively.

Introduction to MySQL DATETIME data type

MySQL `DATETIME` data type allows you to store a value that contains both date and time.

When you query data from a `DATETIME` column, MySQL displays the `DATETIME` value in the following format:

```
'YYYY-MM-DD HH:MM:SS'
```

When you insert a value into a `DATETIME` column, you use the same format. For example:

```
INSERT INTO table_name(datetime_column)
VALUES('2023-12-31 15:30:45');
```

To populate a column with the current date and time, you use the result of the `CURRENT_TIMESTAMP` or `NOW()` function as the default value. For example:

```
CREATE TABLE events(
    id INT AUTO_INCREMENT PRIMARY KEY,
    event_name VARCHAR(255) NOT NULL,
    started_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO events(event_name)
VALUES('Connected to MySQL Server');

SELECT * FROM events;
```

Output:

```
+----+---------------------------+---------------------+
| id | event_name                | started_at          |
+----+---------------------------+---------------------+
|  1 | Connected to MySQL Server | 2023-12-28 07:51:18 |
+----+---------------------------+---------------------+
1 row in set (0.00 sec)
```

By default, `DATETIME` values range from `1000-01-01 00:00:00` to `9999-12-31 23:59:59`. MySQL uses 5 bytes to store a `DATETIME` value.

In addition, a `DATETIME` value can include a trailing fractional second up to microseconds with the format `YYYY-MM-DD HH:MM:SS[.fraction]` e.g., `2015-12-20 10:01:00.999999`.

When including the fractional second precision, `DATETIME` values require more storage as illustrated in the following table:

Fractional Seconds Precision	**Storage (Bytes)**
0	0
1, 2	1
3, 4	2
5, 6	3

For example, `2015-12-20 10:01:00.999999` requires 8 bytes, 5 bytes for `2015-12-20 10:01:00` and 3 bytes for `.999999` while `2015-12-20 10:01:00.9` requires only 6 bytes, 1 byte for the fractional second precision.

MySQL DATETIME vs.TIMESTAMP

MySQL provides another temporal data type that is similar to the `DATETIME` called [`TIMESTAMP`].

The `TIMESTAMP` requires 4 bytes while `DATETIME` requires 5 bytes. Both `TIMESTAMP` and `DATETIME` require additional bytes for fractional seconds precision.

`TIMESTAMP` values range from `1970-01-01 00:00:01 UTC` to `2038-01-19 03:14:07 UTC`. If you want to store temporal values that are beyond 2038, you should use `DATETIME` instead of `TIMESTAMP`.

MySQL stores `TIMESTAMP` in UTC value. However, MySQL stores the `DATETIME` value as is without timezone. Let's see the following example.

First, set the timezone of the current connection to `+00:00`.

```sql
SET time_zone = '+00:00';
```

Next, create a table named `timestamp_n_datetime` that consists of two columns: `ts` and `dt` with `TIMESTAMP` and `DATETIME` types using the following statement.

```sql
CREATE TABLE timestamp_n_datetime (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ts TIMESTAMP,
    dt DATETIME
);
```

Then, insert the current date and time into both `ts` and `dt` columns of the `timestamp_n_datetime` table,

```sql
INSERT INTO timestamp_n_datetime(ts,dt)
VALUES(NOW(),NOW());
```

After that, query data from the `timestamp_n_datetime` table.

```
SELECT
    ts,
    dt
FROM
    timestamp_n_datetime;
```

Both values in `DATETIME` and `TIMESTAMP` columns are the same.

Finally, set the connection's time zone to `+03:00` and query data from the `timestamp_n_datetime` table again.

```
SET time_zone = '+03:00';

SELECT
    ts,
    dt
FROM
    timestamp_n_datetime;
```

The output indicates that the value in the `TIMESTAMP` column is different. This is because the `TIMESTAMP` column stores the date and time value in UTC when we change the time zone, the value of the `TIMESTAMP` column is adjusted according to the new time zone.

It means that if you use the `TIMESTAMP` data to store date and time values, you should take serious consideration when you move your database to a server located in a different time zone.

MySQL DATETIME functions

The following statement sets the variable `@dt` to the current date and time using the `NOW()` function.

```
SET @dt =  NOW();
```

To query the value of the `@dt` variable, you use the following `SELECT` statement:

```
SELECT @dt;
```

MySQL DATE() function

To extract the date portion from a `DATETIME` value, you use the `DATE` function as follows:

```
SELECT DATE(@dt);
```

This function is very useful in case you want to query data based on a date but the data stored in the column is based on both date and time.

Let’s see the following example.

```
CREATE TABLE test_dt (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME
);

INSERT INTO test_dt(created_at)
VALUES('2015-11-05 14:29:36');
```

Suppose you want to know which row created on `2015-11-05`, you use the following query:

```
SELECT
    *
FROM
    test_dt
WHERE
    created_at = '2015-11-05';
```

It returns no rows.

This is because the `created_at` column contains not only the date but also the time. To correct it, you use the `DATE` function as follows:

```
SELECT
    *
FROM
    test_dt
WHERE
    DATE(created_at) = '2015-11-05';
```

It returns one row as expected. In case the table has many rows, MySQL has to perform a full table scan to locate the rows that match the condition.

MySQL TIME function

To extract the time portion from a `DATETIME` value, you use the `TIME` function as the following statement:

```
SELECT TIME(@dt);
```

MySQL YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE and SECOND functions

To get the year, quarter, month, week, day, hour, minute, and second from a `DATETIME` value, you use the functions as shown in the following statement:

```
SELECT
    HOUR(@dt),
    MINUTE(@dt),
    SECOND(@dt),
    DAY(@dt),
    WEEK(@dt),
    MONTH(@dt),
    QUARTER(@dt),
    YEAR(@dt);
```

MySQL DATE_FORMAT function

To format a `DATETIME` value, you use the `DATE_FORMAT` function. For example, the following statement formats a `DATETIME` value based on the `%H:%i:%s - %W %M %Y` format:

```
SELECT DATE_FORMAT(@dt, '%H:%i:%s - %W %M %Y');
```

MySQL DATE_ADD function

To add an interval to a `DATETIME` value, you use `DATE_ADD` function as follows:

```
SELECT @dt start,
       DATE_ADD(@dt, INTERVAL 1 SECOND) '1 second later',
       DATE_ADD(@dt, INTERVAL 1 MINUTE) '1 minute later',
       DATE_ADD(@dt, INTERVAL 1 HOUR) '1 hour later',
       DATE_ADD(@dt, INTERVAL 1 DAY) '1 day later',
       DATE_ADD(@dt, INTERVAL 1 WEEK) '1 week later',
       DATE_ADD(@dt, INTERVAL 1 MONTH) '1 month later',
       DATE_ADD(@dt, INTERVAL 1 YEAR) '1 year later';
```

MySQL DATE_SUB function

To subtract an interval from a `DATETIME` value, you use `DATE_SUB` function as follows:

```
SELECT @dt start,
       DATE_SUB(@dt, INTERVAL 1 SECOND) '1 second before',
       DATE_SUB(@dt, INTERVAL 1 MINUTE) '1 minute before',
       DATE_SUB(@dt, INTERVAL 1 HOUR) '1 hour before',
       DATE_SUB(@dt, INTERVAL 1 DAY) '1 day before',
       DATE_SUB(@dt, INTERVAL 1 WEEK) '1 week before',
       DATE_SUB(@dt, INTERVAL 1 MONTH) '1 month before',
       DATE_SUB(@dt, INTERVAL 1 YEAR) '1 year before';
```

MySQL DATE_DIFF function

To calculate a difference in days between two `DATETIME` values, you use the `DATEDIFF` function. Notice that the `DATEDIFF` function only considers the date part of a `DATETIME` value in the calculation.

See the following example.

First, create a table named `datediff_test` that has one column whose data type is `DATETIME` .

```
CREATE TABLE datediff_test (
    dt DATETIME
);
```

Second, insert some rows into the `datediff_test` table.

```
INSERT INTO datediff_test(dt)
VALUES('2010-04-30 07:27:39'),
        ('2010-05-17 22:52:21'),
        ('2010-05-18 01:19:10'),
        ('2010-05-22 14:17:16'),
        ('2010-05-26 03:26:56'),
        ('2010-06-10 04:44:38'),
        ('2010-06-13 13:55:53');
```

Third, use the `DATEDIFF` function to compare the current date and time with the value in each row of the `datediff_test` table.

```
SELECT
    dt,
    DATEDIFF(NOW(), dt)
FROM
```

```
    datediff_test;
```

In this tutorial, you have learned about MySQL `DATETIME` data type and some useful `DATETIME` functions.

Was this tutorial helpful?

PREVIOUSLY
MySQL DECIMAL Data Type

UP NEXT
MySQL TIMESTAMP Data Type

Search ...

ON DELETE CASCADE

TRUNCATE TABLE

MYSQL TRANSACTIONS

Table Locking

MYSQL DATA TYPES

BIT

INT

BOOLEAN

DECIMAL

DATETIME

TIMESTAMP

DATE

TIME

CHAR

VARCHAR

TEXT

BINARY

VARBINARY

ENUM

BLOB

MYSQL GLOBALIZATION

MySQL Character Sets

MySQL Collation

MYSQL IMPORT & EXPORT

Import a CSV File Into a Table

Export a Table to a CSV File

ADVERTISEMENTS

ABOUT MYSQL TUTORIAL WEBSITE

MySQLTutorial.org helps you master MySQL quickly, easily, and with enjoyment. Our tutorials make learning MySQL a breeze.

All MySQL tutorials are clear, practical and easy-to-follow. More About Us

LATEST TUTORIALS

MySQL Port

MySQL Commands

innodb_dedicated_server: Configure InnoDB Dedicated Server

innodb_flush_method: Configure InnoDB Flush Method

innodb_log_buffer_size: Configure InnoDB Log Buffer Size

innodb_buffer_pool_chunk_size: Configure Buffer Pool Chunk Size

innodb_buffer_pool_instances: Configuring Multiple Buffer Pool Instances for Improved Concurrency in MySQL

innodb_buffer_pool_size: Configure InnoDB Buffer Pool Size

MySQL InnoDB Architecture

How to Kill a Process in MySQL

SITE LINKS

Donation ❤️

Contact Us

About

Privacy Policy

OTHERS

MySQL Cheat Sheet

MySQL Resources

MySQL Books