MySQL TIMESTAMP Data Type

Summary: in this tutorial, you will learn about MySQL `TIMESTAMP` data type and how to use it to define columns that store timestamp data.

Introduction to MySQL TIMESTAMP data type

The MySQL `TIMESTAMP` is a temporal data type that holds the combination of date and time. The format of a `TIMESTAMP` is `YYYY-MM-DD HH:MM:SS` which is fixed at 19 characters.

The `TIMESTAMP` value has a range from `'1970-01-01 00:00:01'` UTC to `'2038-01-19 03:14:07'` UTC.

When you insert a `TIMESTAMP` value into a table, MySQL converts it from your connection's time zone to UTC for storing.

When you query a `TIMESTAMP` value, MySQL converts the UTC value back to your connection's time zone. This conversion does not occur for other temporal data types, such as `DATETIME`.

By default, the connection time zone is the MySQL Server's time zone. You also have the option to use a different time zone when connecting to the MySQL Server.

When you retrieve a `TIMESTAMP` value that was inserted by a client in a different time zone, you

will receive a value different from the one stored in the database.

However, as long as you don't change the time zone, you can retrieve the originally stored `TIMESTAMP` value.

MySQL TIMESTAMP time zone example

Let's take an example to see how MySQL handles `TIMESTAMP` values.

First, created a new table called `t` that has a `TIMESTAMP` column: `t1`;

```
CREATE TABLE t (
   ts TIMESTAMP
);
```

Second, set the session's time zone to `'+00:00'` UTC by using the `SET time_zone` statement.

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

Third, insert a `TIMESTAMP` value into the `t` table.

```
INSERT INTO t(ts)
VALUES('2008-01-01 00:00:01');
```

Fourth, select the `TIMESTAMP` value from the `t` table.

```
SELECT ts FROM t;
```

Output:

```
+---------------------+
| ts                  |
+---------------------+
| 2008-01-01 00:00:01 |
+---------------------+
1 row in set (0.00 sec)
```

Fifth, set the session's time zone to a different time zone to observe what value we receive from the database server:

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

Finally, query data from the table:

```
SELECT ts FROM t;
```

Output:

```
+---------------------+
| ts                  |
+---------------------+
| 2008-01-01 03:00:01 |
+---------------------+
1 row in set (0.00 sec)
```

As you can see, we received a time value adjusted to the new time zone, which is different from the original.

Automatic initialization and updating for TIMESTAMP columns

Consider the following example.

First, create a table named `categories`:

```
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

In the `categories` table, the `created_at` column is a `TIMESTAMP` column whose default value is set to `CURRENT_TIMESTAMP`.

Second, insert a new row into the `categories` table without specifying the value for the `created_at` column:

```
INSERT INTO categories(name)
VALUES ('A');
```

```
SELECT * FROM categories;
```

The output indicates that MySQL used the timestamp at the time of insertion as a default value for the `created_at` column.

So a `TIMESTAMP` column can be automatically initialized to the current timestamp for inserted rows that specify no value for the column. This feature is called **automatic initialization**.

Third, add a new column named `updated_at` to the `categories` table:

```
ALTER TABLE categories
ADD COLUMN updated_at
  TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  ON UPDATE CURRENT_TIMESTAMP;
```

The default value of the `updated_at` column is `CURRENT_TIMESTAMP`.

And, we have a new clause `ON UPDATE CURRENT_TIMESTAMP` that follows the `DEFAULT CURRENT_TIMESTAMP` clause. Let's see its effect.

Fourth, insert a new row into the `categories` table.

```
INSERT INTO categories(name)
VALUES('B');
```

Fifth, query data from the `categories` table:

```
SELECT * FROM categories;
```

The default value of the column `created_at` is the timestamp when the row was inserted.

Sixth, update the value in the column `name` of the row id 2:

```
UPDATE categories
SET name = 'B+'
WHERE id = 2;
```

Seventh, query data from the `categories` table to check the update:

```
SELECT *
FROM categories
WHERE id = 2;
```

Notice that the value in the `updated_at` column changed to the timestamp at the time the row was updated.

The ability of a `TIMESTAMP` column to be automatically updated to the current timestamp when the value in any other column in the row changes from its current value is called **automatic updating**.

The column `updated_at` is referred to as an auto-updated column.

Note that if you execute the `UPDATE` statement to update the same value for the `name` column, the `updated_at` column will not be updated.

```
UPDATE categories
SET name = 'B+'
WHERE id = 2;
```

The value in the `updated_at` remains unchanged.

For more information on automatic initialization and updating, please check out the time initialization.

> As of MySQL 5.6.5, the `DATETIME` columns also have automatic initialization and updating features. In addition, the `DEFAULT_CURRENT_TIMESTAMP` and `ON UPDATE CURRENT TIMESTAMP` can be applied to multiple columns.

Summary

- Use the MySQL `TIMESTAMP` data type to represent date and time values.
- Set the `DEFAULT CURRENT_TIMESTAMP` attribute for a `TIMESTAMP` column to automatically initialize the column with the current timestamp when a new row is inserted.
- Set the `ON UPDATE CURRENT_TIMESTAMP` attribute to update the timestamp whenever the row is modified.
- MySQL stores the `TIMESTAMP` values in UTC format but converts them to the current session timezone when displayed.

PREVIOUSLY
MySQL DATETIME Data Type

UP NEXT
MySQL DATE Data Type

Search ...

GETTING STARTED

What Is MySQL?

Install MySQL Database Server

Connect to MySQL Server

Download MySQL Sample Database

Load Sample Database

QUERYING DATA

SELECT FROM

SELECT

ORDER BY

WHERE

SELECT DISTINCT

AND

OR

IN

NOT IN

BETWEEN

LIKE

LIMIT

MANAGING DATABASES

MANAGING TABLES

AUTO_INCREMENT

Rename Tables

Add Columns

Drop Columns

Drop Tables

Temporary Tables

Generated Columns

MYSQL CONSTRAINTS

Primary Key

Foreign Key

Disable Foreign Key Checks

UNIQUE Constraint

NOT NULL Constraint

DEFAULT Constraint

CHECK Constraint

INSERT DATA

Insert Into

Insert Multiple Rows

INSERT INTO SELECT

Insert On Duplicate Key Update

INSERT IGNORE

Insert DateTimes

Insert Dates

UPDATE DATA

UPDATE

UPDATE JOIN

DELETE DATA

DELETE JOIN

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

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

SITE LINKS

Donation ❤️

Contact Us

About

Privacy Policy

OTHERS

MySQL Cheat Sheet

MySQL Resources

MySQL Books

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