# How to Insert Datetime Values into MySQL Tables

Authored by

**mohammed looti**

January 6, 2026

Inserting date and time data efficiently is fundamental for managing modern relational databases. In MySQL, the process involves utilizing the INSERT command, which is the standard SQL statement used to add records into a specified table and column. This operation ensures accurate record-keeping of events, timestamps, or creation dates.

A common scenario is automatically logging the exact time a record was created. For instance, if you have a table named "users" and a column designated for tracking creation time, typically named "created_at," you can easily populate this field with the current server date and time. This is achieved by using the built-in NOW() function directly within your insertion query.

The query to accomplish this task is concise and powerful. When executed, the following statement automatically inserts the current DATETIME value into the "created_at" column for every new entry added to the "users" table, simplifying timestamp management:

INSERT INTO users (created_at) VALUES (NOW());

## Understanding the MySQL DATETIME Data Type

To store combined date and time information within a MySQL table, you must define the column using the **DATETIME** data type. This type is capable of representing both date and time values, ranging from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Defining a column as **DATETIME** is the first essential step before attempting any insertion.

A critical consideration when manually inserting datetime values is strict adherence to the required format. Unlike some other database systems, MySQL mandates that literal datetime strings conform precisely to the standard ISO 8601-like structure. Failure to follow this format will result in an error, halting the execution of your INSERT statement.

## Required Datetime Format Specification

When supplying a string literal for insertion into a **DATETIME** column, the value must be enclosed in single quotes and structured exactly as follows:

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

Each component of this format plays a specific role in defining the precise point in time being recorded. Understanding the structure ensures data integrity and successful insertion into your database:

**YYYY**: Represents the year using exactly four digits.

**MM**: Represents the month (01 through 12) using two digits.
**DD**: Represents the day of the month (01 through 31) using two digits.
**HH**: Represents the hour of the day (00 through 23) using two digits (24-hour clock format).
**MM**: Represents the minutes (00 through 59) using two digits.
**SS**: Represents the seconds (00 through 59) using two digits.

The following example shows how to insert a datetime column into a table in MySQL in practice.

## Practical Example: Creating a Table with DATETIME

To illustrate the insertion of date and time values, we will construct a hypothetical table named `sales`. This table will be designed to track grocery sales, requiring a column that accurately records the precise moment each transaction occurred. We designate the `sales_time` column using the **DATETIME** data type to ensure complete temporal accuracy.

The following SQL syntax first defines the table structure, including a primary key for the store ID, a text field for the item sold, and the non-nullable `sales_time` column. Subsequently, we use multiple INSERT commands to populate the table with sample data, strictly adhering to the 'YYYY-MM-DD HH:MM:SS' format for the timestamp entries.

```
-- create table
CREATE TABLE sales (
store_ID INT PRIMARY KEY,
item TEXT NOT NULL,
sales_time DATETIME NOT NULL
);

-- insert rows into table
INSERT INTO sales VALUES (0001, 'Oranges', '2015-01-12 03:45:00');
INSERT INTO sales VALUES (0002, 'Apples', '2020-11-25 15:25:01');
INSERT INTO sales VALUES (0003, 'Bananas', '2009-06-30 09:01:39');
INSERT INTO sales VALUES (0004, 'Melons', '2022-04-09 03:29:55');
INSERT INTO sales VALUES (0005, 'Grapes', '2023-05-19 23:10:04');

-- view all rows in table
SELECT * FROM sales;
```

## Reviewing the Successful Insertion Output

After running the creation and insertion queries, we execute a simple `SELECT * FROM sales;` query to view the contents of the table. The output confirms that all five records were successfully

inserted, and the DATETIME values are stored correctly within the `sales_time` column, exactly matching the specified 'YYYY-MM-DD HH:MM:SS' structure.

**Output:**

```
+----------+---------+---------------------+
| store_ID | item | sales_time |
+----------+---------+---------------------+
| 1 | Oranges | 2015-01-12 03:45:00 |
| 2 | Apples | 2020-11-25 15:25:01 |
| 3 | Bananas | 2009-06-30 09:01:39 |
| 4 | Melons | 2022-04-09 03:29:55 |
| 5 | Grapes | 2023-05-19 23:10:04 |
+----------+---------+---------------------+
```

It is important to notice how the database handles the `sales_time` column. Since it was defined as a DATETIME type, every entry is normalized to this standardized format, validating the requirement for strict input formatting during the INSERT operation.

## Handling Insertion Failures: The Importance of Format Consistency

A frequent issue encountered by developers when dealing with temporal data is format mismatch. If you attempt to use the INSERT statement while supplying a date string that deviates from the expected 'YYYY-MM-DD HH:MM:SS' format (for example, using the U.S. common 'MM/DD/YYYY' format), the MySQL server will reject the operation outright.

Consider the attempt to insert a new sale record for 'Pears' using '5/18/2023 05:56:00'. The database engine cannot implicitly interpret this format, leading to a critical error that prevents the row from being added to the table. This confirms that manual parsing or conversion is necessary if the input string does not match the standard internal format.

## Demonstration of Format Error

The following query demonstrates what happens when a non-standard datetime string is provided:

```
-- attempt to insert row using incorrect format
INSERT INTO sales VALUES (0006, 'Pears', '5/18/2023 05:56:00');

-- view all rows in table (will only show the original 5)
SELECT * FROM sales;
```

**Output:**

**ERROR 1292 (22007): Incorrect datetime value: '5/18/2023 05:56:00' for column 'sales_time' at row 1**

## Overcoming Format Limitations Using STR_TO_DATE

While strict formatting is enforced for literal string inputs, MySQL provides robust functions to handle dates presented in non-standard formats. The **STR_TO_DATE** function is specifically designed to parse a string based on a specified format mask and convert it into a proper DATETIME value that the database can accept. This function is invaluable when dealing with legacy data or data imported from external sources where the timestamp format is inconsistent.

The **STR_TO_DATE** function requires two arguments: the date string to be converted, and a format string that describes the exact structure of the input string. For the previous error case, where the input was '5/18/2023 05:56:00' (MM/DD/YYYY HH:MM:SS), the corresponding format mask would be `'%m/%d/%Y %H:%i:%s'`. Utilizing this function allows us to successfully execute the INSERT command.

## Successful Insertion Using STR_TO_DATE

By incorporating **STR_TO_DATE** into our query, we successfully convert the ambiguously formatted date string into a valid DATETIME object before it reaches the `sales_time` column:

```
-- insert row into table using STR_TO_DATE for conversion
INSERT INTO sales VALUES (0006, 'Pears', STR_TO_DATE('5/18/2023 05:56:00', '%m/%d/%Y %H:%i:%s'));

-- view all rows in table
SELECT * FROM sales;
```

**Output:**

```
+----------+---------+---------------------+
| store_ID | item | sales_time |
+----------+---------+---------------------+
| 1 | Oranges | 2015-01-12 03:45:00 |
| 2 | Apples | 2020-11-25 15:25:01 |
| 3 | Bananas | 2009-06-30 09:01:39 |
| 4 | Melons | 2022-04-09 03:29:55 |
```

**| 5 | Grapes | 2023-05-19 23:10:04 |**
**| 6 | Pears | 2023-05-18 05:56:00 |**
**+----------+---------+---------------------+**

By using **STR_TO_DATE**, we are able to insert this new datetime value without any errors, ensuring the database stores the record in the proper format.

## Summary of Best Practices for Datetime Insertion

Inserting datetime values into MySQL is a straightforward process, provided you maintain strict adherence to two key principles. First, always define your column using the appropriate temporal data type, such as DATETIME. Second, ensure that any literal string inputs are formatted according to the 'YYYY-MM-DD HH:MM:SS' standard.

For scenarios requiring the current moment, leveraging the **NOW()** function provides an automated, precise method. Conversely, if you must work with inputs originating in inconsistent formats, the powerful **STR_TO_DATE** function offers the flexibility needed for successful data conversion and insertion, thereby ensuring your data remains clean and usable.

## Related MySQL Tutorials

Mastering temporal data management is just one aspect of effective database administration. The following resources explain how to perform other common tasks in MySQL: