How to Extract Dates from Datetime Values in MySQL

Authored by
mohammed looti

January 6, 2026

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

mohammed looti (2026). *How to Extract Dates from Datetime Values in MySQL*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=124701

Truncating date in MySQL refers to removing the time component from a date value. This can be done using the DATE() function, which extracts the date part from a given date. For example, if we have a date value of "2021-03-17 14:30:00", using the DATE() function will return "2021-03-17" as the truncated date. This is useful when working with date data and only needing the date portion for analysis or comparison purposes.

Understanding Date Truncation in MySQL

Date truncation is a critical operation within any relational database management system, especially when handling columns defined using the DATETIME or TIMESTAMP data types. These composite data types store both calendar information (year, month, day) and time information (hours, minutes, seconds, and sometimes milliseconds). However, for many analytical and reporting tasks, the granular time data is extraneous and can complicate query logic or group aggregation.

The essence of date truncation is standardization. When you apply the DATE() function in MySQL, you are instructing the database engine to perform an explicit conversion, discarding the time component entirely. This operation transforms a potentially long, complex value representing a moment in time into a concise, standardized date string (YYYY-MM-DD format). This uniformity is key to ensuring that datasets can be reliably compared and grouped.

It is important to differentiate between truncating a date (removing the time portion) and formatting a date (changing how the date is displayed). While both manipulate the presentation of date-time data, truncation uses the specialized DATE() function specifically to isolate the date part. This makes the resulting output suitable for operations like JOINs, WHERE clauses, and GROUP BY statements where only the day matters, offering superior performance and clarity compared to manually parsing date strings.

Why Truncate Dates for Data Analysis?

The necessity for date truncation typically arises in scenarios where data needs to be aggregated or compared on a daily basis. Without truncation, two records created on the same calendar day but at different times (e.g., 9:00 AM and 3:00 PM) would be treated as distinct values when grouped. This leads to broken aggregation, rendering daily summaries useless. By standardizing these values using the DATE() function, we ensure that all events belonging to a specific day are correctly linked.

Furthermore, truncated dates improve query performance, particularly in systems handling vast amounts of time-series data. Indexing on a combined DATETIME column is useful for range queries, but if you frequently query by date alone, calculating the date portion on the fly with

ARABPSYCHOLOGY.COM

DATE(column) allows for simplified, indexed lookups on the resulting date values. For developers, using the built-in function simplifies the SQL logic, making the intent of the query--to look at daily figures--immediately obvious.

Truncation is also vital for data integration and reporting. When combining data from multiple sources that might use slightly different timestamp formats or granularities, converting all timestamp fields to a common date format via truncation provides a necessary common denominator. This ensures that business intelligence reports, dashboards, and automated summaries are based on consistent daily metrics, eliminating discrepancies caused by minute-level time variations.

The Core Syntax: Implementing DATE() in MySQL

The standard way to apply date truncation in MySQL is by utilizing the DATE() function directly within your SELECT statement. The syntax is straightforward: wrap the name of the column containing the date-time value inside the parentheses of the function.

SELECT store_ID, item, DATE(sales_time) FROM sales;

This particular example uses the SELECT statement to retrieve three pieces of information. It selects the primary identifier column, **store_ID**, the textual field **item**, and most importantly, the date-only portion of the DATETIME column named **sales_time**. This data is retrieved from the table named **sales**.

By employing the DATE() function, we guarantee that the output for the **sales_time** column will exclude all time components, returning only the calendar date. This is the simplest and most explicit method for date truncation in MySQL, applicable to any column defined as `DATETIME`, `TIMESTAMP`, or even certain string representations of dates.

Practical Demonstration: Setting Up the Sales Table

To illustrate the functionality of date truncation, let's work through a practical example involving a grocery store sales ledger. We will begin by creating a simple table structure capable of holding sales transaction data, including a unique identifier, the item sold, and the precise moment of the sale stored as a DATETIME value. This setup provides the perfect environment for testing the DATE() function.

Suppose we define the table named **sales**. This table will house information about various grocery transactions, showing different items sold at different times across the store's operating history. Notice how the **sales_time** column uses the DATETIME data type, ensuring both date and time are stored initially.

ARABPSYCHOLOGY.COM

```
-- 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;
```

When we execute the final SELECT statement to view all rows, we confirm that the **sales_time** column holds the full timestamp, including hours, minutes, and seconds. This output confirms the starting state of our data before truncation is applied, clearly showing the time component we intend to remove.

Output Before Truncation:

```
+----------+---------+---------------------+
| 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 |
+----------+---------+---------------------+
```

Executing the Truncation Query and Analyzing Results

Now that our data is prepared, we can execute the primary query designed to truncate the time component of the **sales_time** column. This process demonstrates the power of the DATE() function in action, instantly simplifying the date representation for analytical purposes. We will retrieve the transaction ID, the item name, and the truncated date.

ARABPSYCHOLOGY.COM

We use the following syntax to select the **store_ID** column, **item** column and only the date portion of the **sales_time** column:

SELECT store_ID, item, DATE(sales_time) FROM sales;

Upon execution, the MySQL engine processes the values in the **sales_time** column, stripping away all time information. The resulting output clearly shows the transformation, confirming that the function successfully performed the desired truncation. This result set is now suitable for grouping data by day, as all time stamps have been normalized to their corresponding calendar date.

Output After Truncation:

```
+----------+---------+------------------+
| store_ID | item | DATE(sales_time) |
+----------+---------+------------------+
| 1 | Oranges | 2015-01-12 |
| 2 | Apples | 2020-11-25 |
| 3 | Bananas | 2009-06-30 |
| 4 | Melons | 2022-04-09 |
| 5 | Grapes | 2023-05-19 |
+----------+---------+------------------+
```

Notice that only the date portion of the **sales_time** column is returned in the output, with the time portion successfully truncated. The column header, however, defaults to the functional call: **DATE(sales_time)**. While accurate, this name is often cumbersome for use in application layers or complex nested queries, leading us to the next important step: aliasing.

Improving Readability with Column Aliases (The AS Clause)

As demonstrated in the previous result set, when applying a function like DATE(), the resulting column header defaults to the function call itself (e.g., `DATE(sales_time)`). This is usually not ideal for reporting or integration purposes. To provide a clear, meaningful label for the output, we utilize the AS clause, which allows us to assign a custom alias to the computed column.

The AS clause greatly enhances the readability of the query result, making the data easier to consume by end-users or dependent applications. By assigning an alias such as `sales_date` to the truncated date output, we clearly communicate the purpose of the column, transitioning from a functional description to a semantic description.

If you'd like, you can use the **AS** keyword to assign an alias to the resulting truncated date column:

ARABPSYCHOLOGY.COM

SELECT store_ID, item, DATE(sales_time) AS sales_date FROM sales;

Reviewing the revised output confirms the improvement. The name of the truncated date column is now **sales_date**, which is much easier to read and utilize than the functional name **DATE(sales_time)** from the previous example. This simple naming convention change is a best practice in SQL development.

```
+----------+---------+------------+
| store_ID | item | sales_date |
+----------+---------+------------+
| 1 | Oranges | 2015-01-12 |
| 2 | Apples | 2020-11-25 |
| 3 | Bananas | 2009-06-30 |
| 4 | Melons | 2022-04-09 |
| 5 | Grapes | 2023-05-19 |
+----------+---------+------------+
```

Alternative Methods for Date Manipulation

While the DATE() function is the dedicated tool for simple date truncation (removing all time components), MySQL offers other functions for broader date and time manipulation, often depending on the specific truncation level required (e.g., truncating to the start of the month or year).

One highly versatile function is DATE_FORMAT(). Unlike `DATE()`, which always returns the date part in YYYY-MM-DD format, `DATE_FORMAT()` allows you to define a specific output structure. If you format a DATETIME value using a format string that only includes date components (e.g., `%Y-%m-%d`), it achieves a similar truncation effect, though `DATE()` is usually faster and clearer for the standard date-only extraction.

For more advanced truncation, such as rounding or truncating to the start of a specific period (like the start of the quarter or year), MySQL typically relies on combinations of functions like `DATE_ADD` and `DATE_SUB` with `DATEDIFF` or `LAST_DAY` to manually calculate the period start. However, for the simple, straightforward task of removing time stamps to get the calendar date, the DATE() function remains the definitive choice.

Common Pitfalls and Performance Considerations

When implementing date truncation, database professionals must be aware of potential pitfalls, particularly concerning data types and indexing. Although the DATE() function is effective, applying

ARABPSYCHOLOGY.COM

it to a large dataset can impact performance because it forces a computation on every row, potentially preventing the efficient use of indexes defined on the original DATETIME column.

If daily aggregation is a frequent operation, consider the following strategies to mitigate performance issues:

Functional Indexes: In modern MySQL versions (8.0+), you can create an index on the expression `DATE(sales_time)`. This ensures that lookups based on the truncated date are fast and efficient, allowing the query optimizer to skip costly full table scans when filtering by date.

Materialized Columns: Alternatively, you could create a generated column (a materialized or virtual column) that stores the truncated date value permanently in the table definition. This pre-calculated column can then be indexed, providing optimal performance for daily queries without re-calculating the date extraction every time.

Data Type Consistency: Ensure the column you are truncating is indeed a valid date-time type (DATETIME or TIMESTAMP). Applying `DATE()` to an incorrectly formatted string column may result in NULL values or unexpected errors, emphasizing the importance of strong data typing in a database management system.

Conclusion and Further Reading

Mastering date truncation using the DATE() function is a foundational skill for anyone working with time-series data in MySQL. This function provides a clean, reliable, and standardized way to extract the calendar date from a comprehensive timestamp, which is essential for accurate reporting, data aggregation, and simplifying complex SQL queries.

By understanding not only the basic syntax but also the implications for readability (using AS aliases) and performance (using functional indexes), you can write more efficient and maintainable database code. Always prioritize the simplest tool for the job; when the goal is stripping the time, the DATE() function is unmatched.

The following resources explain how to perform other common tasks in MySQL:

MySQL: How to Truncate Strings
MySQL: How to Delete Rows from Table Based on id
MySQL: How to Delete Duplicate Rows But Keep Latest

ARABPSYCHOLOGY.COM