How to Get the Last Day of the Previous Month in MySQL

Authored by

mohammed looti

January 5, 2026

The challenge of handling temporal data in **MySQL** often requires precise calculations, such as determining specific boundaries like the start or end of a month. A highly efficient and common requirement is retrieving the last day of the month immediately preceding the current date. Achieving this requires chaining together specialized date and time functions provided by **MySQL**.

The standard **SELECT statement** necessary to accomplish this task is straightforward, combining three powerful native functions: `CURDATE()`, `DATE_SUB()`, and `LAST_DAY()`. The complete query, which returns the last calendar day of the month that came before the present month, is:

```
SELECT LAST_DAY(DATE_SUB(CURDATE(), INTERVAL 1 MONTH));
```

This single line of code is robust and reliable for virtually all **MySQL** environments, ensuring that time zone and leap year considerations are handled automatically by the internal date calculation engine.

Deconstructing the Key MySQL Date Functions

To fully appreciate the elegance of the solution, it is essential to understand the roles played by each function within the nested query structure. This specific query relies on functions executing sequentially from the innermost bracket outward, starting with determining the reference point and ending with the final transformation.

The sequence begins with **CURDATE()**, which simply returns the current date as a DATE value, ignoring the time component. This establishes the starting reference point for the calculation. For instance, if today is 2024-10-15, **CURDATE()** provides that exact value. If we want the last day of the previous month (September), we must first manipulate this starting date.

The resulting date from **CURDATE()** is then passed to the **DATE_SUB()** function. This function subtracts a specific time value from a given date. The syntax used here, `DATE_SUB(CURDATE(), INTERVAL 1 MONTH)`, instructs **MySQL** to subtract exactly one month from the current date. Continuing our example, subtracting one month from 2024-10-15 yields 2024-09-15. This step successfully shifts the timeline into the previous month.

Finally, the intermediate date is passed to the outermost function, **LAST_DAY()**. The primary role of **LAST_DAY()** is to take any date within a given month and return the date corresponding to the final day of that month. Since the input falls within September, **LAST_DAY()** recognizes this and returns 2024-09-30, which is the desired result. This nested approach ensures that the calculation is precise and handles varying month lengths correctly.

Integrating Date Logic with Table Columns

While calculating the last day of the previous month relative to the current server time is useful, database professionals often need to perform this calculation relative to a date stored within a specific column of a **database table**. This allows for dynamic analysis where each row's calculation is based on its own unique timestamp or date entry.

When applying this logic to a column, we simply replace the `CURDATE()` function with the name of the column containing the relevant date information, such as `sales_date`. The structure remains fundamentally the same: we take the column value, subtract the interval, and then find the last day of the resultant month. This methodology is crucial for tasks like monthly reporting or calculating billing cycles.

For example, if you have a table named `sales`, and you wish to generate a derived column that shows the last day of the month preceding the date recorded in `sales_date`, the syntax is adjusted slightly to reference the necessary columns. This transformation is highly common in data warehousing and business intelligence applications where historical date context is vital for analysis.

The revised query structure used to generate this derived field is shown below, demonstrating how to retrieve the last day of the previous month for a given date in **MySQL**:

SELECT sales_date, LAST_DAY(sales_date - INTERVAL 1 MONTH)
FROM sales;

This particular example creates a new column that contains the last day of the previous month for the corresponding date in the **sales_date** column of the **table** named **sales**.

Practical Demonstration: Setting Up the Sales Table

To illustrate the utility of this query, we will use a sample database table designed to track sales transactions. This table, named `sales`, contains essential fields such as a store identifier, the item sold, and, most importantly, the date of the transaction (`sales_date`). The variety of dates inserted ensures that the calculation handles transitions across different months and years effectively.

The creation script below outlines the schema definition for the `sales` table, specifying data types and constraints. Pay close attention to the definition of the `sales_date` column, as this is the primary field upon which our subsequent calculations will operate.

The following example shows how to use this syntax in practice. Suppose we have the following table named **sales** that contains information about sales made at various grocery stores on various

dates:

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

-- insert rows into table
INSERT INTO sales VALUES (0001, 'Oranges', '2024-02-10');
INSERT INTO sales VALUES (0002, 'Apples', '2024-11-25');
INSERT INTO sales VALUES (0003, 'Bananas', '2024-06-30');
INSERT INTO sales VALUES (0004, 'Melons', '2024-01-14');
INSERT INTO sales VALUES (0005, 'Grapes', '2024-05-19');

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

Output of the Sales Table:

```
+----------+---------+------------+
| store_ID | item | sales_date |
+----------+---------+------------+
| 1 | Oranges | 2024-02-10 |
| 2 | Apples | 2024-11-25 |
| 3 | Bananas | 2024-06-30 |
| 4 | Melons | 2024-01-14 |
| 5 | Grapes | 2024-05-19 |
+----------+---------+------------+
```

Executing the Date Calculation on the Sales Data

Our objective is now to execute the core function chain on every record in the `sales` table. We want to retrieve the original `sales_date` and a newly calculated column representing the last day of the month immediately preceding that sale date. This operation demonstrates how `DATE_SUB` and `LAST_DAY` work together dynamically across diverse date inputs.

Suppose that we would like to select the dates from the **sales_date** column and create a new column that contains the last day of the previous month for each corresponding date in the

sales_date column. We can use the following syntax to do so:

```
SELECT sales_date, LAST_DAY(sales_date - INTERVAL 1 MONTH)
FROM sales;
```

Output:

```
+------------+----------------------------------------+
| sales_date | LAST_DAY(sales_date - INTERVAL 1 MONTH) |
+------------+----------------------------------------+
| 2024-02-10 | 2024-01-31 |
| 2024-11-25 | 2024-10-31 |
| 2024-06-30 | 2024-05-31 |
| 2024-01-14 | 2023-12-31 |
| 2024-05-19 | 2024-04-30 |
+------------+----------------------------------------+
```

Notice that the dates in the new column accurately represent the last day of the previous month for the corresponding date in the **sales_date** column. This includes the row where the subtraction crossed a year boundary (January 14th resulting in December 31st of the previous year).

Improving Output Clarity Using the AS Clause

While the previous output correctly calculated the desired date, the automatically generated column header, `LAST_DAY(sales_date - INTERVAL 1 MONTH)`, is verbose and difficult to use in subsequent queries or reports. To improve the readability and utility of the query result, standard SQL practice dictates the use of column aliases.

In **MySQL**, the **AS** keyword is used to assign a temporary, user-friendly name to a column derived from a function or calculation. By applying an alias such as `last_previous`, we make the results instantly more accessible and easier to interpret by end-users or reporting tools.

If you'd like, you can use the **AS** statement to give a specific name to this new column:

```
SELECT sales_date, LAST_DAY(sales_date - INTERVAL 1 MONTH) AS last_previous
FROM sales;
```

The resulting output clearly shows the benefit of aliasing:

```
+------------+---------------+
```

```
| sales_date | last_previous |
+------------+---------------+
| 2024-02-10 | 2024-01-31 |
| 2024-11-25 | 2024-10-31 |
| 2024-06-30 | 2024-05-31 |
| 2024-01-14 | 2023-12-31 |
| 2024-05-19 | 2024-04-30 |
+------------+---------------+
```

Notice that the new column is now named **last_previous**, which is much easier to read and integrate into reporting frameworks.

Detailed Logic: How the Calculation Handles Dates

It is important to understand that the date subtraction mechanism in **MySQL** is calendar-aware, meaning it operates based on months rather than a fixed number of days (like 30 or 31). This calendar awareness is critical for ensuring accuracy when dealing with dates near the end of the month or during leap years.

This formula works by first subtracting one month from the given date, then by finding the last day of that particular month. Let's take a specific record from our sample data, `2024-02-10`, and trace the execution path of the formula `LAST_DAY(sales_date - INTERVAL 1 MONTH)` step-by-step:

Initial Value: The calculation starts with the input date, which is **2024-02-10**.

Date Subtraction (`DATE_SUB` operation): The inner operation subtracts one month to get **2024-01-10** (January 10th, 2024).

Finding the Last Day (`LAST_DAY` function): The resulting date, 2024-01-10, is passed to the **LAST_DAY()** function. This returns the last day of that month, which is **2024-01-31**.

This precise, two-step method ensures accuracy even when the start date falls on a day that does not exist in the previous month (e.g., subtracting one month from March 31st yields February 28th or 29th, which then feeds correctly into `LAST_DAY()`).

Conclusion and Resources for Further Learning

Mastering **MySQL** date manipulation is essential for powerful reporting and data analysis. The formula `LAST_DAY(DATE_SUB(CURDATE(), INTERVAL 1 MONTH))` provides a concise, accurate, and scalable method for identifying the last day of the preceding month, whether working with the current system date or dynamic column values.

The ability to accurately extract specific temporal anchors, such as the last day of a month, is foundational for creating useful time-series reports, defining fiscal periods, or filtering data based on rolling monthly windows. Always remember to use column aliases (**AS**) to ensure the readability of your derived fields.

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

MySQL: How to Select Rows where Date is Equal to Today