Learning MySQL: Retrieving Data Based on Date Ranges – Selecting Records Older Than One Week

Authored by

Mohammed looti

November 12, 2025

Introduction to Dynamic Date Range Querying in MySQL

When managing large datasets, particularly those involving transactions or time-stamped events, the ability to filter records based on dynamic date ranges is essential. Whether you are performing routine data purging, generating reports on recent activity, or analyzing historical trends, mastering date calculations within MySQL is a core skill for any database professional. This guide focuses on a common requirement: retrieving all records where the associated date is older than a specified threshold, specifically seven days.

To achieve this kind of filtering, we must employ built-in SQL time functions that calculate a moving target date based on the moment the query is executed. This method ensures that the query remains relevant regardless of when it is run. We will leverage powerful functions and operators to dynamically subtract time periods from the current system time, providing a highly flexible solution for data management.

Understanding the Core Syntax for Date Filtering

The mechanism for selecting records older than a specific duration relies on calculating a precise cutoff point. We use the standard SELECT statement combined with the WHERE clause to define our condition. The key components that enable this dynamic calculation are the NOW() function and the INTERVAL operator.

The NOW() function retrieves the current date and time when the query is executed. By subtracting a defined time period using the INTERVAL keyword, we establish the cutoff point. For instance, `NOW() - INTERVAL 1 WEEK` calculates the date and time exactly seven days prior to the current moment. Any record with a date less than (meaning older than) this calculated threshold will be included in the result set.

The following fundamental syntax demonstrates how to return all rows from a table where the date in a date column is older than seven days. In this example, we target a table named **sales** and filter based on the **sales_date** column:

```
SELECT *
FROM sales
WHERE sales_date < NOW() - INTERVAL 1 WEEK;
```

This specific query efficiently selects all rows in the table named **sales** where the value in the **sales_date** column precedes the date seven days ago. This is a crucial technique for maintaining data relevance and ensuring optimal query performance, particularly in large-scale relational database systems.

Practical Implementation: Setting up the Sales Table

To illustrate this concept practically, let us establish a sample scenario involving a simple sales tracking system. We will create a table called `sales` that records transactions, including a unique store identifier, the item sold, and the exact time of the sale. This table structure is typical for time-series data where date filtering is frequently required.

Our `sales` table will contain three columns: `store_ID` (an integer primary key), `item` (text), and `sales_date` (utilizing the DATETIME data type to store both date and time information). Using DATETIME is critical here, as it allows for precise comparison against the output generated by the NOW() function, which also includes time components.

We populate the table with five sample records, featuring various dates ranging from several years ago to very recently. This diversity allows us to effectively demonstrate how the date-range filtering mechanism accurately isolates older records while ignoring recent ones. The setup SQL is as follows:

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

-- insert rows into table
INSERT INTO sales VALUES (0001, 'Oranges', '2024-02-10 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', '2024-01-14 03:29:55');
INSERT INTO sales VALUES (0005, 'Grapes', '2023-05-19 23:10:04');

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

Executing the final `SELECT * FROM sales;` command yields the complete, unfiltered dataset, which serves as our baseline for comparison:

```
+----------+---------+---------------------+
| store_ID | item | sales_date |
+----------+---------+---------------------+
| 1 | Oranges | 2024-02-10 03:45:00 |
```

| 2 | Apples | 2020-11-25 15:25:01 |
| 3 | Bananas | 2009-06-30 09:01:39 |
| 4 | Melons | 2024-01-14 03:29:55 |
| 5 | Grapes | 2023-05-19 23:10:04 |
+----------+---------+---------------------+

Executing the Query and Analyzing Results

We now proceed with the primary goal: selecting only those rows where the sale occurred more than seven days ago. For the purpose of this demonstration, we assume the query is executed on **2/12/2024**. Therefore, any record dated 2/5/2024 or earlier should be retained, while the record dated 2/10/2024 should be filtered out.

We apply the dynamic date filtering logic introduced earlier. This is the precise query we use:

```
SELECT *
FROM sales
WHERE sales_date < NOW() - INTERVAL 1 WEEK;
```

Upon executing this command, MySQL first calculates the cutoff date (seven days prior to 2/12/2024) and then evaluates the condition for every row. Only records whose `sales_date` is strictly older than that threshold are returned.

The resulting output confirms the successful filtering, excluding the most recent record (ID 1):

+----------+---------+---------------------+
| store_ID | item | sales_date |
+----------+---------+---------------------+
| 2 | Apples | 2020-11-25 15:25:01 |
| 3 | Bananas | 2009-06-30 09:01:39 |
| 4 | Melons | 2024-01-14 03:29:55 |
| 5 | Grapes | 2023-05-19 23:10:04 |
+----------+---------+---------------------+

Review of Date Filtering Logic

The resulting table clearly shows that the row for 'Oranges' (ID 1, dated 2024-02-10) was excluded. This is because 2024-02-10 is within the seven-day window relative to the execution date of **2/12/2024**. Conversely, all other sales dates--ranging from 2023-05-19 back to 2009-06-30--are

older than seven days and were successfully selected by the query.

This technique is invaluable for automating database maintenance scripts, such as archiving old log entries or deleting expired user sessions. By utilizing `NOW()` coupled with `INTERVAL`, you avoid hardcoding dates, ensuring your maintenance operations are always based on the current system time. The flexibility of the `INTERVAL` operator allows you to easily adjust the time period (e.g., months, hours, or years) to meet various business requirements.

Additional Resources

To continue advancing your skills in time-based data manipulation within MySQL, consider reviewing tutorials on related tasks:

MySQL: How to Select Rows where Date is Equal to Today
How to use `DATEDIFF()` for custom date calculations.
Filtering records that fall within a specific month or year.