Learning to Filter Data with TODAY() in Google Sheets QUERY

Authored by

Mohammed looti

October 31, 2025

The **TODAY() function** in Google Sheets is a cornerstone of dynamic spreadsheet management, providing the current date that automatically updates daily. When leveraged alongside the incredibly powerful **QUERY() function**, this combination allows users to build highly responsive data analysis systems. This synergy is essential for filtering datasets based on real-time chronological conditions, enabling the creation of dashboards and reports that refresh without requiring constant manual intervention, thereby streamlining numerous data management and reporting workflows.

Mastering this integration is critical for any advanced user of Google Sheets. This comprehensive guide will walk you through the precise process of embedding the **TODAY() function** within your **QUERY() statements**. Our primary focus will be on constructing dynamic criteria that filter data based on whether a date column falls before, exactly on, or after the current date. This technique forms the foundation for building robust and time-sensitive data analysis solutions.

To successfully filter records where a date variable precedes the current date, you must properly format the date within the query string. The following basic syntax demonstrates how to achieve this dynamic filtering capability in Google Sheets:

=QUERY(A1:C9,"select A, B, C where B < date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'", 1)

The subsequent sections will meticulously detail the components of this formula, providing practical, real-world examples to illustrate its application. By the conclusion of this tutorial, readers will possess the expertise required to seamlessly leverage the **TODAY() function** within complex queries, significantly enhancing the dynamism and overall utility of their spreadsheet reports.

Understanding the TODAY() Function

The **TODAY() function** stands out as one of the most straightforward yet profoundly useful date functions available within the Google Sheets environment. Functionally, it requires no arguments and performs the single task of returning the current date, refreshing automatically upon every sheet load and across calendar days. This inherent automatic update mechanism is precisely what makes it indispensable for constructing dynamic reports, dashboards, and automated lists that must perpetually reflect the most current chronological perspective without manual input.

Consider the practical application: entering `=TODAY()` into any cell immediately displays the system's current date. Crucially, if the spreadsheet is accessed the following day, the displayed date automatically advances. This self-updating characteristic is vital for any query operation that requires comparison against the present moment. Typical use cases include automatically identifying tasks that are past due, aggregating sales figures up to the current calendar day, or filtering upcoming events scheduled for future dates.

It is important to maintain a clear distinction between **TODAY()** and the related **NOW() function**. While **TODAY()** returns only the date component (e.g., 2024-05-15), **NOW()** returns both the current date and the current time (e.g., 2024-05-15 10:30:00). For the vast majority of date-based filtering operations within the **QUERY() function**, where the filtering criterion typically ignores the time component, **TODAY()** remains the most precise and unambiguous choice.

Integrating TODAY() with QUERY() for Dynamic Date Filtering

The **QUERY() function** in Google Sheets utilizes a declarative language syntax highly analogous to **SQL (Structured Query Language)**, providing exceptional flexibility for data manipulation and extraction. However, a specific requirement exists when handling date values within the `where` clause of a query: dates must be formatted strictly as `'YYYY-MM-DD'` text strings and must be explicitly identified using the `date ' '` wrapper. This rigorous formatting ensures that the **QUERY() function** correctly interprets the string as a chronological value for accurate comparison.

Since the output of TODAY() is a numerical date value, we must employ the **TEXT() function** to achieve the required text string conversion. The **TEXT() function** is crucial here because it allows us to transform the numerical date output into a user-defined text format. By nesting the functions as `TEXT(TODAY(), "yyyy-mm-dd")`, we successfully generate the precise `'YYYY-MM-DD'` string mandated by **QUERY()**, conforming precisely to the internationally recognized **ISO 8601** standard for date representation.

A detailed examination of the core query structure reveals the mechanism of concatenation: `"select A, B, C where B < date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'"`. The initial segment, `"select A, B, C where B < date '"`, establishes the query parameters and initiates the required `date '` wrapper. The ampersand operator (`&`) then seamlessly stitches this initial string to the dynamically formatted current date generated by the `TEXT(TODAY(),...)` component. Finally, `&"'"` appends the necessary closing single quote, completing the dynamic date criteria. This sophisticated concatenation process is the fundamental technique required for constructing flexible and powerful date filters that update automatically.

Example 1: Retrieving Rows Before the Current Date

Imagine a scenario involving the management of a high-volume sales dataset, wherein records track transactions from multiple locations across various dates. The practical requirement is to filter this extensive data to display only those sales events that definitively occurred before the current date. This filtering approach is invaluable for reviewing historical performance metrics, conducting retrospective data analysis, or identifying patterns in past operational cycles.

We will utilize the following representative dataset, which details total sales figures recorded by two

distinct stores on a sequence of dates. The objective is to apply a dynamic filter to this range:

	A	B	C	D
1	**Store**	**Date**	**Sales**	
2	A	1/17/22	34	
3	A	1/18/22	39	
4	A	1/19/22	31	
5	A	1/20/22	40	
6	B	1/17/22	23	
7	B	1/18/22	29	
8	B	1/19/22	28	
9	B	1/20/22	27	
10				
11				
12				
13				
14				
15				

For the execution of this demonstration, let us establish a hypothetical current date of **January 18, 2022**. Our goal is specifically to extract all corresponding records where the date recorded in column 'B' (labeled 'Date') strictly precedes this assumed current date. This operation provides a clear illustration of how to dynamically exclude all future or present data points, ensuring a focus solely on historical events.

To dynamically retrieve the rows where the value in the **Date** column is chronologically before the current date, we employ the less-than operator (<) within the query string. This dynamically constructs the filtering condition against the formatted date provided by the nested functions:

=QUERY(A1:C9,"select A, B, C where B < date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'", 1)

Upon execution, the **QUERY() function** processes the source data range `A1:C9`. The condition `where B < date 'YYYY-MM-DD'` rigorously ensures that only those rows where the date in column B is strictly earlier than the current date (as derived and formatted by the `TEXT(TODAY(),...)` segment) are successfully returned. The final parameter, `1`, correctly specifies that the first row of the selected range is to be treated as header information.

The resulting output, as demonstrated below, confirms the successful application of the dynamic filter:

A11 =QUERY(A1:C9,"select A, B, C where B < date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'",1)

	A	B	C	D	E	F	G
1	**Store**	**Date**	**Sales**				
2	A	1/17/22	34				
3	A	1/18/22	39				
4	A	1/19/22	31				
5	A	1/20/22	40				
6	B	1/17/22	23				
7	B	1/18/22	29				
8	B	1/19/22	28				
9	B	1/20/22	27				
10							
11	Store	Date	Sales				
12	A	1/17/22	34				
13	B	1/17/22	23				
14							
15							
16							
17							
18							

As anticipated, only the rows containing dates preceding January 18, 2022, are visible in the result set. This robustly filters out all sales data from the current date and future, providing an uncluttered, accurate view of historical sales performance. The key advantage here is that this dynamic filter updates automatically every day, ensuring the historical data view remains perpetually current without any user intervention.

Example 2: Retrieving Rows Equal To or After the Current Date

In contrast to reviewing historical data, organizational needs often require a focus on current operations or upcoming schedules. This scenario is paramount for tracking active tasks, monitoring future sales projections, or managing scheduled events yet to be initiated. By simply modifying the relational operator within the **QUERY() function**, we can effectively shift the focus from past events to present and future data points.

We will continue to reference the same structured dataset, which outlines the total sales figures achieved by two stores across different recorded dates:

	A	B	C	D
1	**Store**	**Date**	**Sales**	
2	A	1/17/22	34	
3	A	1/18/22	39	
4	A	1/19/22	31	
5	A	1/20/22	40	
6	B	1/17/22	23	
7	B	1/18/22	29	
8	B	1/19/22	28	
9	B	1/20/22	27	
10				
11				
12				
13				
14				
15				

Once more, we will assume the current operating date is **January 18, 2022**. In this modified example, our explicit objective is to retrieve every row where the date contained in the 'Date' column is either precisely the current date or any subsequent date in the future. This proactive filtering capability is essential for operations that rely on immediate tracking and forward planning based on current and forthcoming data.

To successfully execute this future-focused filter, we must alter the comparison operator from the previous example's `<` (less than) to `>=` (greater than or equal to). This minor adjustment fundamentally changes the query's outcome, ensuring that it encompasses the current date along with all subsequent chronological data points. The foundational formula structure--relying on the **TEXT() function** to format the output of TODAY() for the **QUERY() function**--remains consistent.

=QUERY(A1:C9,"select A, B, C where B >= date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'", 1)

The following screenshot visually confirms the results achieved by applying this revised formula in practice, demonstrating the efficacy of the greater-than-or-equal-to operator:

A11 =QUERY(A1:C9,"select A, B, C where B >= date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'",1)

	A	B	C	D	E	F	G
1	**Store**	**Date**	**Sales**				
2	A	1/17/22	34				
3	A	1/18/22	39				
4	A	1/19/22	31				
5	A	1/20/22	40				
6	B	1/17/22	23				
7	B	1/18/22	29				
8	B	1/19/22	28				
9	B	1/20/22	27				
10							
11	Store	Date	Sales				
12	A	1/18/22	39				
13	A	1/19/22	31				
14	A	1/20/22	40				
15	B	1/18/22	29				
16	B	1/19/22	28				
17	B	1/20/22	27				
18							

Observation of the returned data confirms that only rows where the value in the **Date** column is precisely equal to or subsequent to the current date are included. This output furnishes an immediate, comprehensive overview of all current and prospective sales activity, establishing the formula as an excellent instrument for monitoring ongoing operations and planning for future periods. Critically, just like the previous example, this query maintains its dynamic nature, updating automatically with the transition of each calendar day.

Advanced Date Filtering Scenarios

Moving beyond basic "before" or "after" comparisons, the powerful combination of the TODAY() function and the **QUERY() function** enables the construction of highly complex and granular date filters. Users can readily adapt the foundational syntax presented earlier to define filters for specific chronological spans, such as isolating data from the last seven days, focusing on the entirety of the current month, or retrieving records tied to a particular year relative to the present moment.

For example, to retrieve all data points recorded within the last 7 days, including today, the query must incorporate a range defined by two dynamic date boundaries. This calculation necessitates subtracting seven days from the current date using simple arithmetic (`TODAY()-7`) to establish the starting point. The resulting formula structure would resemble: `"select A, B, C where B >= date '"&TEXT(TODAY()-7,"yyyy-mm-dd")&"' and B <= date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'"`. This illustrates the high degree of flexibility afforded by integrating arithmetic operations

directly with TODAY() to dynamically define precise date boundaries.

Another sophisticated application involves filtering based on specific date components, such as isolating data by year or month. While the **QUERY() function** natively supports functions like `year()`, `month()`, and `day()` within its `where` clause, combining these with TODAY() requires careful syntax. For instance, to filter all records that fall within the current year, one could use the structure `where year(B) = year(date '"&TEXT(TODAY(),"yyyy-mm-dd")&"'")`. Although direct date comparisons using the `date 'YYYY-MM-DD'` format are often more performant and easier to maintain, component-based filtering offers unparalleled precision for specialized reporting needs.

Best Practices and Further Resources

To ensure that your dynamic reports utilizing TODAY() and **QUERY()** remain robust, efficient, and easily maintainable within Google Sheets, adherence to several key best practices is strongly recommended. Foremost among these is the absolute necessity of ensuring that the column containing the dates you intend to query is consistently and correctly formatted as a date within Google Sheets. Inconsistent or non-standard date formats are the leading cause of unexpected results or formula errors in complex queries.

When dealing with extraordinarily large datasets, it is prudent to consider potential performance implications. Although **QUERY()** is typically highly optimized, executing very complex calculations or handling millions of rows may result in noticeable delays during sheet refresh times. In such demanding scenarios, strategies such as pre-processing the source data or decomposing highly complex queries into simpler, chained steps can significantly improve efficiency. Furthermore, spreadsheet developers must remain acutely aware of the timezone settings in Google Sheets (accessible via File > Spreadsheet settings), as the output generated by TODAY()'s output is intrinsically linked to the sheet's configured local timezone.

The advanced techniques detailed in this article provide a strong foundation for sophisticated, automated data manipulation. Achieving mastery in the interaction between the dynamic TODAY() function and the comprehensive analytical capabilities of **QUERY()** will undoubtedly elevate your ability to generate dynamic, insightful, and timely reports. For those seeking to further expand their expertise, we encourage exploring additional official resources focused on other specialized Google Sheets functions and the intricacies of the Query Language.

The following related tutorials offer explanations on how to perform other essential operations within Google Sheets:

How to Use **QUERY()** to Select Columns in Google Sheets

How to Use **QUERY()** with **WHERE** Clause in Google Sheets

How to Use **QUERY()** with **LIKE** in Google Sheets