Learning to Filter Data by Date Range Using the Google Sheets QUERY Function

Authored by

Mohammed looti

November 2, 2025

Working with time-series data and defining filters based on chronological constraints are essential components of robust data analysis. Within Google Sheets, this powerful data extraction capability is primarily provided by the versatile QUERY function. While the QUERY function offers unparalleled flexibility for filtering numerical and text data using its SQL-like syntax, handling dates introduces unique requirements. Unlike simple comparisons, dates must be meticulously formatted as specific date literals within the query string. Successfully filtering a dataset by a precise date range requires mastering the integration of several built-in Sheets functions to ensure the query engine interprets the dates correctly.

The standard syntax utilized by the QUERY function demands that any date used for comparison be presented in the strict literal format of `yyyy-mm-dd`. This requirement stems from the underlying Google Visualization API Query Language. However, Google Sheets stores dates internally as sequential serial numbers--a format ideal for mathematical operations but incompatible with the query engine's literal requirements. Therefore, we must employ a crucial combination of the DATEVALUE function and the TEXT function. This two-step conversion process is indispensable: it first converts the human-readable date into its serial number equivalent, then rigidly reformats that number into the required `yyyy-mm-dd` string. This meticulous preparation is critical for accurate comparison against the dates in your source data.

The following foundational formula illustrates the technique used to filter data based on a date that occurs chronologically after a specified starting point. This method is crucial for constructing complex data extraction reports, analyzing recent activity, and building dynamic dashboards in Google Sheets:

=QUERY(A1:C9,"select * where A > date '"&TEXT(DATEVALUE("1/1/2020"),"yyyy-mm-dd")&"'")

In this initial demonstration, the formula is specifically designed to scan the defined data range, **A1:C9**. It selects and returns all rows where the date contained in column A is strictly greater than (i.e., occurs after) January 1st, 2020. The key takeaway here is the necessity of the internal conversion functions--the `TEXT(DATEVALUE(...))` segment--which is paramount to mastering reliable date filtering within the QUERY function. Without this exact formatting, the query will fail to recognize the date as a comparable chronological value.

Understanding the Date Filtering Challenge in Google Sheets

The fundamental difficulty encountered when filtering by date stems from the inherent duality of date representation within spreadsheet software. To the end-user, a date is intuitively understood as a readable string, such as "1/1/2020" or "January 1, 2020." However, for computational efficiency, Google Sheets treats this date as a sequential serial number. For instance, January 1st,

2020, corresponds to the serial number 43831 (representing the number of days elapsed since the epoch date). If a user attempts to directly compare this raw serial number within the QUERY function's criteria, the results will be either inconsistent, inaccurate, or entirely erroneous because the underlying query language explicitly expects a structured date literal format.

To successfully overcome this technical mismatch, we must employ a reliable technique involving string concatenation and functional conversion. The process begins with the DATEVALUE function, which takes a static date string (e.g., "1/1/2020") and returns its associated serial number. This serial output is then immediately passed into the TEXT function. The TEXT function is instructed to apply the mandatory formatting mask `"yyyy-mm-dd"`. This conversion is absolutely critical because the resulting output--a text string conforming to the strict ISO 8601 standard--is what is dynamically concatenated directly into the query statement. This satisfies the stringent requirements of the `date` keyword within the SQL syntax.

This careful sequence ensures that the underlying query statement is syntactically sound before execution. The final constructed SQL command dynamically assembled by the formula will look precisely like `"select * where A > date '2020-01-01'"`. This robust, standardized approach guarantees reliable filtering regardless of the user's local date settings, effectively neutralizing common errors associated with date parsing, localization, and regional formatting inconsistencies. By adhering to the `yyyy-mm-dd` literal, we provide the query engine with the exact chronological context it requires.

To clearly demonstrate these capabilities across various filtering scenarios, we will utilize the following sample dataset for all subsequent examples. This structured table includes a set of dates, associated transaction IDs, and corresponding monetary values, allowing us to accurately observe the precise effects of applying specific chronological filtering criteria:

	A	B	C	D	E
1	**Date**	**Product**	**Revenue**		
2	1/1/2020	A	10		
3	1/3/2020	A	6		
4	1/3/2020	B	8		
5	1/2/2020	C	14		
6	1/5/2020	A	10		
7	1/9/2020	B	19		
8	1/23/2020	B	22		
9	1/19/2020	C	14		
10	1/14/2020	A	18		
11	1/9/2020	B	8		
12	1/19/2020	C	4		
13	1/22/2020	C	7		
14	1/25/2020	A	7		
15	1/4/2020	B	11		
16	1/3/2020	B	13		
17	1/13/2020	C	8		
18					
19					
20					

Example 1: Filtering Data Before a Specific Date

A frequent requirement in historical data analysis is the need to isolate and extract all records that occurred prior to a designated cutoff date. This objective is easily met by making a simple modification to the comparison operator within the QUERY function. Instead of using greater than (>), we switch to the less than (<) operator. Crucially, we maintain the exact same date conversion methodology to ensure chronological accuracy and prevent any potential parsing errors, thereby keeping the formula structure consistent and reliable.

Consider a scenario where we need to isolate all transactions that were recorded prior to, but not including, January 10th, 2020. We utilize the less than operator (<). This comparison rigorously checks if the date contained in column A is chronologically earlier than the specified cutoff date. If the requirement were slightly different--to include the cutoff date itself--the user would simply substitute the less than operator with the less than or equal to operator (<=). Understanding this distinction between exclusive and inclusive boundaries is essential for precise report generation.

The complete formula below demonstrates how to filter the entire sample dataset, spanning the range **A1:C17**, to return only those rows where the date precedes 1/10/2020:

=QUERY(A1:C17,"select * where A < date '"&TEXT(DATEVALUE("1/10/2020"),"yyyy-mm-dd")&"'")

Upon successful execution, the resulting output clearly and visually confirms the effectiveness of the filter. Only the rows that strictly satisfy the condition--where the date is chronologically before January 10th, 2020--are successfully returned. This demonstrates the precise subsetting of data based on a historical timing boundary, offering significant analytical utility when reviewing past records or excluding recent activity.

E1 =QUERY(A1:C17,"select * where A < date '"&TEXT(DATEVALUE("1/10/2020"),"yyyy-mm-dd")&"'")

	A	B	C	D	E	F	G
1	**Date**	**Product**	**Revenue**		Date	Product	Revenue
2	1/1/2020	A	10		1/1/2020	A	10
3	1/3/2020	A	6		1/3/2020	A	6
4	1/3/2020	B	8		1/3/2020	B	8
5	1/2/2020	C	14		1/2/2020	C	14
6	1/5/2020	A	10		1/5/2020	A	10
7	1/9/2020	B	19		1/9/2020	B	19
8	1/23/2020	B	22		1/9/2020	B	8
9	1/19/2020	C	14		1/4/2020	B	11
10	1/14/2020	A	18		1/3/2020	B	13
11	1/9/2020	B	8				
12	1/19/2020	C	4				
13	1/22/2020	C	7				
14	1/25/2020	A	7				
15	1/4/2020	B	11				
16	1/3/2020	B	13				
17	1/13/2020	C	8				
18							
19							
20							
21							
22							

Example 2: Filtering Data After a Specific Date

Conversely, the task of isolating data recorded after a significant event, such as a product launch, system upgrade, or project start date, is equally fundamental and straightforward to implement. This filtering pattern is frequently applied when analysts need to focus exclusively on recent activity, analyze post-event performance, or systematically exclude legacy data from current calculations. To achieve this forward-looking filter, we utilize the greater than operator (>).

If the analytical objective is to examine all transactions that occurred strictly after January 10th, 2020 (meaning January 10th itself is excluded), the greater than operator is the appropriate choice.

Consistent with the previous example, if the requirement necessitates including the specified date in your results--to create an inclusive boundary--you must substitute the greater than operator with the greater than or equal to operator (`>=`).

The following formula filters the data range **A1:C17** to retrieve only the rows where the date in column A is chronologically after 1/10/2020. Notice that the structural pattern remains identical to the preceding examples, which reinforces the repeatable nature of the crucial date conversion segment, regardless of the comparison operator used:

=QUERY(A1:C17,"select * where A > date '"&TEXT(DATEVALUE("1/10/2020"),"yyyy-mm-dd")&"'")

As clearly visualized in the output below, the filter successfully excludes all records up to and including the specified date of January 10th, returning only the subsequent chronological entries. This outcome decisively confirms that the date literal conversion process is functioning correctly, allowing the query engine to execute accurate chronological comparisons across the entire dataset, thereby providing a clear view of post-event activity.

E1 fx =QUERY(A1:C17,"select * where A > date '"&TEXT(DATEVALUE("1/10/2020"),"yyyy-mm-dd")&"'")

	A	B	C	D	E	F	G
1	**Date**	**Product**	**Revenue**		Date	Product	Revenue
2	1/1/2020	A	10		1/23/2020	B	22
3	1/3/2020	A	6		1/19/2020	C	14
4	1/3/2020	B	8		1/14/2020	A	18
5	1/2/2020	C	14		1/19/2020	C	4
6	1/5/2020	A	10		1/22/2020	C	7
7	1/9/2020	B	19		1/25/2020	A	7
8	1/23/2020	B	22		1/13/2020	C	8
9	1/19/2020	C	14				
10	1/14/2020	A	18				
11	1/9/2020	B	8				
12	1/19/2020	C	4				
13	1/22/2020	C	7				
14	1/25/2020	A	7				
15	1/4/2020	B	11				
16	1/3/2020	B	13				
17	1/13/2020	C	8				
18							
19							
20							
21							

Example 3: Defining a Precise Date Range

The most sophisticated and commonly required application of date filtering is defining a precise date range. This task requires establishing both a start date condition and an end date condition, which must be simultaneously satisfied by the data records. This is accomplished by combining the two filtering criteria using the mandatory `AND` logical operator within the QUERY function's criteria string. The use of `AND` ensures that a record must fall within the specified chronological window to be returned.

When engineering a comprehensive date range filter, it is essential to repeat the full date conversion process for both the beginning and the end dates of the range. This means integrating two separate, self-contained conversion structures (the `&TEXT(DATEVALUE(...))&` segment) into the single query string. These two conditions are explicitly linked using the `AND` keyword, which enforces both the lower boundary (Start Date) and the upper boundary (End Date) simultaneously. Failing to format both dates correctly will inevitably result in an incomplete or failed query execution, highlighting the importance of consistency.

To filter for data falling strictly between January 5th, 2020, and January 15th, 2020, the formula must be carefully constructed as follows. In this example, we employ the greater than (>) operator for the start date and the less than (<) operator for the end date, ensuring that the resulting range is exclusive of the boundary dates themselves:

=QUERY(A1:C17,"select * where A > date '"&TEXT(DATEVALUE("1/5/2020"),"yyyy-mm-dd")&"'and A < date '"&TEXT(DATEVALUE("1/15/2020"),"yyyy-mm-dd")&"'")

The resulting query string effectively establishes a closed chronological window for data extraction, ensuring that only records chronologically positioned strictly within the two specified dates are returned. This technique is absolutely indispensable for generating monthly or quarterly reports, analyzing data based on fiscal periods, or tracking specific seasonal trends with high precision.

=QUERY(A1:C17,"select * where A > date '"&TEXT(DATEVALUE("1/5/2020"),"yyyy-mm-dd")&"'ar

A	B	C	D	E	F	G
Date	**Product**	**Revenue**		Date	Product	Revenue
1/1/2020	A	10		1/9/2020	B	19
1/3/2020	A	6		1/14/2020	A	18
1/3/2020	B	8		1/9/2020	B	8
1/2/2020	C	14		1/13/2020	C	8
1/5/2020	A	10				
1/9/2020	B	19				
1/23/2020	B	22				
1/19/2020	C	14				
1/14/2020	A	18				
1/9/2020	B	8				
1/19/2020	C	4				
1/22/2020	C	7				
1/25/2020	A	7				
1/4/2020	B	11				
1/3/2020	B	13				
1/13/2020	C	8				

As clearly illustrated in the output, only the rows where the date falls between January 5th, 2020, and January 15th, 2020, are successfully extracted. Achieving this precise control over the data scope is a hallmark of advanced data manipulation and reporting mastery within Google Sheets.

Advanced Considerations and Dynamic Date Formatting

While the preceding examples utilized static date strings (e.g., "1/1/2020") for simplicity, in professional and large-scale spreadsheet applications, you will almost certainly need to use dynamic dates. These dates are typically derived from cell references, user inputs, or the output of other complex formulas. The fundamental methodology for conversion remains strictly the same, but instead of passing a static string literal to the DATEVALUE function, you reference the specific cell containing the desired date.

For instance, if your report's start date is housed in cell **E1**, the critical portion of your formula responsible for the date conversion would be written as: `&TEXT(E1,"yyyy-mm-dd")&`. A key simplification to note is that if the referenced cell (E1) already contains a date value that Google Sheets recognizes as a date serial number, you can often simplify the process by relying solely on the TEXT function. This function alone is capable of applying the strict `yyyy-mm-dd` formatting

required by the QUERY function, thereby bypassing the use of the DATEVALUE function altogether, simplifying the overall formula structure.

It is paramount to continually reinforce the fact that the vast majority of date filtering failures stem directly from the incorrect formatting of the date literal within the query string. Always rigorously verify that the output generated by the TEXT function segment strictly adheres to the ISO 8601 standard (YYYY-MM-DD). Any deviation from this standard--such as accidentally including extraneous spaces, using regional date formats, or employing two-digit years--will cause the query to fail silently or, more frustratingly, return zero results because the query language cannot accurately interpret the date literal provided.

Mastering this dynamic technique allows developers and analysts to construct highly flexible, robust, and reusable dashboards. In these applications, end-users can simply input start and end dates into designated control cells, and the main data table automatically updates its filtered view via the QUERY function. This level of automation significantly streamlines reporting workflows and enhances user interactivity.

Additional Resources for Date Operations

A comprehensive understanding of how dates are managed and manipulated in Google Sheets extends far beyond the scope of simple filtering demonstrated here. Operations such as precisely calculating the duration or difference between two dates, or efficiently extracting specific chronological components (like the year, month, or day of the week), are foundational skills for comprehensive data analysis and business intelligence reporting. We strongly encourage further exploration of these advanced date handling techniques to maximize the utility of your spreadsheets:

Learning to calculate the precise number of working days between two dates, correctly excluding weekends and holidays, using the powerful `NETWORKDAYS` function.

Utilizing the `EOMONTH` function for reliable financial reporting, ensuring accurate calculation of the last day of any given month.

Applying sophisticated conditional formatting rules based on date proximity, such as automatically highlighting deadlines approaching expiration or records that are newly entered.

The following tutorials explain how to perform other common operations with dates in Google Sheets: