Can I format Date Values in PROC SQL?

Authored by

stats writer

November 19, 2025

RECOMMENDED CITATION

stats writer (2025). *Can I format Date Values in PROC SQL?*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=96920

The answer to whether one can format date values in PROC SQL is an emphatic **yes**. While standard SQL environments handle data types differently, within the powerful SAS system, specialized functions and statements are available specifically for managing and displaying temporal data. Formatting dates is a crucial requirement for effective reporting, ensuring that the complex numerical representations used internally by the system are translated into human-readable and standardized calendar formats for end-users.

In PROC SQL, there are primary mechanisms for achieving this formatting flexibility. The most direct and commonly utilized method is leveraging the FORMAT statement directly within the `SELECT` clause. This approach dynamically applies a chosen display format to the output column without modifying the underlying dataset. Additionally, for complex data type conversions or when manipulating dates stored as character strings, SAS provides the intrinsic `PUT function` and `INPUT function`.

While the FORMAT statement exclusively controls the aesthetic appearance of the output, the `INPUT function` and `PUT function` are essential for converting values between character, numeric, and temporal types. Understanding when to apply the display-only format versus the conversion functions is key to mastering temporal data manipulation within PROC SQL queries.

You can use the **FORMAT** statement within **PROC SQL** in SAS to format date values in a specific way. This powerful declarative approach is highly efficient for controlling how results are displayed without altering the fundamental structure of the data source.

The following detailed examples illustrate how to implement the **FORMAT** statement effectively, both for existing columns and for newly created, calculated variables within your SQL queries.

Understanding Date Handling in SAS

Before delving into the practical application of formatting, it is essential to understand the unique way the SAS system manages dates internally. Unlike many other database systems that store dates as structured objects, SAS stores all temporal data--dates, times, and datetimes--as numeric values. A standard **SAS date value** represents the total number of days between January 1, 1960, and the specific date in question. This means that January 1, 1960, is stored as 0, while subsequent dates are stored as progressively larger positive integers.

This numerical storage method is highly advantageous because it enables direct arithmetic operations on dates. Users can effortlessly add or subtract a fixed number of days, weeks, or months directly from the date variable without relying on complex, specialized functions, as demonstrated in subsequent examples involving derived variables. However, because the underlying data is a large integer, an **SAS format** must be applied to translate this numeric value into a calendar date that users can readily interpret, such as DDMMMYY or MM/DD/YYYY.

The distinction between the internal representation (the numeric count of days) and the external display (the formatted calendar date) is the foundation of mastering date manipulation in PROC SQL. If a column contains a numeric SAS date value but lacks an explicit format, the output will simply show the large integer count of days since 1960, rendering it essentially meaningless in standard reporting. Therefore, any SAS date variable intended for reporting must have an associated format, whether it was permanently assigned during the data creation DATA step or dynamically applied in a PROC SQL query using the **FORMAT statement**.

The Role of the FORMAT Statement in PROC SQL

The **FORMAT statement** is the core mechanism within PROC SQL for controlling the presentation layer of data, specifically for temporal and complex numeric variables. When employed within a `SELECT` statement, the FORMAT statement temporarily overrides any format that might have been permanently assigned to the variable in the input dataset or catalog. This capability provides immense flexibility, allowing users to tailor the display format dynamically based on the specific reporting requirements of the executing query.

The syntax is intuitive: specify the column name, followed by the keyword `FORMAT=`, and then the desired SAS format name. For date variables, developers frequently utilize formats such as `DATE9.` (e.g., 01JAN2023), `MMDDYY8.` (e.g., 01/01/23), or `DDMMYY10.` (e.g., 01-01-2023). It is paramount to include the trailing dot (`.`) when specifying the format name in PROC SQL, as this is a fundamental requirement of SAS syntax for referencing formats and informats.

Crucially, using the FORMAT statement in this manner ensures that the raw numeric data remains completely unaltered in the original table definition. The formatting change is applied exclusively to the output view generated by the SQL procedure. This makes the process highly efficient and non-destructive, suitable for producing multiple reports with varied date displays from a single, consistent source table. Moreover, this flexibility extends beyond existing columns, allowing the format to be applied simultaneously to new columns derived from complex calculations, ensuring seamless data manipulation and presentation.

Example 1: Formatting Existing Date Variables

To demonstrate the practical application of formatting existing date columns, let us first establish a sample dataset in SAS. This dataset, named `my_data`, models a transactional scenario, tracking the **start date** of various retail promotions and the corresponding total **sales** achieved. Note that in the DATA step below, the `start_date` is initially read using the `DATE9.` informat and is assigned the same `DATE9.` format for its default display.

The dataset creation and its initial viewing are demonstrated in the following code block. Observe how the `format date9.;` statement within the `DATA step` ensures the initial display conforms to

the standard SAS date format (DDMMMYYYY).

```
/*create dataset*/
data my_data;
format start_date date9.;
input start_date :date9. sales;
datalines;
01JAN2023 22
01FEB2023 16
14MAR2023 11
01MAY2023 32
13MAY2023 15
18AUG2023 11
20OCT2023 36
;
run;

/*view dataset*/
proc print data=my_data;
```

The initial output of this dataset confirms the standard date format assignment applied during the data step:

Obs	start_date	sales
1	01JAN2023	22
2	01FEB2023	16
3	14MAR2023	11
4	01MAY2023	32
5	13MAY2023	15
6	18AUG2023	11
7	20OCT2023	36

Next, we utilize **PROC SQL** to select all records from this dataset, but we instruct the procedure to display the values in the **start_date** column using a completely different format: `MMDDYY8.`. This format renders the date in the common MM/DD/YY structure (e.g., 01/01/23). This clearly demonstrates the ability of PROC SQL to override the default format defined in the source dataset.

```
/*select all rows and format start_date column using mmddyy8.*/
proc sql;
select start_date format=mmddyy8., sales
from my_data;
quit;
```

The successful execution of this query yields a new output table where the **start_date** column is correctly displayed in the requested **MM/DD/YY** format, validating the effectiveness of the dynamic **FORMAT statement** within the `SELECT` clause. This modification is purely aesthetic and localized to the result set generated by the query.

start_date	sales
01/01/23	22
02/01/23	16
03/14/23	11
05/01/23	32
05/13/23	15
08/18/23	11
10/20/23	36

Extending Formatting to Derived Variables

A significant strength of the **FORMAT statement** in PROC SQL is its applicability to variables that are newly created or derived during the query execution. Since SAS dates are stored as numeric values, simple arithmetic operations--such as adding a fixed number of days to an existing date column to determine a projected end date--result in a new numeric value that is also a valid SAS date.

When a new date column is generated through calculation (e.g., `start_date + 7`), this derived variable does not inherit or possess any default format. Consequently, if a format is not explicitly applied, PROC SQL will display the raw numeric value, which is unreadable to most users. To ensure this newly calculated variable is meaningful and presentable, the FORMAT statement must be utilized immediately after defining the calculated expression.

The standard syntax for this application is structured as: `(calculation expression) AS new_column_name FORMAT=desired_format.` This concise syntax allows data professionals to

perform complex date manipulations, assign a readable column alias, and control the presentation format all within a single, highly efficient SQL clause. This significantly streamlines the process of generating comprehensive, readable reports.

Example 2: Creating and Formatting New Date Columns

Continuing with our sample data, let us now create a new variable, `end_date`, which is calculated to be exactly 7 days after the `start_date`, representing the duration of the promotion period. We must calculate this new temporal value and simultaneously ensure that it is displayed using a robust, easily understood format, such as `DATE9.`.

In the following query, we perform simultaneous formatting actions. First, we re-apply the `MMDDYY8.` format to the original `start_date` column. Second, we define the new variable using the additive expression `start_date + 7`, assign it the alias `end_date` using the `AS` keyword, and immediately apply the `DATE9.` format. This ensures the result displays clearly as, for instance, 08JAN2023.

```
/*create new end_date column with specific format*/
proc sql;
select start_date format=mmddyy8., start_date+7 as end_date format=date9., sales
from my_data;
quit;
```

Upon reviewing the final output, it is clear that we successfully specified the desired display format for both the existing **start_date** variable (using `MMDDYY8.`) and the newly created **end_date** variable (using `DATE9.`). This single query perfectly illustrates the flexibility and compounding power of using multiple **FORMAT statements** within a single PROC SQL statement to manage presentation details.

start_date	end_date	sales
01/01/23	08JAN2023	22
02/01/23	08FEB2023	16
03/14/23	21MAR2023	11
05/01/23	08MAY2023	32
05/13/23	20MAY2023	15
08/18/23	25AUG2023	11
10/20/23	27OCT2023	36

Alternative Methods: INPUT and PUT Functions

While the **FORMAT statement** is the optimal solution for controlling the display of existing SAS numeric date values, the `INPUT function` and `PUT function` are critical when data type conversion is necessary--specifically, translating data between character and numeric representations. These functions operate at the fundamental data manipulation level, not merely the output display level.

The `INPUT function` is specifically designed to convert a character value into a numeric (or date) value, requiring an informat to guide the conversion process. This function is vital if the date information is initially stored as text in the dataset and needs to be recognized by SAS as a true date for arithmetic operations or comparisons. The function utilizes the syntax `INPUT(source_variable, informat.)`. For example, converting a character string '20230101' into the corresponding SAS date value would necessitate an appropriate informat, such as `YYMMDD8.`.

Conversely, the `PUT function` performs the opposite conversion, transforming a numeric value (like a calculated SAS date) into a character string, using a specified format. The syntax is `PUT(source_variable, format.)`. Using the `PUT function` within PROC SQL is required if the report output demands the date to be strictly character-based (e.g., when concatenating the date with other text fields to form a composite key, or when outputting data to an external system that accepts only string input). While the **FORMAT statement** is generally preferred for simple display changes, the `PUT function` is the necessary technical tool for permanent character conversion within a query.

Choosing the Right Date Format in SAS

The SAS system provides an extensive library of predefined formats tailored for displaying date values, allowing developers to meet diverse regional standards and reporting specifications. Selecting the appropriate format is not merely a stylistic choice; it requires careful consideration of the required output length, the necessary inclusion of separation characters (slashes, hyphens), and the desired representation of the year (two-digit or four-digit). A key caution is avoiding formats that are too narrow for the data (e.g., using `DATE7.` for a DATE9. value), which will result in the display of truncation indicators, typically asterisks (`********`).

For comprehensive data presentation, here is a breakdown of commonly utilized SAS date formats:

DATE9.: Displays the date in DDMMMYYYY format (e.g., 01JAN2023). This format is universally recognized and is highly recommended due to its unambiguous nature.

MMDDYY8.: Displays the date in MM/DD/YY format (e.g., 01/01/23). While compact, be mindful of the two-digit year, which may lead to ambiguity if the century is not implied or known.

MMDDYY10.: Displays the date in MM/DD/YYYY format (e.g., 01/01/2023). This is generally preferred over the 8-digit version as it explicitly includes the century, eliminating ambiguity.

YYMMDD10.: Displays the date in YYYY-MM-DD format (e.g., 2023-01-01). This format adheres to ISO standards and is often favored for international data exchange and efficient chronological sorting.

DDIIS8.: Displays the date in DD/MM/YY format, which is better suited for reporting audiences following common European dating standards.

When specifying the format width (the number following the dot, e.g., `DATE9.`), always ensure the width is ample to accommodate the full character representation, including all separator characters. For example, `MMDDYY10.` requires a width of 10 to include the two slashes and the four-digit year, while `MMDDYY8.` requires a width of 8 for the two slashes and the two-digit year. Utilizing the correct format ensures report accuracy and significantly enhances overall data readability, a straightforward task easily accomplished within PROC SQL.