MWSUG 2017 – Paper SA04

PROC SORT (then and) NOW

Derek Morgan, PAREXEL International, Billerica, MA

ABSTRACT

The SORT procedure has been an integral part of SAS® since its creation. The sort-in-place paradigm made the most of the limited resources at the time, and almost every SAS program had at least one PROC SORT in it. The biggest options at the time were to use something other than IBM's SYNCSORT as the sorting algorithm, or whether you were sorting ASCII data versus EBCDIC data.

These days, PROC SORT has fallen out of favor; after all, PROC SQL allows merging without using PROC SORT first, while the performance advantages of HASH sorting cannot be overstated. This leads to the question: is the SORT procedure still relevant to any other than the SAS novice, or the terminally stubborn who refuse to HASH? The answer is a surprisingly clear, "yes." PROC SORT has been enhanced to accommodate twenty-first century needs, and this paper will discuss those enhancements.

INTRODUCTION

The largest enhancement to the SORT procedure is the addition of collating sequence options. This is first and foremost a recognition that SAS is an international software package, and that SAS users no longer work exclusively with English-language data. This capability is part of National Language Support (NLS,) and doesn't require any additional modules. You may use standard collations, SAS-provided translation tables, custom translation tables, standard encodings, or rules to produce your sorted dataset. However, you may only use one collation method at a time.

USING STANDARD COLLATIONS, TRANSLATION TABLES AND ENCODINGS

A long time ago, SAS would allow you to sort data using ASCII rules on an EBCDIC system, and vice versa. The list below shows the standard collating sequences that are available in SAS 9.4:

- ASCII
- *DANISH*
- EBCDIC
- *FINNISH*
- *NATIONAL*
- *NORWEGIAN*
- *REVERSE*
- *SWEDISH*

To use the Finnish collation, you would add it to the PROC SORT statement.

```
PROC SORT DATA=mydata FINNISH;
BY var1;
RUN;
```

The NATIONAL collation may not be available in your SAS installation; check with your SAS administrator before trying to use it. The standard translation tables provided by SAS add Italian, Polish, and Spanish to the above list as collating sequences. However, these require the use of the SORTSEQ= option. If none of the SAS-provided translation tables work for your situation, you may even create your own translation table. Creation of a custom translation table should be viewed as a last resort. It depends on the installation of SAS, and there are very specific rules for implementing a custom translation table. Here is an example of using a translation table:

```
PROC SORT DATA=mydata SORTSEQ=ITALIAN;
BY var1;
RUN;
```

Encoding values such as “wlatin-1” or “utf-8”, can be used in the SORTSEQ= option, which will perform a binary collation of the character data represented in the specified encoding. A full list of available encodings is in the National Language Support documentation.

```
PROC SORT DATA=mydata SORTSEQ=latin2; /* Central European ISO Standard */
BY var1;
RUN;
```

Should you need any of this functionality, the SORTSEQ= option will provide it; moreover, it can be specified as a system option, e.g., OPTIONS SORTSEQ=ITALIAN;. That will become the default collating sequence throughout your program.

RULES-BASED COLLATION

I believe this is the way to unleash the hidden power inside PROC SORT. It requires the use of the SORTSEQ= option on the PROC SORT statement itself; you cannot specify it as a system option. The keyword to invoke rules-based collation is SORTSEQ=LINGUISTIC.

The LINGUISTIC keyword causes SAS to sort characters according to the linguistic rules associated with the language and locale in effect. However, the LINGUISTIC keyword has multiple options that modify the linguistic collating sequence. Some of these solve problems that have required creative SAS coding for years. You may use more than one LINGUISTIC option, but you cannot use SORTSEQ=LINGUISTIC with a translation table or encoding.

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(ALTERNATE_HANDLING=SHIFTED);
BY var1;
RUN;
```

The ALTERNATE_HANDLING= option will allow your sort to treat the handling of differences in spaces, punctuation and symbols as less important than differences between letters. Without this option, differences in those characters are of equal weight as letters, which is the default.

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(COLLATION=collation-value);
BY var1;
RUN;
```

The COLLATION= option specifies character ordering. One advantage of specifying collation this way, as opposed to using encoding, is the handling of multiple languages (STROKE) with a single collation definition.

There are several valid values:

BIG5HAN	Specifies Pinyin ordering for Latin and specifies bug5 charset ordering for Chinese, Japanese, and Korean characters.
DIRECT	Specifies a Hindi variant.
GB21312HAN	Specifies Pinyin ordering for Latin and specifies gb2312han charset ordering for Chinese, Japanese, and Korean characters.
PHONEBOOK	Specifies a telephone-book style for ordering of characters. Select PHONEBOOK only with the German language.
PINYIN	Specifies an ordering for Chinese, Japanese, and Korean characters based on character-by-character transliteration into Pinyin. This ordering is typically used with simplified Chinese.
POSIX	This option specifies a “C” locale ordering of characters.
STROKE	Specifies a nonalphabetic writing style ordering of characters. Select STROKE with Chinese, Japanese, Korean, or Vietnamese languages. This ordering is typically used with Traditional Chinese.
TRADITIONAL	Specifies a traditional style for ordering of characters.

SOLVING THE "ADDRESS PROBLEM" WITH RULES-BASED COLLATION

Have you ever dealt with addresses in a single variable, where the numeric portion of the address is part of the same field? How would you sort the following list?

1801 Somewhere Ave.
1652 Somewhere Ave.
7137 Somewhere Ave.
10381 Somewhere Ave.
4177 Somewhere Ave.
4200 Somewhere Ave.
7262 Somewhere Ave.
12425 Somewhere Ave.
506 Somewhere Ave.

If you just use PROC SORT, the resulting list will be sorted in a logical fashion—unless you're a delivery person who has to make deliveries to all of these addresses. You'll be working your way from one end of Somewhere Ave. to the other end to the middle to the end (from 4200 to 506) back to the middle.

```
PROC SORT DATA=mydata;
BY address;
RUN;
```

1	10381 Somewhere Ave.
2	12425 Somewhere Ave.
3	1652 Somewhere Ave.
4	1801 Somewhere Ave.
5	4177 Somewhere Ave.
6	4200 Somewhere Ave.
7	506 Somewhere Ave.
8	7137 Somewhere Ave.
9	7262 Somewhere Ave.

Do you need to create a dataset with the address number separated from the street name before sorting? Maybe a few years ago, but now the PROC SORT option NUMERIC_COLLATION=ON will solve your problem without having to parse the address or change the data structure!

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(NUMERIC_COLLATION=ON);
BY var1;
RUN;
```

1	506 Somewhere Ave.
2	1652 Somewhere Ave.
3	1801 Somewhere Ave.
4	4177 Somewhere Ave.
5	4200 Somewhere Ave.
6	7137 Somewhere Ave.
7	7262 Somewhere Ave.
8	10381 Somewhere Ave.
9	12425 Somewhere Ave.

Changing LOCALE for the Duration of the SORT

This setting will override the LOCALE settings in effect, allowing PROC SORT to use a different LOCALE (and usually language) from the remainder of the program. LOCALE values can be found in the National Language Support documentation.

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(LOCALE=locale-value);
BY var1;
RUN;
```

ADJUSTING THE SORT SENSITIVITY

The final set of LINGUISTIC options has to do with the sensitivity of the sorting algorithm. The STRENGTH= option allow you to specify which differences are in effect for the sort taking place.

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(STRENGTH=strength-value);
BY var1;
RUN;
```

The possible strength-values are as follows:

VALUE	ALIAS	Explanation
PRIMARY	1	PRIMARY specifies differences between characters, but not case or accents. For example, “a” < “b”, but “a” = “A”, and “Á” = “A”). It is the strongest difference.
SECONDARY	2	Accents in characters are considered secondary differences (for example, "as" < "às" < "at").. Depending on the language, other differences between letters will also be considered secondary differences.
TERTIARY	3	**This is the default sort strength for US English**. Upper and lowercase differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò"). A non-English example would be the difference between large and small Kana
QUATERNARY	4	When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, "a-b" < "ab" < "aB"). The quaternary level should be used if ignoring punctuation is required or when processing Japanese text.
IDENTICAL	5	When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the Normalization Form D (NFD) form of each string are compared at this level, just in case there is no difference at levels 1-4. This level should be used sparingly, because code-point value differences between two strings rarely occur. For example, only Hebrew cantillation marks are distinguished at this level.

You can also define how you want uppercase and lowercase letters sorted with the CASE_FIRST= option. CASE_FIRST=UPPER sorts uppercase letters before lowercase letters, while CASE_FIRST=LOWER sorts lowercase letters before uppercase letters. This option only works in conjunction with the STRENGTH= option, and only when STRENGTH is TERTIARY, QUATERNARY, or IDENTICAL:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(STRENGTH=TERTIARY
                                        CASE_FIRST=UPPER);
BY var1;
RUN;
```

If the STRENGTH= value is PRIMARY or SECONDARY, the CASE_FIRST option has no effect because upper- and lowercase differences are not detected at these levels.

Instead of using the UPPERCASE() function to create an identical-case version of the variable you want to sort on, you can use STRENGTH=PRIMARY or SECONDARY to perform case-insensitive sorting, and use an option to avoid modifying your data before sorting.

```
PROC SORT DATA=mydata;
BY name;
RUN;
```

The above code will result in the following:

OBS	name
1	MACK
2	MacCarron
3	MacManus
4	Macallen
5	Macarthur
6	Maccaray
7	Maccarron
8	Mc Grady
9	McAllen

To perform a case-insensitive sort you could:

```
DATA mydata2;
SET mydata;
uname = UPCASE(name);
RUN;

PROC SORT DATA=mydata2;
BY uname;
RUN;
```

Or you could reduce the sensitivity of the SORT to ignore case, avoiding the function call and the extra dataset:

```
PROC SORT DATA=mydata
SORTSEQ=LINGUISTIC(STRENGTH=PRIMARY)
;
BY name;
RUN;
```

Using UPPERCASE() in a DATA step		
OBS	**uname**	**name**
1	MACALLEN	Macallen
2	MACARTHUR	Macarthur
3	MACCARAY	Maccaray
4	MACCARRON	MacCarron
5	MACCARRON	Maccarron
6	MACK	MACK
7	MACMANUS	MacManus
8	MC GRADY	Mc Grady
9	MCALLEN	McAllen

PROC SORT with LINGUISTIC processing	
OBS	***name***
1	Macallen
2	Macarthur
3	Maccaray
4	MacCarron
5	Maccarron
6	MACK
7	MacManus
8	Mc Grady
9	McAllen

ENHANCED HANDLING OF RECORDS WITH DUPLICATE KEYS

Everyone who uses PROC SORT is most likely familiar with the NODUPKEY option, which removes records with identical keys. Even this has been enhanced, with the ability to choose which of the observations is kept, and you can send the duplicates that aren't kept to a dataset. The next set of examples will use a dataset containing customer ID numbers and the level of ticket purchased for a given event. The who customers have purchased tickets to more than one event are bolded in the table. The original sort order is by customer ID and descending ticket level.

Customer ID	Ticket Level
10270	7
12230	3
19323	**5**
19323	**4**
22779	3
28819	6
29252	2
30457	**7**
30457	**3**
30457	**2**
30457	**2**
31918	**4**
31918	**1**

How can you find out which records are eliminated? Try this:

```
PROC SORT DATA=sortsamp OUT=sort1 NODUPKEY DUPOUT=dups;
BY cid;
RUN;
```

Dataset SORT1			Dataset DUPS	
Customer ID	Ticket Level		Customer ID	Ticket Level
10270	7		19323	4
12230	3		30457	3
19323	5		30457	2
22779	3		30457	2
28819	6		31918	1
29252	2			
30457	7			
31918	4			

This is handy when you have multiple keys and a lot of records and you have to track down why you have duplicate-keyed records; perhaps you don't have enough keys for uniqueness or you have a different

problem with the data. Again, this is all done within PROC SORT, so you don't have to write DATA step or SQL code to keep your removed duplicates.

Another PROC SORT option lets you pull out all the duplicate keyed records. Instead of running a DATA step like this:

```
DATA inspectdups;
SET sortsamp;
BY cid;
IF NOT (FIRST.cid AND last.cid) THEN
   OUTPUT;
RUN;
```

Use the NOUNIQUEKEY option on PROC SORT:

```
PROC SORT DATA=sortsamp NOUNIQUEKEY OUT=alldups UNIQUEOUT=uniques;
BY cid;
RUN;
```

Dataset INSPECTDUPS Created by DATA step			**Dataset ALLDUPS Created by PROC SORT and NOUNIQUEKEY option**	
Customer ID	**Ticket Level**		**Customer ID**	**Ticket Level**
19323	5		19323	5
19323	4		19323	4
30457	7		30457	7
30457	3		30457	3
30457	2		30457	2
30457	2		30457	2
31918	4		31918	4
31918	1		31918	1

And the unique records? Using the UNIQUEOUT= option in combination with the NOUNIQUEKEY options sends all your uniquely-keyed records to their own dataset:

Dataset UNIQUES	
Customer ID	**Ticket Level**
10270	7
12230	3
22779	3
28819	6
29252	2

What About the NODUPLICATES Option?

Surprisingly, the NODUPLICATES option is no longer documented as a part of PROC SORT, although it is documented in the SORT() function in SAS Component Language. However, the option still functions. The NODUPLICATES option has always come with the warning that it doesn't test each record against all other records for a duplicate; it only tests against the adjacent record. Therefore, it doesn't always work in an unsorted dataset. PROC SQL provides a better way to remove complete duplicates:

```
PROC SQL:
CREATE TABLE all_duplicates_removed AS
SELECT DISTINCT *
FROM data_with_dup_records
ORDER BY keyvar1… keyvarN
;
QUIT:
```

ADDITIONAL SORT OPTIONS

There are other PROC SORT options available, such as DATECOPY, which retains the date and time of the original, unsorted dataset in the sorted version. This can be useful with version control.

The REVERSE option collates in reverse, according to the character set in use. This is also the same as SORTSEQ=REVERSE. If you use the REVERSE option with SORTSEQ=REVERSE, they will cancel each other out, and the dataset will be sorted in regular order.

The PRESORTED option is an efficiency aid with SAS datasets. If you think the data are already sorted, using this option will skip the sort process if the dataset is already sorted. This ONLY applies to SAS datasets. Do NOT use this option when using other DBMS or SAS/ACCESS®, as the records may not be accessed in the assumed order.

SUMMARY

PROC SORT has grown in its capabilities since the early days of SAS. Although you no longer need to sort in order to merge, and there are much more efficient ways to sort data, version 9 of SAS has returned PROC SORT to relevance. Many of the enhancements are linked with National Language Support, and acknowledge that SAS is truly an international software package working with data in multiple languages.

Several options for PROC SORT replace DATA step or SQL code. You can perform case-insensitive sorting, select uniquely-keyed records while placing the eliminated records into a data set for later inspection. Conversely, you can select all the duplicate-keyed records, eliminating the uniquely-keyed ones.

Other options maintain the date of the unsorted data set, reverse the collating sequence (not quite the same as sorting in descending order), and allow SAS to check the sort status of a data set before sorting it, which can prove to be quite an efficiency boost.

While HASH sorting is undeniably more efficient, the language enhancements in PROC SORT and the options that replace common SAS code may even those odds. PROC SORT is once again a useful tool in the SAS programmer's toolkit.

REFERENCES

The SAS Institute White Paper on PROC SORT Linguistic Capabilities, http://support.sas.com/resources/papers/linguistic_collation.pdf, 28 August 2017

Creating Order out of Character Chaos: Collation Capabilities of the SAS System, Scott Mebust and Michael Bridgers, *Proceedings of the SAS® Global Forum 2007 Conference,* SAS Institute Inc. Cary, NC

CONTACT INFORMATION:

Comments and questions welcome to:

Derek Morgan
E-mail: mrdatesandtimes@gmail.com