

Using Formats to Subset Impossibly Large SAS Datasets

Michael A. Raithel, Westat, Inc.

ABSTRACT

SAS programmers are finding themselves working with larger and larger SAS datasets. As the size of SAS datasets increase, so too does the strain on computing resources. Programs that took minutes to execute are running longer and longer as the number of observations increases. Sometimes the SAS datasets are so large that normal programming practices such as match-merging files simply fail due to their overwhelming the available computer resources. This particular issue can surface when you are using a file of ID variables to subset an impossibly large SAS dataset. What to do?

This presentation illustrates a methodology for overcoming computing resource issues when matching a SAS dataset of ID variables to an exceptionally large SAS dataset in order to extract matching observations. The method starts with the creation of a SAS format with the ID variables of the smaller SAS dataset. Once created, the format is used in a sequential read of the larger SAS dataset to select only those observations that match to the ID variable values stored in the format. Consequently, this methodology uses a single pass of each of the SAS datasets, so no sorting or match-merging of the datasets is necessary.

Readers can begin using this technique in their programs that subset impossibly large SAS datasets immediately.

INTRODUCTION

Exceptionally large SAS datasets usually require more resources to be processed. They consume more memory, more network bandwidth, and more work space resources. Programs processing very large SAS datasets take longer to run and thus demand more of a programmer's time. Familiar SAS tasks such as sorting, match-merging SAS datasets, and sub-setting via matching datasets seem to lag, thus putting a burden on programmers who need reasonable turnaround times to meet project schedules.

Matching a SAS dataset of identifiers to an extremely large SAS dataset to select matches and create a subset is a case-in-point. To perform this task, it is necessary to first sort both SAS datasets by the identifier variable(s). The identifier variables could be in upper, lower, or mixed case thus potentially causing mismatches in the upcoming matching. This condition could necessitate a complete pass of each SAS dataset to standardize the ID variables—making them all upper case or lower case in both datasets. Then, the matching program might either take a long, long, long time to run or fail due to an “out of resources” condition. Consequently, it is advisable to find an alternative to the usual subsetting-by-matching-identifiers regimen when dealing with huge SAS datasets.

THE TRADITIONAL PROGRAMMING METHOD

The goal of the example program is to subset a “large” SAS dataset of the most common surnames from the 2010 census to only those with likely Asian surnames. The census dataset is named CensusSurnames, the Asian surnames dataset is named APISurnames. NOTE: Both datasets were downloaded from the Internet. Links to those files can be found in the References section of this paper.

This is an example of the code for the traditional approach:

```
/* Specify the data directory */  
libname datadir "I:\Important Project\Production Directory";
```

```

/* Sort the API Surname dataset by surname */
proc sort data=datadir.ApiSurnames (keep=surname)
    out=ApiSurnames_Sorted;
    by surname;
run;

/* Sort the Census dataset by surname */
proc sort data=datadir.CensusSurnames out=CensusSurnames_Sorted;
    by surname;
run;

/* Match merge the files and keep only the matches*/
Data API_Census_Only;
merge ApiSurnames_Sorted(in=a)
      CensusSurnames_Sorted(in=b);

    by surname;

    if a and b;

run;

```

In the example, both datasets are sorted, and then a common match-merge is performed by variable SURNAME. So, each SAS dataset is effectively read twice—once in a sort and again in a DATA step. As previously mentioned, this method can potentially have issues if the values of the ID variables are in differing cases (upper, lower, mixed) between the two datasets being matched. In that case, additional DATA steps may be needed to UPCASE or LOWERCASE the value stored in SURNAME either before or after the SORTs.

USING A FORMAT TO SUBSET

The methodology this paper is presenting first advises you to read the smaller SAS dataset of identifiers. This is done in a DATA _NULL_ step wherein you create a format value-label pair from each identifier variable. The value is taken from what is stored in the identifier variable and the label is simply the word “YES”. The DATA _NULL_ step also creates the requisite PROC FORMAT statements and sandwiches the value-label pairs between them. All of these statements are written to a temporary flat file. After the DATA _NULL_ step finishes running, the temporary flat file containing the complete PROC FORMAT specification is %INCLUDED to execute and create a temporary format catalog containing the value-label pairs formed from the SAS dataset of identifiers.

Next, a *three-lines-of-code* SAS data step is run to create the extract SAS dataset. The DATA statement has the name of the new extract SAS dataset that will be built. The SET statement identifies the large SAS dataset that is to be subset. It contains a WHERE option specifying that only observations where there is a match between the identifier variable and an entry in the previously-created format are to be written to the new extract SAS dataset. The third statement is the requisite RUN statement.

Recapping the methodology; there are two basic steps:

1. Read the file of identifiers and create a format from the ID variable
2. Read the large file and select only those observations whose ID variables match those in the format

There are several advantages to this methodology. First, neither file needs to be sorted. This reduces the overall run time that a costly sort of the large SAS dataset would entail. Secondly, the ID variables can

be in upper, lower, or mixed case in either or both of the SAS datasets. They do not need to be normalized between the two datasets; thus, keeping the original values of their cases in each dataset. Thirdly, only one complete read of each file is needed in order to create the extract SAS dataset. Finally, SAS is optimized for sequential reads, so this process should run faster than a series of sorts and DATA steps.

This is an annotated example of the code for the format subsetting method.

The first block of code performs some housekeeping tasks. First, it allocates a large number of catalog buffers, a large SAS dataset block size, and a large number of SAS dataset buffers. Then, the SAS data library containing both SAS datasets is specified.

```

/*****
/* Specify a large number of catalog page buffers for better      */
/* performance.                                                  */
*****/
options cbufno=20;
options bufsize = 64k bufno=20;

/*****
/* Specify the SAS data library containing the datasets to be    */
/* processed.                                                    */
*****/
libname datalib "I:\Important Project\Production Directory";

```

The FILENAME statement below specifies that a temporary flat file with a FILEREF of “buildfmt” is to be created in the SAS work library. That flatfile will be populated with PROC FORMAT statements that are created in the following DATA _NULL_ step.

```

/*****
/* Specify a temporary file to hold PROC FORMAT statements */
*****/
filename buildfmt temp;

```

The DATA _NULL_ step creates all the PROC FORMAT statements and writes them to the aforementioned temporary flatfile. It starts by writing the PROC FORMAT statement and the VALUE statement to the flatfile. Then, for every observation read from the smaller file of identifiers, the DATA _NULL_ step creates a value/label pair. Those value/label pairs are also written to the flatfile.

Note that the value (the ID variable) is UPCASEd in the format. This is done to ensure that there are no issues matching to the ID variables in the large dataset where the variables may be all upper, all lower, or mixed case. The subsequent DATA step is going to make provisions for the large dataset's variables not necessarily being UPCASEd.

The DATA _NULL_ step concludes by writing a semicolon and a RUN statement to the flatfile. At this point, the flatfile contains a fully formed format procedure that has each desired ID variable linked to the label “YES”.

```

/*****
/* Read the file of ID variables to be matched, create the PROC  */
/* FORMAT statements and store them in the Temp file.            */
*****/
data _null_;
set datalib.apisurnames(keep=surname) end=eof;

file buildfmt;

```

```

retain doublequote '"';

/* Create first part of PROC FORMAT */
if _n_ = 1 then do;

    put "proc format;";
    put "value $APISurnames";

end;

/* Create name/value pairs of the format */
bigline = doublequote || strip(uppercase(surname)) || doublequote || " = "
           || doublequote || "YES" || doublequote;

put bigline;

/* Create ending part of PROC FORMAT */
if eof then do;
    put ";";
    put "run;";
end;

run;

```

The %INCLUDE includes the fully formed PROC FORMAT procedure stored in the *buildfmt* flatfile and executes it, to create the \$APISurnames format in the work library.

```

/*****
/* %INCLUDE the Temp file to create the format */
*****/
%include buildfmt;

```

Now that the format has been produced, the large SAS dataset can be processed to create the dataset of matching identifiers. This is done by employing the format in a WHERE option on the SET statement. The ID variable is UPCASEd and then *bounced* against the format. If there is a match, the observation is written to the API_Respondents SAS dataset. If not, then the data step moves on to the next observation.

```

/*****
/* Create a file of API-named respondents from the Census Surnames */
/* SAS dataset */
*****/
data datalib.API_Respondents;
set datalib.censusurnames(
    where=(put(strip(uppercase(surname)),
                $APISurnames.) = "YES"));

run;

```

Upon completion of the above DATA step, the API_Respondents SAS dataset will contain all observations in CensusSurnames whose ID variable (SURNAME) was found in the APISurnames lookup dataset.

CONCLUSION

This paper presented a methodology for using a format to match a file of identifiers to a large SAS dataset in order to select observations with matching ID variables. There are two basic steps:

1. Read the file of identifiers and create a format from the ID variable
2. Read the large file and select only those observations whose ID variables match those in the format

There are several advantages to this methodology.

- Neither file needs to be sorted. This reduces the overall run time that a costly sort of the large SAS dataset would entail.
- The ID variables can be in upper, lower, or mixed case in either or both of the SAS datasets. They do not need to be normalized between the two datasets; thus, keeping the original values of their cases in each dataset.
- Only one complete read of each file is needed in order to create the extract SAS dataset.
- SAS is optimized for sequential reads, so this process should run faster than a series of sorts and DATA steps.

This methodology worked well in a real-life example in the author's organization. A talented Westat statistician was trying to match a SAS dataset with 756 observations (subjects) with a SAS dataset containing 657, 007,006 observations (labs and vital signs) that was 150 Gb. She got a fatal "Out of Resources" ERROR: after two-and-a-half hours of processing on a Windows workstation. When she switched to the PROC FORMAT methodology, her program completed in 12 minutes with the resulting extract file containing 2,729, 675 observations.

REFERENCES

Raithel, Michael A. 2017. Did You Know That? Essential Hacks for Clever SAS Programmers: Over 100 Essential Hacks to Make Your Programs Leaner, Cleaner, and More Competitive. Bethesda, Maryland: Michael A. Raithel

Available: http://www.amazon.com/Michael-A.-Raithel/e/B001K8GG90/ref=ntt_dp_epwbk_0

Proc Format, a Speedy Alternative to Sort / Sort /Merge

By Jenine Milum, Wells Fargo, Charlotte, NC

<https://support.sas.com/resources/papers/proceedings12/428-2012.pdf>

SAS 9.4 and Viya 3.5 Programming Documentation

https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/p1upn25lbfo6mkn1wncu4dyh9q91.htm

Most Common Asian Surnames & Meanings

<https://forebears.io/asia/surnames>

Frequently Occurring Surnames in the 2010 Census

<https://www2.census.gov/topics/genealogy/2010surnames/>

ACKNOWLEDGMENTS

The author like to thank Westat management for supporting his participation in SESUG 2024.

RECOMMENDED READING

Raithel, Michael A. 2017. **Did You Know That? Essential Hacks for Clever SAS Programmers: Over 100 Essential Hacks to Make Your Programs Leaner, Cleaner, and More Competitive.** Bethesda, Maryland: Michael A. Raithel

Available: http://www.amazon.com/Michael-A.-Raithel/e/B001K8GG90/ref=ntt_dp_epwbk_0[®]

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Michael A. Raithel
Westat, Inc.
MichaelRaithel@westat.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

APPENDIX A – SUBSET A LARGE DATASET WITH A FORMAT.SAS

This is the entire program that was annotated in the earlier section of this paper. You might consider copying it into one of your own SAS programs, customizing it, and then running it to facilitate faster subsetting of your organization's very large SAS datasets.

```
/* ***** */
/* Specify a large number of catalog page buffers for better performance */
/* ***** */
options cbufno=20;
options bufsize = 64k bufno=20;

/* ***** */
/* Specify the SAS data library containing the data sets to be processed */
/* ***** */
libname datalib "I:\Important Project\Production Directory";

/* ***** */
/* Specify a temporary file to hold PROC FORMAT statements */
/* ***** */
filename buildfmt temp;

/* ***** */
/* Read the file of ID variables to be matched, create the PROC FORMAT */
/* statements and store them in the Temp file */
/* ***** */
data _null_;
set datalib.apisurnames(keep=surname) end=eof;

file buildfmt;

retain doublequote '"';

/* Create first part of PROC FORMAT */
if _n_ = 1 then do;

    put "proc format;";
    put "value $APISurnames";
```

```

end;

/* Create name/value pairs of the format */
bigline = doublequote || strip(upcase(surname)) || doublequote || " = " ||
           doublequote || "YES" || doublequote;
put bigline;

/* Create ending part of PROC FORMAT */
if eof then do;
    put ";";
    put "run;";
end;

run;

/*****
/* %INCLUDE the Temp file to create the format */
*****/
%include buildfmt;

/*****
/* Create a file of API-named respondents from the Census Surnames SAS      */
/* data set                                                                    */
*****/
data datalib.API_Respondents;
set datalib.censussurnames(where=(put(strip(upcase(surname)), $APISurnames.)
                                     = "YES"));

run;

```