Common Sense Tips and Clever Tricks for Programming with Extremely Large SAS® Data Sets

Kathy Hardis Fraeman, United BioSource Corporation, Bethesda, MD

ABSTRACT

Working with extremely large SAS data sets – where the numbers of observations are in the hundreds of millions – can pose many challenges to the SAS programmer. Simple SAS code, including programs that involve basic DATA step programming and ordinary PROC SORTs, can become extremely difficult to run on such large data sets. Encountered problems range from extremely long run times, exceeding available memory, and seemingly unexplainable and random I/O errors that cause previously working programs to crash unexpectedly. This paper discusses common sense tips and clever tricks developed in Base SAS to improve the programming process with these extremely large SAS data sets. SORT/MERGEs on large data sets can be avoided with a trick with using SAS formats and hash tables, and the SAS options SGIO (Scatter-read/Gather-write Input Output) and BUFNO can be combined to avoid I/O errors that sometimes occur when reading extremely large SAS data sets. These and other programming tips and techniques were developed over many years' experience of writing program to analyze extremely large SAS databases consisting of medical insurance claims, inpatient hospitalization, and electronic medical records (EMR) data.

INTRODUCTION

When I first started programming with SAS about 30 years ago, I considered a SAS data set to be large if it had thousands or tens of thousands of observations. The first part of my career was spent using SAS to manage and analyze occupational epidemiologic data, where the study data were manually abstracted from old hard-copy employment history files. A large occupational epidemiology study cohort would consist of hundreds, to sometimes thousands, of employees. I then worked for many years as a SAS programmer with clinical trials data, where a large clinical trial would consist of the Case Report Form (CRF) data for typically hundreds of patients.

For the past seven years I have returned to my epidemiologic roots and am again doing SAS programming and data analysis for epidemiologic studies. However, times have changed, and the data used to conduct epidemiologic studies are no longer always being abstracted manually from hard-copy paper files. Epidemiologists now have access to many extremely large, de-identified electronic databases, including medical insurance claims databases, inpatient hospitalization billing records, and electronic medical records (EMR) data. Many different types of epidemiologic studies can be conducted using these extremely large databases.

These health-related databases are extremely large, by anyone's definition. I work with data sets that have hundreds of millions of observations and are over 50 GB in size. I have written relatively simple SAS programs that can take up to 24 hours to run. Over the past seven years of using SAS to analyze these extremely large data sets, I have run out of disk space, run out of memory, crashed the SAS server, randomly gotten unusual SAS I/O error messages while running programs that worked just fine the day before, and have taxed my available SAS resources to the limit.

This paper discusses some of the common sense tips and clever tricks I've developed over these past several years of working with these extremely large data sets. Some of these tips may seem obvious and are very simple to implement. One of these tricks incorporates some obscure SAS options that only a computer scientist could truly understand. The tips and tricks presented in this paper are not the only SAS techniques that can be used when working with extremely large data sets. For example, I don't either use indexes or compress my data, as those techniques are not appropriate for the types of data with which I work, or for the analyses I do.

MAKE THE SAS DATA SET SMALLER

The first step in working with extremely large data sets is to – if possible -- make a another new version of the original data set that is as small as possible. This step should be done as quickly as possible in the programming process. The ways to make a SAS data set smaller include deleting any unnecessary observations, and dropping any unnecessary variables. Changing the characteristics of data set variables that need to be in the data set, such as reducing the variables' lengths, also has a tremendous impact in reducing the size of SAS data sets.

DELETE OBSERVATIONS

One of the first steps involved in processing insurance claims data is to use a "subsetting IF" statement to only keep the insurance claims of interest. These claims of interest are usually for specific drugs, identified by an NDC code variable (NDC_CODE), or for specific medical procedures identified by a procedure code variable (PROC_CODE). These claims are identified with flag variables because, in actual claims data, there may be hundreds of different code values, often found in arrays of different code variables. These flag variables are also very useful in subsequent data analyses.

```
data out.smaller_insurance_claims_file;
     set in.EXTREMELY_LARGE_INSURANCE_CLAIMS_FILE;

     label drug_flag = "Drug code of interest"
           proc_flag = "Procedure code of interest";

     if ndc_code in (11111111111,22222222222) then drug_flag = 1;
     else drug_flag = 0;
     if proc_code in ("J1111","J2222") then proc_flag = 1;
     else proc_flag = 0;

     if sum(drug_flag, proc_flag) > 0;

     run;
```

The example above uses the "subsetting IF" statement to subset a data set based on a newly created, derived variable. A "subsetting WHERE" statement can also be used at the beginning of a DATA step to subset an input data set based on previously existing variables already in the data set.

VARIABLES TAKE UP "LESS SPACE" BY ADJUSTING THEIR LENGTH

The two new flag variables created in the example above don't need to have the SAS default length of 8. They can have a minimum length of 3. This little programming step may not seem like much, but it will have a very significant effect on the output data set's size, especially if the output data set has millions of observations and has hundreds of these flag variables.

```
data out.smaller_insurance_claims_file;
     set in.EXTREMELY_LARGE_INSURANCE_CLAIMS_FILE;

     label drug_flag = "Drug code of interest"
           proc_flag = "Procedure code of interest";

     length drug_flag proc_flag 3.;

     if ndc_code in (11111111111,22222222222) then drug_flag = 1;
     else drug_flag = 0;
     if proc_code in ("J1111","J2222") then proc_flag = 1;
     else proc_flag = 0;

     if sum(drug_flag, proc_flag) > 0;

     run;
```

Character variables can have an even shorter length than a numeric flag variable, although character flag variables are more difficult to use. The length of existing variables in an extremely large data set can also be reduced, but only after testing that no data will be lost in the process. The lengths of large character variables

can also sometimes be made smaller after determining the maximum length of the character variable actually found in the data.

GET RID OF VARIABLES THAT AREN'T NEEDED

If you're not going to use all of the variables in on original large data set, use either the KEEP = or DROP = options to only keep the variables you need. Either option will work, but I personally like to use the KEEP = option because it indicates the names of the variables in the data set. Sometimes using the DROP = is more efficient when "KEEP"-ing many more variables than "DROP" -ing. The KEEP = option can also be used with a MERGE statement, in addition to the use of the SET statement given in the example below.

```
data out.smaller_insurance_claims_file;
     set in.EXTREMELY_LARGE_INSURANCE_CLAIMS_FILE
         (keep = pat_id claim_date ndc_code proc_code);

     label drug_flag = "Drug code of interest"
           proc_flag = "Procedure code of interest";

     length drug_flag proc_flag 3.;

     if ndc_code in (11111111111,22222222222) then drug_flag = 1;
     else drug_flag = 0;
     if proc_code in ("J1111","J2222") then proc_flag = 1;
     else proc_flag = 0;

     if sum(drug_flag, proc_flag) > 0;

     run;
```

%SQUEEZE THE LARGE DATA SET

Sridharma (2006) published a macro to reduce the size of a large data set by finding the minimum lengths required for both numeric and character variables in a SAS data set. The paper also includes another macro %DROPMISS to identify and drop SAS variables that have only missing or null values.

MANAGE LARGE TEMPORARY DATA SETS

Every DATA step in a SAS program creates a temporary data set in the WORK directory, and these temporary data sets remain in the work directory until the SAS session – either interactive or batch – is ended. These temporary data sets can also be deliberately deleted with PROC DATASETS. Having lots of extremely large temporary data sets in the WORK directory can take up a lot of space and can hinder running programs, especially when the space is needed to do PROC SORTs.

When writing SAS programs with multiple DATA steps that involve extremely large SAS data sets, try to limit the number of these DATA steps by consolidating the programming that done in each step. Use DATA step programming efficiently, while making sure that the logic involved in the DATA steps is not compromised.

SORTING LARGE DATA SETS

Sorting extremely large SAS data sets should be avoided whenever possible. Sorting extremely large SAS data sets takes a very, very long time and requires a lot of free disk space in the work directory. The rule of thumb when running SAS under Windows is that a sort requires up 4 times the size of a SAS data set of free space on the work disk space. For example, sorting a 25GB SAS data set will require 100GB of space in the work directory, and SAS will run out of resources during the sort if that space in not available. The extra disk space required for the sort will be freed up after the SORT, but the disk space must all be available during the sort.

Although running out of disk work space is the most common reason for the SORT of an extremely large data set not to work, SAS can also run out of memory while running large sorts. The SORTSIZE and MEMSIZE options can be adjusted to increase memory for a sort, but you must be careful to leave enough memory for SAS

overhead and the operating system. Craver (SAS Global Forum 2009) provides a discussion on how to optimize the SORTSIZE and MEMSIZE options for both Unix and Windows servers.

USE SAS FORMATS TO AVOID SORT/MERGES

Using PROC FORMAT to avoid SORT/MERGES is very useful for situations where a "code" variable in a large SAS data set is used as a "key" variable in a separate reference data set that has one observation for each of the code values. Insurance claims and hospitalization data have many of these types of code variables, such as drug NDC codes; disease diagnosis ICD9 codes; ICD9, CPT4, and HCPCS medical procedure codes; and medical service provider codes. The value of each code may be repeated numerous times in the extremely large data set, and additional detailed information corresponding to the value of each code will be repeated only once in the reference data set.

In the following example, the very large SAS data set is named LARGE_DRUG_FILE. This SAS data set has variables to identify the patient (PATIENT_ID), the date on which the patient's drug prescriptions were filled, and the drug NDC code (NDC_CODE) associated with the filled prescription. This data set has one observation for each filled prescription for each patient, and is sorted by patient and drug prescription date.

The reference file is named DRUG_REFERENCE_FILE. It has one observation for each possible NDC code currently used by pharmacies. The ***inefficient way*** to combine information from the very large data set and the reference data set would be:

```
proc sort data=LARGE_DRUG_FILE;
     by ndc_code;
     run;

proc sort data=IN.DRUG_REFERENCE_FILE out=DRUG_REFERNCE_FILE;
     by ncd_code;
     run;

data LARGE_DRUG_FILE_DETAILS;
     merge LARGE_DRUG_FILE (in=d)
           DRUG_REFERENCE_FILE
           ;
     by ndc_code;
     if d;
     run;

proc sort data=LARGE_DRUG_FILE_DETAILS;
     by patient_id drug_date;
     run;
```

The faster and much more efficient way to get information from the drug reference file into the extremely large patient drug file, while avoiding all the PROC SORTs is as follows:

```
/*************************************************************/
/*  Create a format for drug name from the Drug Reference File
/*************************************************************/
data drgfmt (keep = start label fmtname);
     set in.DRUG_REFERENCE_FILE;
     length label $ 30;
     start = ndc_code;
     label = drug_name;
     fmtname = "drgfmt";
     run;

proc format cntlin=drgfmt library=library;
     run;

data NEW_DRUG_FILE;
     set LARGE_DRUG_FILE;
     length drug_name $ 30;
     label  drug_name = "Drug Name";
     drug_name = put(ndc_code, drgfmt.);
     run;
```

Multiple formats can be made using that single code as the “START” value of a format, depending on the amount of information found in the reference file.

USE HASH TABLES TO AVOID SORT/MERGES

Hash Tables are also a very powerful way to join multiple data sets without sorting. Here is a way to get information about the drug reference file into the extremely large patient drug file, as in the example above except using the Hash Table technique:

```
/*************************************************************/
/*  Join IN.DRUG_REFERENCE_FILE to LARGE_DRUG_FILE on the variable
/*   ndc_code using Hash Tables
/*************************************************************/
data NEW_DRUG_FILE (drop = rc);
     declare hash getdrug ();

     rc = getdrug.DefineKey  ('ndc_code');
     rc = getdrug.DefineData ('drug_name');
     rc = getdrug.DefineDone ();

     do until (eof1);
        set IN.DRUG_REFERENCE_FILE end = eof1;
        rc = getdrug.add ();
     end;

     do until (eof2);
        set LARGE_DRUG_FILE end = eof2);
        call missing(drug_name);
        rc = getdrug.find ();
        output;
     end;
     stop;
     run;
```

Many other Hash Table programming techniques are available to join multiple SAS data sets without sorting.

WHEN YOU MUST SORT

At some point extremely large SAS data sets will need to be sorted. Analyses of insurance claims data sets usually require the data to be sorted first by unique patient ID, and then sorted chronologically by date within patient ID. I usually do this PROC SORT as soon as the data set is as small as possible, and I try to only sort the data set once. The output of the sort is saved as a permanent SAS data set, and when sorting a permanent SAS data set, always use the OUT= option.

If the extremely large data set is too large to sort with the available resources of disk space and memory, I sometimes subset the large data set into smaller datasets prior to sorting. I usually subset according to date, but other methods can be used for subsetting. Once each of the data subsets have been sorted, I combine the sorted subsets back into a single data set in a DATA step, with a SET statement and a BY option.

DEVELOPING PROGRAMS TO WORK WITH LARGE DATA SETS

Developing and testing programs that use extremely large data sets as input can be challenging. I have had many occasions where a program run for hours, only have an error in the very last DATA step due to something as simple as a missing semi-colon, or to see errors when the actual input data had a value that was not anticipated in the program.

RUN WITH OBS=0

Before running any final SAS program that uses an extremely large data set as input, first run the program with the option OBS = 0 to make sure that the program's syntax is correct. If the program creates an output SAS data set, then be sure to look at a PROC CONTENTS of the output data set to make sure the variables in the new data set have the correct formats and labels.

Be very careful when sorting data using the OBS=0 testing option. If a PROC SORT does not have the OUT= option when sorting a permanent SAS data set, then the PROC SORT will overwrite the permanent data set with a sorted, but empty version of the data. All data will be lost. The SAS system option NOREPLACE will protect permanent SAS data sets from accidentally being overwritten in a PROC SORT without the OUT=option.

RUN WITH A RANDOM SAMPLE

Running a program with no actual input data will check the program's syntax for errors, but it will not permit testing the program with a range of actual data values. Once a program's syntax is verified, a random sample of observations in an extremely large data set can be used to further test the program.

SAS code to create a random 5% sample of a data set is given below:

```
data random_sample;
     set in.EXTREMELY_LARGE_INSURANCE_CLAIMS_FILE;
     if ranuni(123456789) <= .05;
     run;
```

WRITE MORE, YET SHORTER, PROGRAMS

When working with extremely large data sets, I never just have one program that takes the original extremely large data set as input and produces the final set of analytic reports as output. Not only would that single program take an extremely long time to run, it would be extremely hard to debug and maintain. As a general rule, I don't like any single SAS program to be more than 1,000 lines long, including comments.

I like to break down the processing and analyses of these extremely large data sets into separate programs, each program with its own conceptual purpose, and each with its clearly defined input and output SAS data sets. The SAS program header comment block shown below gives an idea of how I like to conceptually break down a series of SAS programs that will set up a study's analysis files. Each of these programs will have a specific function and a specific set of input and output data sets.

```
/*  Program:   Flag_Claims.sas
/*
/*  Function:  Adds flag variables to the patient and control claims files
/*
/*  Note:      1. PULL_DATA.SAS - Identify patients with disease and controls,
/*                pull their claims and their patient-level data
/*             2. DRUG_DATA.SAS - Create SAS data sets of the NDC codes for the
/*                drugs of interest
/*        -->  3. FLAG_CLAIMS.SAS - Put flags on the patient and control claims
/*                that will be used for the analyses
/*             4. PATIENT_VARS.SAS - Calculates "patient level" derived variables
/*
/*  Input:     IN.STUDY_PATIENT_CLAIMS
/*             IN.STUDY_CONTROL_CLAIMS
/*
/*  Output:    OUT.ALL_CLAIMS_FLAGS
```

UNUSUAL I/O ERRORS GENERATED WHILE RUNNING PROGRAMS ON LARGE DATA SETS

I have a very simple single DATA step program, similar to the program given in the first example of this paper, which simply reads in an extremely large claims data set and generates an output data set of the subset of claims needed for analysis. "Extremely large" in this case means over 750 million observations.

I observed the program would run successfully for about three times before I started getting the SAS error messages shown below:

```
ERROR: An I/O error has occurred on file ORIG.CLAIMS_DATA.DATA.
NOTE: The data step has been abnormally terminated.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: SAS set option OBS=0 and will continue to check statements. This may cause
NOTE: No observations in data set.
NOTE: There were 736329613 observations read from the data set ORIG.CLAIMS_DATA.
WARNING: The data set DER.CLAIMS_FLAGS may be incomplete.  When this step was
stopped there were 167190 observations and 19 variables.
WARNING: Data set DER.CLAIMS_FLAGS was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           2:06:52.80
      cpu time            49:05.96
```

If I ran the program again, the number of observations read from the data set ORIG.CLAIMS_DATA would get smaller in the resulting error message. I would continue to get this error message, with successively fewer observations being read in, until the SAS server was rebooted. After the reboot, I could then again run the program about three times without any errors before I started getting the same error messages.

SCATTER-READ/GATHER-WRITE INPUT/OUTPUT (SGIO)

SGIO is SAS option that must be implemented in the SAS configuration when SAS is opened up, not as an OPTION statement in a SAS program. Using SGIO allows SAS I/O to bypass the system file cache and is intended to speed I/O performance when dealing with these extremely large data sets. According to the SAS technical support website, several conditions must be met for SGIO to work:

- The data being read must be on a 4KB or multiple 4KB pagesize
- Access to the file must be sequential, not in update or random mode
- The data cannot be a Version 6 SAS data set

The SAS technical support website also says that the larger the data set being read, the better SGIO will work, and data sets over 1 GB are "good candidates" for SGIO.

To run a SAS program with the SGIO option in batch mode in a Windows environment, the command line will look like this:

"*SAS Location Pathname*\sas.exe" "*SAS Program Pathname**SAS Program Name.*SAS" **–sgio**

To run interactive SAS with the SGIO option in a Windows environment, the command line to open SAS will have –sgio at the end.

Once I started running the program with the SGIO option turned on at the beginning of my SAS session, and with an OPTIONS BUFNO = 0 statement at the beginning of the program, I could run the program successfully as many times as needed without any error messages, and without the need to reboot SAS. Setting BUFNO to 0 will increase the program execution time but will decrease the likelihood of getting I/O errors. The SAS LOG file for the program when working is given below:

```
NOTE: There were 768727929 observations read from the data set ORIG.CLAIMS_DATA.
NOTE: The data set DER.CLAIMS_FLAGS has 174030 observations and 19 variables.
NOTE: DATA statement used (Total process time):
      real time           2:08:33.78
      cpu time            1:01:04.90
```

When SAS programs are run using the SGIO option, any data sets being created either in the WORK library or in a permanent library will not appear in the library until the data sets are complete. You will not be able to monitor the progress of a program by looking at the increasing size of a newly created large data set.

CONCLUSION

Working with extremely large SAS data sets can be difficult, but sometimes simple steps to make the data smaller or clever tricks to avoid time-consuming PROC SORTS can make the process much easier.

REFERENCES:

Crevar, Judy, "How to Maintain Happy SAS9 Users", *Proceedings of the SAS® Global Forum Conference, 2009.*

Eason, Jenine, "Proc Format, a Speedy Alternative to Sort/Merge", *Proceedings of the 30th Annual SAS® Users Group (SUGI) Conference*, 2005.

Fickbohm, David, "Using the DATASETS Procedure", *Proceedings of the 31th Annual SAS® Users Group (SUGI) Conference*, 2006.

Gupta, Sunil, "WHERE vs. IF Statements: Knowing the Difference in How and When to Apply", *Proceedings of the 31st Annual SAS® Users Group (SUGI) Conference*, 2006.

Loren, Judy, "How Do I Love Hash Tables? Let Me Count The Ways!", *Proceedings of the SAS® Global Forum Conference, 2008.*

SAS Support, "Usage Note 12299: SGIO is beneficial on a case-by-case basis"
http://support.sas.com/kb/12/299.html

SAS Support, "Usage Note 4355: I/O Errors can occur with large values of BUFNO set with SGIO"
http://support.sas.com/kb/4/355.html

Sridharma, Selvaratnam, "How to Reduce the Disk Space Required by a SAS Data Set", *Proceedings of the 19th Annual North East SAS® Users Group (NESUG) Conference*, 2006.

Sridharma, Selvaratnam, "Splitting a Large SAS Data Set", *Proceedings of the 28th Annual SAS® Users Group (SUGI) Conference*, 2003.

ACKNOWLEDGMENTS

CONTACT INFORMATION

Please contact the author with any comments, questions, or additional ideas on how to work with large data sets:

Kathy H. Fraeman
United BioSource Corporation
7101 Wisconsin Avenue, Suite 600
Bethesda, MD 20832
(240) 235-2525 voice
kathy.fraeman@unitedbiosource.com